UNIVERSITY OF SUSSEX

Applying Temporal Chunk Signals Analysis to Measure Programming Competence by the Transcription of Java Program Code

Noorah Abdullah Albehaijan

January 2022

School of Engineering & Informatics

DISSERTATION SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE OF DOCTOR OF PHILIOSOPHY

I hereby declare that this thesis has not been and will not be, submitted in whole or in part to another University for the award of any other degree.

Signature: ……

Publications related to this thesis:

Albehaijan, N., & Cheng, P. C. (2019). Measuring Programming Competence by Assessing Chunk Structures in a Code Transcription Task. In C. Goel, Ashok, Seifert, Colleen and Freksa (Ed.), In *Proceedings of the 41st Annual Meeting of the Cognitive Science Society.* (Vol. 2, pp. 76–82). Montreal, Canada, 24 - 27 July, 2019.

Cheng, P. C., & Albehaijan, N. (2020). Some Determinants of Chunk Size in Sequential Behavior : Individual Differences in the Transcription of Alphanumeric Strings. In D. Dension, M. Mack, Y. Xu, & B. C. Armstrong (Eds.), In *Proceedings of the 42nd Annual Conference of the Cognitive Science Society* (Vol. 1, pp. 1164–1170). Austin: TX.

To my parents, Abdullah Albehaijan and Fawziah Alateeq.

To my husband, Fahad Alnafisi.

To my sisters and brothers, Maryam, Behaijan, Hussah, Mohammed, and Luluwah.

To my children, Ghadah, Tamim, and Abdullah.

Thank you very much for supporting and believing in me.

ACKNOWLEDGMENTS

First and foremost, I would like to thank ALLAH for providing me with the physical, spiritual, and health abilities to complete my PhD journey, without which it would not have been feasible to accomplish this research, which demanded endless time and effort.

It is an honour and a pleasure for me to express my heartfelt gratitude to Prof. Peter Cheng, my wonderful academic supervisor. His encouragement, motivation, comments, and advice have always been present in times of need. My supervisor gave me limitless support, which was vital in overcoming the impact of the Covid-19 pandemic that swept the entire world during my research. Prof. Cheng is the ideal supervisor that every PhD student would like to have, and he is someone I hope to emulate in my future academic career.

Acknowledgement would be inadequate if I did not express my gratitude to everyone who has helped me accomplish my thesis. Thank you very much to my parents, Abdullah Albehaijan and Fawziah Alateeq, as well as my brothers and sisters, Maryam, Behaijan, Hussah, Mohammed, and Luluwah, who took care of my small boy Abdullah for around two years. Thanks for their unlimited support and encouragement. A great thank you to my husband, Fahad Alnafisi, who has always been my backbone, and to my children, Ghadah, Tamim, and Abdullah, for giving me hope and motivation, as well as for being patient with me as I reached the end of my PhD journey. My great thanks to my wonderful lab mates at the University of Sussex, Dr Ronlad, Grecia, Munirah, Frances, Felix, Kinda, Rachael, Fiorenzo, Hadeel, Aron, and Ben, for their encouragement and for every single comment and piece of advice. Also, I greatly thank my friends in Brighton and Hove, Amani, Abeer, Duaa, Nermeen, Suha, Aydah, Munirah, Hussah, Eman, Reem, Mona, Sarah and their children, for making me and my children happy and shortening our long stay abroad apart from our family.

My gratitude also extends to my examiners, Prof. Gary Jones and Dr. Natalia Beloff, whose thoughtful comments and recommendations made a significant contribution to the improvement of my thesis. I also wish to express my gratitude to everyone whose names I haven't included.

Final thanks go to my sponsors, the Higher Education Ministry and Imam Abdulrahman Bin Faisal University in Saudi Arabia, for their unlimited financial and moral support. I aim to repay the favour by doing everything I can to serve my nation and my countrymen.

2022

Noorah Abdullah Albehaijan

SCHOOL OF INFORMATICS
UNIVERSITY OF SUSSEX
NOORAH ABDULLAH ALBEHAIJAN
DEGREE OF DOCTOR OF PHILIOSOPHY

Applying Temporal Chunk Signals Analysis to Measure Programming Competence by the Transcription of Java Program Code

ABSTRACT

This thesis investigates the basis for a novel method of quickly and efficiently assessing programming comprehension. It investigates the feasibility of assessing learners' mental chunk structures, and their temporal chunk signals, as a way of measuring their competence. The focus is on the Java programming language. The thesis investigates the feasibility of chunk-based measures in two different simple transcription tasks: view display, where stimulus is visible at all times; and hide and show, where the stimulus is only made visible when a participant presses a special button. University computer science students and faculty are the target group. Chunking theory is utilised to define three chunking measures of competence and to anticipate how they would vary across participants with different degrees of Java competence. The measures are as follows: (1) the number of characters transcribed per view (or the number of views) of the Java program code; (2) the time spent writing between the views; and (3) the duration of pauses before writing each written character. Ninety-six participants participated in the three experiments, transcribing on graphics tablets in experimental settings, and evidence of chunking's essential role in transcription tasks was revealed. Significant relationships were discovered between the chunking measures of competence and independent measures of Java competence (Java familiarity scores and students' final test marks (for the third experiment)). The third experiment included a longitudinal post-test component spanning three months of learning, in which changes to the mean scores in characters per view, writing-times, and pauses reflected the students' amount of learning.

Contents

DECLARATION 2

ACKNOWLEDGMENTS 4

ABSTRACT 6

Contents 7

Figures 15

Tables 20

Glossary 23

1 Introduction 24

1.1 Thesis Objectives 24

1.2 What is programming comprehension? 26

1.3 Why choose the Java programming language? 27

1.4 Typical tasks for measuring programming comprehension 28

1.4.1 Computer-Based Assessment (CBA) 29

1.5 Why transcription and temporal chunk signals? 30

1.6 Conclusion 34

2 Literature review: Studies of programming knowledge, programming tasks, and chunk structures & signals 36

2.1 Introduction 36

2.2 Programming knowledge 37

2.2.1 Different kinds of programming knowledge 39

2.2.1.1 Thesis design/classification of programming knowledge 43

2.2.2 Chunking in programming domains 45

2.2.3 Low- and High-competent programming knowledge structure differences 46

2.3 Programming tasks 48

2.3.1 Kinds of task that have been used to measure programming comprehension 49

2.3.1.1 Comprehension studies 51

2.3.1.2 Recall studies 55

2.3.1.3 Combined comprehension and recall studies 57

2.3.2 Assessment equipment (mode of production) 60

2.3.2.1 Paper and pen 60

2.3.2.2 Keystrokes (typing) 62

2.4 Chunk structures and pause analysis 63

2.4.1 Chunking as cognitive processing in general (other domains) 63

2.4.2 Pause analysis discovery 65

2.4.2.1 Relation between pauses and cognitive processes 66

2.4.3 Transcription and working memory 69

2.4.4 Signals and measures of chunk structures 70

2.4.4.1 Keystroke logging 70

2.4.4.2 Graphical Protocol Analysis: Why I decided to use GPA 71

2.4.5 Studies of the micro-behavioural temporal measure as reflection of chunk structures 72

2.5 Design space 79

2.5.1 Aspects that may influence programmers' cognitive processes 81

2.5.1.1 Stimuli presentation modes 82

2.5.1.2 Syntax highlighting 82

2.5.1.3 Indentation 83

2.5.1.4 Memorability 84

2.5.1.5 Flowcharts 85

2.5.1.6 Code scrambling 86

2.6 Summary 87

3 Experiment 1: Measuring Java comprehension 88

3.1 Introduction 88

3.2 Method 93

3.2.1 General experiment design 93

3.2.2 Classification of participants 94

3.2.3 Materials 94

3.2.4 Procedures 95

3.2.5 Stimuli design 96

3.2.5.1 Stimuli difficulty factors: Simple (S) & difficult (D) 98

3.3 Results 99

3.3.1 Independent measure of competence 100

3.3.2 Evidence of the role of chunking in a Java transcription task 102

3.3.3 Regression analysis for writing-times and view-times against view-numbers 109

3.3.4 Further evidence of the role of chunking in Java comprehension 110

3.4 Discussion 116

3.4.1 How this experiment builds on previous findings 117

3.4.2 The relationships between the dependent behavioural measures 118

3.4.3 Content analysis 120

3.4.4 Suggestions and future work 121

3.5 Summary 122

4 Experiment 2: Improved stimuli design 123

4.1 Introduction 124

4.2 Method 128

4.2.1 General experiment design 128

4.2.2 Participant classification 129

4.2.3 Materials 129

4.2.4 Stimuli design 130

4.2.4.1 Stimuli difficulty factors: basic (B) & advanced (A) 132

4.2.5 Procedures 132

4.3 Results and discussion 134

4.3.1 Results part 1 134

4.3.1.1 Independent measures of competence 134

4.3.1.2 The evidence for the role of chunking in the Java transcription task 136

4.3.1.3 Regression analysis for the behavioural measures against familiarity 141

4.3.1.4 PPCS normalization 149

4.3.1.5 Regression analysis for writing-times, view-times, and pauses against characters per view 151

4.3.2 Discussion part 1 154

4.3.2.1 Are the behavioural measures performing as expected? 154

4.3.2.2 This experiment's stimuli design 159

4.3.3 Results part 2 160

4.3.3.1 Stimuli complexity analysis 160

4.3.3.2 Discrimination between the two stimuli complexity levels across all behavioural measures 163

4.3.4 Discussion part (2) 175

4.4 Overall discussion 178

4.4.1 Suggestions and future work 181

4.5 Summary 182

5 Experiment 3: Longitudinal post-test study of the impact of learning on participants' behaviour 184

5.1 Introduction 184

5.2 Method 187

5.2.1 General experiment design 188

5.2.2 Participants 188

5.2.3 Materials 188

5.2.4 Stimuli design 188

5.2.5 Procedure 189

5.3 Results and discussion 189

5.3.1 Results part 1: Comparing pre-test and post-test results 189

5.3.1.1 Behavioural measures 189

5.3.2 Discussion part 1 195

5.3.3 Results part 2: The effectiveness of the behavioural measures at detecting learning gain 197

5.3.3.1 Independent measures of competence 198

5.3.3.2 Regression analysis for the behavioural measures against final exam marks 200

5.3.3.3 Regression analysis for the behavioural measures against characters per view 206

5.3.4 Discussion part 2 210

5.4 Overall discussion 214

5.4.1 Suggestions and future work 217

5.5 Summary 217

6 Discussion and conclusion 218

6.1 Introduction 218

6.2 Recap of chunking theory 220

6.3 How this thesis relates to previous work 221

6.4 Main findings 222

6.4.1 Does the longitudinal post-test study show learning gain over a short period of learning time? 225

6.4.2 Does the complexity analysis improve the discrimination of the stimuli? 226

6.5 Discussion of findings, limitations, and implications for future work 227

6.5.1 Why a freehand transcription (i.e., copying) task? 227

6.5.2 Transcription modes 228

6.5.3 What kind of stimuli were used? 229

6.5.4 Can the components of the chunks be revealed? 230

6.5.5 What kind of measure is suitable? 231

6.5.5.1 Extra evidence of chunking 233

6.5.6 Does normalization improve the measures? 233

6.5.7 Novel contributions 234

6.5.8 Limitations and suggestions for future research. 236

6.6 Real-world application of the approach and final conclusion 239

6.7 Summary 240

Appendices 241

Stimuli for practice items in Chapters 3, 4, and 5 241

Stimuli for PPCS in Chapter 4 241

Questionnaire for Chapter 3 242

Questionnaire for Chapter 4 245

Questionnaire for Chapter 5 248

References 249

Figures

Figure 2.1 An expert's generation of a pause profile 67

Figure 2.2 A novice's generation of a pause profile 67

Figure 2.3 Indentation example 83

Figure 3.1 Generation of view-numbers and writing-times profile: Expert 90

Figure 3.2 Generation of view-numbers and writing-times profile: Novice 90

Figure 3.3 Total number of views for participants, ranked in order of familiarity, across simple and difficult stimuli 102

Figure 3.4 Median writing-times for participants, ranked in order of familiarity, across simple and difficult stimuli 102

Figure 3.5 Median view-times for participants, ranked in 102

Figure 3.6 Mean view-numbers across stimuli difficulty types and levels of competence 104

Figure 3.7 Mean of median writing-times across stimuli difficulty types and levels of competence 104

Figure 3.8 Mean of median view-times across stimuli difficulty types 104

Figure 3.9 Correlation of view-numbers with familiarity across simple and difficult stimuli for all participants, low and high competence 107

Figure 3.10 Correlation of writing-times with familiarity across simple and difficult stimuli for all participants, low and high competence 107

Figure 3.11 Correlation of view-times with familiarity across simple and difficult stimuli for all participants, low and high competence 107

Figure 3.12 Relation of writing-times to view-numbers (simple stimulus) 109

Figure 3.13 Relation of view-times to view-numbers (difficult stimulus) 109

Figure 4.1 Total characters per view for participants across basic and advanced stimuli; participants are ranked by their familiarity scores 137

Figure 4.2 Median writing-times for participants across basic and advanced stimuli; participants are ranked by their familiarity scores 137

Figure 4.3 Median view-times for participants across basic and advanced stimuli; participants are ranked by their familiarity scores 137

Figure 4.4 Q3 pauses for participants across basic and advanced stimuli; participants are ranked by their familiarity scores 137

Figure 4.5 Mean characters per view across stimuli types and competence levels 138

Figure 4.6 Mean of median writing-times across stimuli types and competence levels 138

Figure 4.7 Mean of median view-times across stimuli types and competence levels 138

Figure 4.8 Mean of Q3 pauses across stimuli types and competence levels 138

Figure 4.9 Relation of characters per view to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli..... 143

Figure 4.10 Relation of writing-times to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli..... 143

Figure 4.11 Relation of view-times to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli..... 143

Figure 4.12 Relation of pauses to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli..... 143

Figure 4.13 Relation of writing-times to characters per view (basic (blue diamonds) and advanced (red crosses) stimuli) 152

Figure 4.14 Relation of view-times to characters per view (basic (blue diamonds) and advanced (red crosses) stimuli) 152

Figure 4.15 Relation of pauses to characters per view (basic (blue diamonds) and advanced (red crosses) stimuli) 152

Figure 4.16 (Basic) Total characters per view for participants across simple and complex stimuli; participants are ranked by their familiarity scores 164

Figure 4.17 (Advanced) Total characters per view for participants across simple and complex stimuli; participants are ranked by their familiarity scores 164

Figure 4.18 (Basic) Mean characters per view across stimuli types and competence levels 164

Figure 4.19 (Advanced) Mean characters per view across stimuli types and competence levels . 164

Figure 4.20 (Basic) Median writing-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores 167

Figure 4.21 (Advanced) Median writing-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores 167

Figure 4.22 (Basic) Mean of median writing-times across stimuli types and competence levels .. 167

Figure 4.23 (Advanced) Mean of median writing-times across stimuli types and competence levels 167

Figure 4.24 (Basic) Median view-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores 170

Figure 4.25 (Advanced) Median view-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores 170

Figure 4.26 (Basic) Mean of median view-times across stimuli types and competence levels 170

Figure 4.27 (Advanced) Mean of median view-times across stimuli types and competence levels 170

Figure 4.28 (Basic) Q3 pauses for participants across simple and complex stimuli; participants are ranked by their familiarity scores 172

Figure 4.29 (Advanced) Q3 pauses for participants across simple and complex stimuli; participants are ranked by their familiarity scores 172

Figure 4.30 (Basic) Mean of Q3 pauses across stimuli types and competence levels 172

Figure 4.31 (Advanced) Mean of Q3 pauses across stimuli types and competence levels 172

Figure 5.1 Total characters per view for basic stimuli across pre-test and post-test; participants are ranked by their exam marks 190

Figure 5.2 Total characters per view for advanced stimuli across pre-test and post-test; participants are ranked by their exam marks 190

Figure 5.3 Median writing-times for basic stimuli across pre-test and post-test; participants are ranked by their exam marks 191

Figure 5.4 Median writing-times for advanced stimuli across pre-test and post-test; participants are ranked ordered by their exam marks 191

Figure 5.5 Median view-times for basic stimuli across pre-test and post-test; participants are ranked by their exam marks 192

Figure 5.6 Median view-times for advanced stimuli across pre-test and post-test, participants are ranked by their exam marks 192

Figure 5.7 Q3 pauses for basic stimuli across pre-test and post-test; participants are ranked by their exam marks 193

Figure 5.8 Q3 pauses for advanced stimuli across pre-test and post-test; participants are ranked by their exam marks 193

Figure 5.9 Lines for linear best fit of characters per view to exam marks for all participants, basic (blue) and advanced (red) stimuli 201

Figure 5.10 Lines for linear best fit of writing-times to exam marks for all participants, basic (blue) and advanced (red) stimuli 201

Figure 5.11 Lines for linear best fit of view-times to exam marks for all participants, basic (blue) and advanced (red) stimuli 201

Figure 5.12 Lines for linear best fit of pauses to exam marks for all participants, basic (blue) and advanced (red) stimuli 201

Figure 5.13 Relation of writing-times to characters per view (basic (blue) and advanced (red) stimuli) 208

Figure 5.14 Relation of view-times to characters per view (basic (blue) and advanced (red) stimuli) 208

Figure 5.15 Relation of pauses to characters per view (basic (blue) and advanced (red) stimuli). 208

Figure 5.16 Pre-test and post-test familiarity scores for participants, ranked according to post-test familiarity scores..... 211

Tables

Table 1.1 Pros and cons of typical assessment tasks 29

Table 1.2 Paper and pen assessment (PPA) and computer-based assessment (CBA) pros and cons 29

Table 2.1 Kinds of programming knowledge 39

Table 2.2 Thesis programming knowledge classification 43

Table 2.3 Kinds of programming comprehension task 49

Table 2.4 Thesis design space aspects 81

Table 3.1 Stimulus versions (S1, S2, D1, D2) and the total number of characters and source code lines 97

Table 3.2 Correlations between competence measures (n=24, Pearson correlation, 1 tail, critical value is 0.472 at p<.01) 100

Table 3.3 Parameter of best-fit power relation for writing-times and view-times to view-numbers 109

Table 3.4 Coding scheme for various types of stimulus content (code content categories) 113

Table 3.5 Definition of each content analysis measure, S means simple and D means difficult 114

Table 3.6 Correlation between familiarity scores and number of breaks associated with each category (n=24, Pearson correlation, 1 tail, critical value is 0.472 at p<.01) 114

Table 4.1 Stimulus versions (B1, B2, A1, A2) 131

Table 4.2 Correlation between competence measures (n=51, Pearson correlation, 1 tail, critical value is 0.354 at p<.01, 0.273 at p<.05) 135

Table 4.3 Parameter of best-fit linear relation for characters per view, writing-times, view-times, and pauses to familiarity 145

Table 4.4 Parameter of best-fit linear relation for writing-times, view-times, and Q3 pauses to characters per view 153

Table 4.5 Proportions of the no. of characters for stimuli content categories 162

Table 4.6 Mean total number of characters per view across low- and high-competence groups and across stimuli difficulty and complexity factor for both (analysis parts 1 & part 2) 166

Table 4.7 Mean writing-times across low- and high-competence groups and across stimuli difficulty and complexity factor for both (analysis parts 1 & 2) 169

Table 4.8 Mean view-times across low- and high-competence groups and across stimuli difficulty and complexity factor for both (analysis parts 1 & 2) 172

Table 4.9 Q3 Pauses across low- and high-competence groups and across stimuli difficulty and complexity factors for both (analysis parts 1 & 2) 174

Table 5.1 Mean and SD for the difference between post-test and pre-test behavioural measure values across all behavioural measures over basic and advanced stimuli, HS=hide and show, VD=view display, WT=writing-times, VT=view-times 190

Table 5.2 Correlations for post-test - pre-test scores between various behavioural measures (n=21, Pearson correlation, 1 tail, critical value is 0.5034 at p<.01) 195

Table 5.3 Correlation between competence measures with both familiarity and exam marks (n=21, Pearson correlation, 1 tail, critical value is 0.503 at p<.01, 0.369 at p<.05) 198

Table 5.4 Relationship of characters per view, writing-times, view-times, and pauses to familiarity and exam marks 199

Table 5.5 Relationship of characters per view, writing-times, view-times, and pauses to exam marks 202

Table 5.6 Relation of writing-times, view-times, and pauses to characters per view 209

Table 6.1 Relationship of view-numbers, characters per view, writing-times, pauses, and view-times to familiarity and exam marks 222

Table 6.2 Mean post-test and pre-test values and the percentage of change in the values of behavioural measures across all the behavioural measures over basic and advanced stimuli 225

Table 6.3 Mean values of all the behavioural measures across all participants, over different stimuli difficulty (basic, advanced) and complexity (simple, complex) levels 226

Glossary

Term	Definition
Breaks	Refers to the time between stopping writing and starting writing again, which could include one or more views.
CBA	Computer-based assessment.
Char per view	The total number of characters per view, calculated by dividing the total number of characters of a stimulus by the view-numbers per trial.
Chunk	Collection of concepts in memory that have strong associations to one another and much weaker associations to other chunks concurrently in use (Cowan, 2001).
Complexity Analysis	The stimulus syntactic density (punctuations or text).
Content Analysis	Discovering the identity of each stroke.
Difficulty factor	Different levels of undergraduate Java modules taught to informatics students at Sussex University at the beginning and end of the year were considered when designing the stimuli.
Familiarity scores	On a 5-point Likert scale, participants rate their familiarity with each item in the stimulus.
HS	Hide and Show: where the stimulus is only made visible when a participant presses a special button.
Pause	Time durations before start writing each stroke.
PL	Programming Language.
Post-test	After learning the Java module.
Pre-test	Before learning the Java module.
PPA	Paper and pen assessment.
PPCS	Participants Preferred Cluster Size.
VD	View Display: where stimuli are shown all the time.
Views	Refers to any time a person presses the special button to display the stimuli (i.e., there could be several views without writing).
View-times	The length of each look at the stimulus, obtained by subtracting the time when they release the special button from the time when they press the button.
View-numbers	The total number of views of the stimulus in a trial, obtained by counting the number of button presses to view the stimulus.
Writing-times	The time spent writing between two consecutive views.

1 Introduction

Chapter content:

- The overall thesis goals
- The definition of programming comprehension.
- Justification for the thesis programming language selection (Java).
- Examples of common programming comprehension assessment tasks.
- Justifying the relationships between transcription and temporal chunk signals.
- Conclusion.

1.1 Thesis Objectives

How can we quickly and accurately assess students' programming knowledge? In the UK, computer science is taught in more than 100 institutions. In Saudi Arabia, almost 70% of the 41 government and private universities have computing and informatics departments (Ministry of Education, 2017). Computer science is taught at A level in both the UK (Department of Education, 2016) and Saudi Arabia. The number of computer science students being assessed every year is thus substantial. As a result, there are an enormous number of assessments which must be processed throughout the academic year, in which huge effort is involved.

Chunking is a fundamental concept in cognitive science (Miller, 1956; Cowan, 2001). Shneiderman (1976, p.131) defined chunking as "a recoding process that human beings seem to do without conscious effort". Cowan (2001, p.144) defines chunks as "collection of concepts that have strong associations to one another and much weaker associations to other chunks concurrently in use". Cognitive science recognises that chunking underpins competence in general, and that the number and structure of the chunks differ between low- and high-competent individuals. Can the difference between the structure of chunks that low- and high-competent have in their memory be used to measure competence in programming? Previous work on measuring competence in mathematics

and in second language learners by Cheng and Rojas-Anaya (2007), Cheng (2014), Cheng (2015), and Zulkifli (2013) shows that behavioural measures based on temporal chunk signals can provide some competence measures which may be effective in simple freehand copying tasks. This thesis develops those methods and shows how effective they might be in measuring programming competence.

This thesis focuses on programming learners for several reasons: (1) computer programming is a rich knowledge domain that has different skills associated with it, it involves various subtasks, and consists of various types of knowledge (Pennington, 1987; Shneiderman, 1976); (2) programming is challenging and many students encounter difficulties in learning it, especially at the beginning (Martinez & Mead, 1988); (3) as Martinez and Mead (1988) have pointed out, students find answering programming questions difficult compared to other aspects of computing; (4) there is a great deal of inconsistency in marking programs, especially in those marked by hand, as markers usually make their own marking judgments, even if they are experts, with the same programs often gaining different marks (Higgins, Gray, Symeonidis, & Tsintsifas, 2005). Current methods for assessing programming comprehension commonly involve tasks such as composing or modifying a program. Such tasks are limited in various ways (as discussed below).

Two different freehand transcription tasks were used: *view display* (VD), in which the stimulus is visible at all times, and *hide and show* (HS), in which the stimulus is only made visible when a participant presses a button. Why use these two different transcription tasks? VD provides a temporal chunk measure (pauses) that has been established before in the previous work listed earlier, whereas HS provides the characters per view, view-numbers, writing-times, and view-times measures. HS measures were introduced and tried in this thesis because of the success of pauses in measuring competence in previous studies.

In brief, using the temporal chunk signals in the context of a freehand transcription task is a pioneering approach to measuring programming competence. It is made possible by obtaining rich data on the cognitive processes that take place when writing.

Chunking theory, within a simple transcription task, therefore, is used in this research in order to develop an approach to programming assessment. Thus, this thesis aims to answer the following question:

Can programming competence be measured by analysing patterns of chunking behaviour and temporal chunk signals in the task of program code transcription?

1.2 What is programming comprehension?

Pea and Kurland (1984, p.149) define programming as a "set of activities involved in developing a reusable product consisting of a series of written instructions that make a computer accomplish some task". Studies of programming comprehension have improved enormously over the last two decades. Programming comprehension studies the cognitive processes involved in programming, which help in different things such as assessing students' competence, and improving programming tools and training courses.

Shneiderman (1977, p.467) defined programming comprehension as "the recognition of the overall function of the program, an understanding of intermediate level processes including program organization and comprehension of the function of each statement in a program". Shneiderman focused on three essential features of programming comprehension: a program's overall function, intermediate processes, and understanding each statement. In addition to these programming features, this thesis emphasises the chunk structures feature (illustrated in section 2.2). Chunk structure is considered a critical aspect of programming comprehension, thus measuring manifestations of it may be a good general measure. This motivated the selection of this thesis's

approach of measuring programming comprehension by assessing cognitive chunk structures and temporal chunk signals.

1.3 Why choose the Java programming language?

A programming language (PL) is used to write sets of instructions (i.e., code statements) to accomplish specific functions, such as calculating factorials or converting Celsius to Fahrenheit and vice versa. While there are over 2,000 PLs, few of them are commonly used (Hemmendinger, 2017). Grocevs and Prokofjeva (2016) indicate that previously the C and PASCAL PLs were utilized in programming education, particularly to explain fundamental programming concepts; however, nowadays, Java and other high-level PLs are used.

Java was used in this research because: (1) Java is widely used; it is utilized in various domains such as education and industry. Especially relevant to this study is the fact that Java is the primary programming language taught to undergraduate students at the University of Sussex (i.e., many of this study's participants). (2) I needed to select a language representative of the kinds of knowledge that is presented in the literature review (section 2.2.1), which are generally considered important for programming – for example, a language that includes the main programming knowledge features such as controls, iterations, and functions. Consequently, there are many PLs which I decided not to employ, for instance HTML. HTML was considered, but rejected, because there are not enough control flow notions, notions of variables, or conditionals.

1.4 Typical tasks for measuring programming comprehension

Regarding traditional educational assessments, there are many tasks which are currently used in order to determine students' level of competence and to measure their programming comprehension. These tasks can be presented to students in diverse contexts, such as composing, debugging, fill-in-the-blanks, and multiple choice. It is important to discuss these tasks in order to gain an overview of programming assessments in the educational environment, and to show that these typical assessments are limited and time- and effort-consuming. For instance, with regards to the disadvantages of multiple choice, even if it is electronically marked: (1) it is a shallow measurement of students' knowledge; (2) the ratio of guessed answers is high (Ye & Salvendy, 1996).

Table 1.1 below consists of four rows which show the main programming assessment tasks (composing/modifying, debugging, fill-in-the-blanks, multiple choice), presenting each task's advantages and disadvantages in terms of the test-takers and makers, derived from the literature review and the researcher's own experience in programming marking.

Because of their nature, there is no single correct solution in programming assignments (Sitthiworachart & Joy, 2004). Tasks commonly used to measure programming comprehension include: (1) compose a statement (or more). For example, code for a particular function (i.e., method declaration or definition). (2) Modify a program. (3) Provide students with a program which includes errors but seems correct at first glance, and ask them to explain the errors and rewrite the program in the correct executable form. (4) Fill-in-the-blanks: present students with a program which has gaps and ask them to fill them in. (5) Multiple choice, where the questions are varied, such as: which of the choices is the correct output of the program; choose the number of the lines that include errors; give the value of a specific variable at a specific point in the code.

No.	Task	Practical Advantages	Practical Disadvantages
1	Composing (i.e. modifying)	- Easy and quickly questions development. - Freedom in choosing the method of answering questions.	- Time-consuming for both marker and student (slow). - No specific right answer (imprecise). - Inconsistent (i.e., inaccurate, unreliable) scores, need more than one marker. - Stressful for both markers and students. - Hard to mark automatically.
2	Debugging	- Quicker to mark than composing.	- No specific right answer (imprecise). - Time-consuming for both marker and student (slow). - Stressful for both markers and students. - Hard to mark automatically.
3	Fill-in-the-blanks	- Quicker, but a bit harder to mark. - Less stress in answering. - Can be marked automatically.	- Time- and effort-consuming for markers. - More difficult to mark than multiple choice (slower). - Need for tutor interference when automatically marking.
4	Multiple choice	- Quicker and easier to mark. - Can be marked automatically. - Less fatigue and stress for students. - Precise in term of choosing only ONE answer.	- Time- and effort-consuming for constructor. - Inaccurate or unreliable because it allows for guessing answers (unrealistic).

Table 1.1 Pros and cons of typical assessment tasks

1.4.1 Computer-Based Assessment (CBA)

No.	Method	Practical Advantages	Practical Disadvantages
1	PPA	- It is guaranteed in case of system/device crashes.	- Grading is a complex, tiresome task. - Time-consuming. - Produces fatigue and stress for both markers and students. - Always prone to errors and leads to inconsistencies. - Late feedback.
2	CBA	- Rapid. - Efficient and sufficient. - Requires less pedagogue interference. - Reduces markers' workload. - Aids students in decoding a program (syntax error is obvious). - Students can examine their code validity. - Markers can execute and debug students' programs.	- Needs very complex automated systems for grading which take lots of effort and preparation to develop, especially if they are manually coded.

Table 1.2 Paper and pen assessment (PPA) and computer-based assessment (CBA) pros and cons

There are two typical response media assessments which are paper and pen assessment (PPA) and computer-based assessment (CBA). The advantages and disadvantages of PPA and CBA are presented in Table 1.2 above. Kalogeropoulos, Tzigounakis, Pavlatou, and Boudouvis (2013) found that students achieved greater programming scores in CBA compared to PPA. Using PPA is prone to errors, as markers may, for instance, forget to add a mark or subtract one by mistake (i.e., human error), and furthermore PPA leads to inconsistencies in students' marks because different markers correct programs using different schemes (Higgins et al., 2005; Romli, Sulaiman, & Zamli, 2015). Moreover, feedback is slow for students using PPA (Tremblay, Gu´erin, Pons, & Salah, 2008; Truong, Bancroft, & Roe, 2005). Markers, although proficient, will have their own understanding of the marking system (Higgins et al., 2005). These issues may lead to unreliable, inconsistent, and inaccurate marking.

Kalogeropoulos et al. (2013) indicate that the use of CBA aids students in decoding a program as the compiler automatically highlights any syntax mistakes, making them obvious to the user. Regarding markers, it helps them execute, examine, and debug students' solutions.

As the amount of data is quite large, these methods are time-consuming. On the whole, a method that could save time and effort in reliably assessing programming competence is desirable. In brief, because of the above-mentioned disadvantages, this thesis focuses on the following question in the literature review chapter: **What computer-based methodologies can be used to measure programming comprehension reliably, rapidly and efficiently**?

1.5 Why transcription and temporal chunk signals?

In this thesis, *transcription* means that participants transcribe chunks of Java computer code presented on the screen by hand using a special pen and paper (attached to a graphics tablet). Why

transcription as a task? The main reason that a transcription task is used to measure programming competence in this thesis is that there has been previous success in applying this approach to measuring competence in various areas (Cheng & Rojas-Anaya, 2007; Cheng, 2014; Cheng, 2015; Zulkifli, 2013). Transcription can be done quickly and easily, because participants copy exactly the same code statements (i.e., the stimulus), which makes the cognitive process simple. Thus, their amount of knowledge is apparent. They do not need to compose a program themselves, which would involve editing the code (i.e., delete, write, rewrite, debug, run, rerun, etc.), which would produce noise; this noise would negatively affect the measurement. Any influence the participants' knowledge has may be hidden, and it will be difficult to precisely spot different chunk structures. Measures based on transcription can be extracted more easily. Transcription tasks might also be more reliable, because what I am trying to do is investigate the chunks which are held in the memory. Chunks in memory are mainly built whilst individuals are developing competence (Miller, 1956; Cowan, 2001).

Participants will only be asked to copy a portion of a Java program by hand. It may be surprising that I am proposing this, but there are good theoretical justifications for this method. Participants will be asked to transcribe a block of Java program code that consists of several lines.

Matsuhashi (1981, 1987) used real-time testing techniques in the writing domain. Matsuhashi used videotaping, timing, and think-aloud protocols to measure cognitive processes in writing. With regards to pause lengths, Schilperoord (1996) differentiated long pauses from short pauses, in that the former indicate a high effort process, whereas the latter indicate the opposite. Chase and Simon (1973, p.59) define short and long pauses as "long pauses would correspond to boundaries between successive chunks, while short time intervals between pieces would indicate that the pieces belonged to the same chunk in memory". Thus, pauses were used to symbolize cognitive processes that occur when writing.

It is important to note that what I mean by *Pause analysis* in this thesis is the analysis of the time durations before start writing each stroke. A shorter Q3 pause signifies participants processing a familiar stimulus (i.e., program statements), because when they look at the stimulus, they will grasp larger chunks, which indicates their level of familiarity. They will not pause for a long time while writing, which can be used as a signal that they have a higher level of expertise (i.e., they are experts). Meanwhile, a longer pause implies that participants are processing an unfamiliar program, as they will have smaller chunks, and they will pause for a longer time. As a result, this is an indication of a lower level of expertise.

In terms of the VD measure (i.e., pauses) and the 'pause analysis' technique, many studies, such as Cheng (2014), Cheng and Rojas-Anaya (2006, 2007), Cheng and Obaidellah (2009), van Genuchten, Cheng, Leseman and Messer (2009), and Zulkifli (2013) have shown that cognitive processes are apparent while using a copying task within the context of pause analysis. The pause analysis technique has been successfully used in various domains such as mathematics (Cheng, 2014) and with second language learners (Zulkifli, 2013), but has not been used before to measure competence in the programming domain. Such previous research thus motivated the idea of trying the method to measure programming comprehension.

In regards to the HS measures, response durations have been used in some studies to assess programming comprehension in a whole task, such as sets of multiple-choice questions, lasting for several minutes (e.g., Adelson, 1981, 1984; Ye & Salvendy, 1996). Here, the focus is on the time required for component activities within a task, rather than overall task time, and the examination of process durations that may be directly related to the chunks possessed by participants.

With regards to the use of the chunk signals in the current thesis, cognitive processes become visible via pause lengths between chunks, response durations, number of views of the stimulus, and/or the

number of characters per stimulus view. The chunk in this context might be, for a low-competence participant, a Java keyword such as CLASS, INT, or FLOAT. For a high-competence participant, it might be a line of code or a block of code. For example, a block (i.e., statements, more than one line of code) may be lines of a specific method, such as println(), max(), and random(), or lines of loop/iteration blocks, for instance 'for', 'do…while', and 'while'. In any case, it is possible to tailor the difficulty of the test items to the level of the test-takers. For example, the examiner may design a stimulus with a basic syntactic level and understandable key words such as PUBLIC and SYSTEM, or introduce programs that include sophisticated syntax in order to be able to differentiate test-takers' competence levels. Therefore, participants may be unfamiliar with some program statements, but at the same time they can still copy them.

Measuring participants' differing levels of programming expertise is still applicable despite the fact that the participants are copying the same stimulus. Because participants with varying degrees of expertise do not all have the same level of familiarity with the stimulus, the transcription process is not easy for everyone. Participants' familiarity with the stimulus will cause them to see the stimulus differently, thus they will process it in different ways depending on their competence level. Throughout this thesis, the design of alternative stimuli is intended to improve test accuracy, i.e., produce better correlations or R-squared values (i.e., maximise the disparity in participants' levels of programming competence), and this is done by considering stimulus design factors such as the level of syntax density.

In summary, transcription is considered a sufficient, efficient, quick, and simple writing task that can be utilized to examine cognitive processes, because of the quality and consistency of data it provides. In addition to these advantages, transcription was chosen as the main task in this thesis because it is a novel approach that has not previously been applied to measure programming comprehension.

1.6 Conclusion

The specialised approach used to assessing programming competence in this thesis has previously been used in a variety of disciplines, such as writing sentences and sequences of numbers and drawing diagrams (Cheng & Obaidellah, 2009; Cheng & Rojas-Anaya, 2005, 2006, 2008; van Genuchten & Cheng, 2010; van Genuchten, Cheng, Leseman, & Messer, 2009), copying mathematical equations (Cheng & Rojas-Anaya, 2007; Cheng, 2014, 2015), and copying second language sentences (Zulkifli, 2013). These studies have shown that the analysis of chunk signals might be used to measure competence in these domains. The goal of this thesis is to determine whether or not this approach can be applied to the programming domain. As programming requires considerable mental effort for both students and examiners, developing a novel tool that quickly and efficiently – but reliably and accurately – assesses programming competence would be highly worthwhile.

To conclude, program code transcription, within the context of freehand writing, is adopted in this thesis. The overall question is: can programming competence be measured by analysing temporal chunk signals in the task of program code transcription?

The general objective of this thesis is:

To investigate a potential novel method that quickly and efficiently – but reliably and accurately – assesses programming comprehension by analysing cognitive chunk structures using behavioural measures during the activity of Java code transcription.

This thesis investigates various factors, including:

a. The freehand transcription technique.

b. The two stimulus display techniques: view display (VD) and hide and show (HS).

c. The design of the stimuli to be transcribed, with the aim that they are able to effectively, reliably, and accurately differentiate levels of competence. The factors considered include: programming language, size and length of the stimulus, and different features of the stimulus design, such as whether indentation is included and whether standardisation of font and colour is applied.

d. The possible cognitive process measures that can be used to access Java programming competence. These are independent measures, such as education level, general programming competence, Java competence, stimuli familiarity scores, and students' final exam marks. And dependent measures, such as characters per view (i.e., view-numbers), writing-times, view-times, and $pause_{Q3}$.

e. The normalization process, in order to eliminate the effect of participants' different working memory capacities (if any) on the behavioural measures used to measure programming comprehension.

f. The impact of the longitudinal post-test study (i.e., a complementary approach to testing the effectiveness of the thesis approach) on participants' behaviour.

This thesis has five further chapters, the first of which is a review of the literature. Three experiments were carried out and are addressed in Chapters 3, 4, and 5. The first experiment uses HS measures to assess Java programming comprehension. The second experiment makes use of a better stimulus design. The third experiment is a longitudinal post-test study. Finally, Chapter 6 provides an overview of the thesis and discussion of the findings.

2 Literature review: Studies of programming knowledge, programming tasks, and chunk structures & signals

Chapter content:

- Critical analysis to the concept of chunking and how it relates to programming:
  - Various programming knowledge categories.
  - Chunking in programming.
  - Programming knowledge at various levels of expertise.
- Critical analysis to various programming tasks:
  - Programming comprehension assessments tasks.
  - Assessment equipment.
- Chunk structure and pause analysis as techniques for measuring cognitive processes:
  - Chunking as a cognitive process.
  - Pauses analysis discovery.
  - Transcription and working memory.
  - Chunk structures signals and measures.
  - Micro-behavioural temporal measure as reflection of chunk structures.
- Thesis design space:
  - Design aspects that may affect programmers' cognitive processes.
- Summary.

2.1 Introduction

This thesis uses a freehand transcription task, involving VD and HS transcription, to measure participants' Java programming competence. Employing chunking theory and the use of temporal chunk signals in the context of a freehand transcription task is a pioneering technique in regards to assessing programming comprehension. It is made feasible by collecting extensive data on the cognitive processes that occur throughout the writing process. In brief, this research employs

chunking theory via a simple transcription task to build a method for assessing programming. This approach was motivated by the successful use of this technique in prior studies to evaluate participants' competence in other areas, particularly mathematics and learning a second language (Cheng & Rojas-Anaya, 2007; Cheng, 2014, 2015; Zulkifli, 2013). When compared to other approaches that have been used to measure programming competence (presented in 2.3.1), this approach is easier and faster, thus it saves both students and examiners time and effort.

This chapter covers the three main areas of research in the literature upon which this thesis builds and that motivated the research. These areas are (2.2) studies of programming knowledge, (2.3) programming tasks, and (2.4) chunk structure and pause analysis. These three areas are relevant to the research because they involve the main features of this study.

The overall research question for this thesis is:

Can programming competence be measured by analysing patterns of chunk behaviour and temporal chunk signals in the task of program code transcription?

Thus, the focus in this chapter is on the various types of programming knowledge, the various kinds of task that have been used to measure programming comprehension, clarifying the meaning of chunk structures and temporal chunk signals, and the importance of the latter in measuring competence in various domains, all of which address particular aspects of this overall research question.

2.2 Programming knowledge

In this thesis, the phrase 'programming knowledge' encompasses *programming organization* (Barfield, 1986; McKeithen, Reitman, Rueter, & Hirtle, 1981), *programming structure* (Shneiderman, 1977; Wiedenbeck, 1991), and *programming mental representations* (Adelson, 1981; Ye & Salvendy,

1996) – the latter refers to the way in which programming information is represented in the programmer's mind.

Computer programming involves different aspects, as presented in Pennington (1987), Shneiderman (1976), Soloway and Ehrlich (1984), and Shneiderman and Mayer (1979). In order to do any programming activities, such as composing a program's source code, or debugging it, programming knowledge needs to be acquired. The aspect that will be focused on in this study is knowledge about program structure. Programming knowledge in this thesis is defined as the structure (i.e., organization) of learners' programming *chunks* or *schemas*. Chunks and schemas are both considered collections of components. A *chunk* consists of various elements (i.e., components) of knowledge that are tightly associated with each other, and the elements within a chunk are weakly associated with elements in other chunks (Cowan, 2001). But a *schema* is another, wider knowledge type; it is a special and richer kind of chunk with more sophisticated structures, considered as a set of slots which are related to each other, each of which a human can fill with information (Anderson, 2000).

In this thesis, it is assumed that participants have already acquired their various amounts of knowledge, thus each participant's existing amount of programming knowledge will be measured. Accordingly, answering the question 'how do programmers acquire their programming knowledge?', as explained in Anderson's (2000) work, is outside the scope of this thesis.

2.2.1 Different kinds of programming knowledge

Classifications of Programming Knowledge Representations										
STUDIES	Semantic	Syntactic	Apparent / Concrete / Physical / Basic	Abstract / Advance	Text-Structure	Plan Knowledge	Discourse Rules	Logical	Conceptual	Objective
(Shneiderman, 1977)	**	**	*	*	*	*		*		
(Shneiderman & Mayer, 1979)	**	**	*	*	*	*		*		
(McKeithen et al., 1981)	*	*	**	**	*	*		*		
(Adelson, 1981)	*	*	**	**	*	*		*		
(Adelson, 1984)	*	*	**	**	*	*		*		
(Soloway & Ehrlich, 1984)	*		*			**	**			
(Barfield, 1986)	*	*	**	**	*	*		*		
(Pennington, 1987)	*	*	*	*	**	**		*		
(Ye & Salvendy, 1996)	*	*	**	**	*	*		**	**	**
THIS THESIS	*	*	**	**	*	*		*		

Table 2.1 Kinds of programming knowledge

Most research in this area studies kinds of programming knowledge, as presented in Table 2.1 above. Each column in the table represents a programming knowledge type and each row signifies a research study. The symbols * and ** are included in the table where a particular kind of programming knowledge is focused on in a particular research study: * is added when this particular kind of knowledge, e.g., *logical*, is referred to by the meaning only; for example, the phrase "dealing with objects in a logical way" refers to the logical type of knowledge, as in Shneiderman's (1977) study, without using the term *logical* as a particular name for a particular kind of programming knowledge. Conversely, ** is applied to studies where a specific kind of knowledge is referred to using the same terminology; for example, *objective* in Ye and Salvendy (1996). Each programming knowledge category presented in Table 2.1 above will be clarified in the following paragraphs.

First, the *semantic and syntactic* classification of programming knowledge was developed by Shneiderman (1977) and Shneiderman and Mayer (1979). Although the semantic/syntactic classification is used in all studies presented in Table 2.1 above, only Shneiderman (1977) and Shneiderman, Mayer, McKay, and Heller (1977) use the exact terms semantic/syntactic in their studies. Semantic is specifically related to the meaning of a particular PL's concepts, such as iteration and conditional. As a result, it is considered to be independent, as it is learned through meaningful learning and is not dependent on one specific PL. The semantic category proposed by Shneiderman and Mayer (1979) is split into three levels: (1) *low-level* notions of what a line of code does, such as what the data types are; (2) *intermediate* notions, for instance multiplying the array content, or developing a technique for finding the smaller of five values; and (3) *high-level* notions, for example binary searching and sorting strategies. Syntactic, on the other hand, is another form of information stored in long-term memory; it is related to the arrangement of elements within a language that make up well-formed sentences. It is more specific and detailed than semantic (Shneiderman & Mayer, 1979). In concrete terms, syntactic deals with features related to a specific PL, for instance Java-specific libraries, such as java.net (networking library), javax.swing.* and java.awt.* (graphics libraries), and java.lang.Math (maths library) (Deitel & Deitel, 2017). This thesis utilizes the general idea of a semantic/syntactic model, as it measures semantic knowledge, explained above, in terms of the general programming notions which are part of specific PLs (in this case Java), as well as syntactic programming knowledge, which is particularly dependent on Java.

The second category is *concrete* and *abstract*. Concrete relates to the understanding of an object's appearance, such as a specific device or program code. Abstract refer to the comprehension of the framework functionality, thus computer programming knowledge in this level may include, for example, algorithms, data structures such as lists and stacks, and object-oriented modules. Understanding these functionalities creates the functionality level of computer programming

knowledge. Regarding different levels of expertise, low-competent individuals focus on superficial characteristics (i.e., concrete) such as simple syntactic differences, whereas high-competent focus more on meaningful things such as underlying functions (i.e., abstract) (Adelson, 1981). Thus, the perceptions of people with distinguished levels of expertise are quite different, hence they will have different structures of chunks, which might give different durations of pauses and other temporal chunk signals.

Different studies differentiate concrete from abstract knowledge in distinct ways (Adelson, 1981, 1984; Barfield, 1986; McKeithen et al., 1981). For instance, program semantics, task plans, and algorithm plans are regularly considered abstract knowledge, whereas different languages' syntax, such as SWITCH structure, are commonly considered concrete knowledge (Ye & Salvendy, 1996).

Third, Pennington (1987) suggested that *text-structure* and *plan-knowledge* are fundamental in programming knowledge organizations. They differ in terms of the associations between program statements. Text-structure consists of three main types of program structure, each of which is a potential schema: (1) sequence (the control passes line-by-line in the program), (2) iteration (looping), and (3) conditional (if-then-else). Plan-knowledge is defined as different lines of code that should be grouped together in one block in order to achieve a specific aim (Pennington, 1987; Soloway & Ehrlich, 1984); it is mostly based on data flow relations. Pennington (1987) indicates that mental representations in computer programming are (1) procedural, which shows the flow of control in the program, or (2) functional, which demonstrates the hierarchical aim of the program.

Next, the *programming plans* and *discourse rules* categories were developed by Soloway and Ehrlich (1984), who second experiment focused on experts only. These are sources of chunk structure for high competent but are absent for low competent (Soloway and Ehrlich, 1984). Programming plans show any sequence of actions in the program; this category is pointed to by Soloway and Ehrlich

(1984) and Pennington (1987). Rules of programming discourse refer to programming conventions such as the name of a specific variable, which usually matches its function. Rules of discourse will not be focused on in this thesis, not only because they are not used in other studies as a programming knowledge category, but also because they are outside the scope of this project.

Lastly, the *logical*, *conceptual*, and *objective* programming knowledge classifications suggest another basis for different groups of concepts. The logic concept involves the significance and purpose of the presence of an object and deals with it in a logical process (Ye & Salvendy, 1996). Semantics of programming code are considered at the logical level. For instance, if a *for* loop is exploited in a Java program, the loop control will be comprehended at the logical level. The logical classification is indicated by most researchers, such as Adelson (1981, 1984), Barfield (1986), Mckeithen et al. (1981), and Pennington (1987), because it is a fundamental concept in computer programming. But only Ye and Salvendy (1996) use the term 'logical' in their programming knowledge classification. Finally, conceptual and objective programming knowledge classifications were also classified by Ye and Salvendy (1996). Programming knowledge is considered to be conceptual when it receives, converts, and produces information. That information is area-independent, in that it can be used to represent objects in diverse areas such as telecommunication. On the contrary, programming knowledge is supposed to be objective when it deals with matters like the usability of a specific interface, execution effectiveness, or framework restrictions. The conceptual and objective levels will not be utilized in this thesis, for reasons which are illustrated in next subsection.

2.2.1.1 *Thesis design/classification of programming knowledge*

No.	Category	Description
1	Basic (concrete/ physical/ apparent)	Low/intermediate level.
		Variable definition, data types, Java syntax
		Control flows (procedural/sequential – looping/iteration – conditional).
2	Advanced (function/ abstract)	High level.
		Data flow – functional.

Table 2.2 Thesis programming knowledge classification

It is not easy to formulate precise and consistent programming metrics that demonstrate programmers' mental representations. This is because programming is a very complex process and programmers comprehend programs at various levels based on their experience. In order to manage all of the complexity of the different kinds of knowledge associated with programming, rather than dealing with all of those sophisticated individual programming knowledge types as presented in Table 2.1, this project focuses on two programming knowledge classifications, as shown in Table 2.2 above. Firstly, *basic* (low/intermediate-level), which refers to the apparent (i.e., visible) programming characters of any object in the program, such as variables' and objects' names, different data types, and program controls (such as looping and (i.e., for), and conditioning (i.e., if..else)). Program controls considered vital as it affects the ability to divide the program into functional chunks (Curtis et al., 1984). This classification is referred to as *text-structure* in Pennington's (1987) work, *logical* in Ye and Salvendy (1996), *apparent* in McKeithen et al. (1981), *concrete* in Adelson (1981, 1984) and Barfield (1986), and *physical* in Ye and Salvendy (1996). Secondly, *advanced* (high-level) refers to invisible programming features and how basic characters function, e.g., what is the function of a specific *for* loop in this particular program? Usually, functions can be developed via more than one (explicit) method. When participants understand the function,

they will comprehend it as a whole chunk. For instance, to develop a calculator program, a *for*, *while*, or *switch* loop can be manipulated. Also, *advanced* points to data flow, comprehension, and functionality such as algorithms and data structures. This category is covered by the same term in McKeithen et al. (1981), and Ye and Salvendy (1996), and is called *plan-knowledge* in Pennington (1987). It is referred to as *abstract* in Adelson (1984) and Barfield (1986).

Ultimately, the *basic* and *advanced* categorization was considered to be enough for this thesis, as it is the first attempt to use transcription as a method of measuring programming comprehension. Also, it is considered general as it covers different abilities from low-level to high-level. High-competent understand programs as functional schemas (i.e., when there is a program which consists of declaring variables, declaring constructors and the main method, experts understand it as three blocks (1- variable declaration, 2- constructor, 3- main)). Low-competent, on the other hand, comprehend a program line-by-line, as demonstrated in previous studies. As this thesis deals with the two levels of programming knowledge. basic and advanced, the stimuli for this thesis's experiments were designed in terms of basic and advanced programming knowledge classification as well.

Summing up, most previous studies, such as Adelson (1984), Barfield (1986), and McKeithen et al. (1981), as presented in Table 2.1 above, classify programming knowledge into concrete and abstract. Hence, it was decided to follow their classification, but in this thesis these are indicated using the terms *basic* and *advanced*, which are also used in the experiments' stimuli design. Further, the details of Ye and Salvendy's (1996) scheme, such as conceptual and objective knowledge (as illustrated in section 2.2.1), are avoided in this project because they are beyond this thesis's scope.

2.2.2 Chunking in programming domains

Let us consider studies that use chunking in programming, as this supports the feasibility of utilizing temporal chunk signals (i.e., patterns of pauses and other temporal signals caused by chunk processes) in this thesis. Barfield and LeBold (1983, p.647) asserted that “Cognitive Skills, as measured by standard cognitive ability tests, are not shown to be helpful in predicting programming proficiency, whereas chunking appears to be a useful method for determining computer proficiency”. This emphasizes the feasibility of using chunking, and hence pause analysis and other temporal chunk signals generally, as a technique to measure programming comprehension. Davis (1984) points out that the chunks act together in order to achieve the complete program function. Therefore, the relationship between the chunks might indicate the program structure.

It was asserted by Curtis et al. (1984, p.83) that the concept of chunking is important in computer programming comprehension, specifically as an indicator of programming competence: “The nature of the concepts that a programmer has been able to build into a chunk provides one indication of ability”. The concept of chunking in programming was originally suggested by Atwood, Turner, Ramsey, and Hooper (1978), Brooks (1983), Curtis et al. (1984), and Davis (1984) as particular configurations of information in memory that are highly associated. These organisations are constructed in a hierarchical way (Atwood & Ramsey, 1978; Curtis et al., 1984). Brooks (1983) and Davis (1984) pointed out that programming learners study program code as blocks related to its function, and not statement-by-statement (Barfield, 1986).

The studies by Atwood and Ramsey (1978), Curtis et al. (1984), Brooks (1983), and Davis (1984) have clearly answered this thesis’s question ‘Can learners’ memory chunk organizations be used to measure programming competence?’ Hence, because of the demonstrated viability of utilizing

chunking as a cognitive method to discriminate between different levels of programming competence, it will be utilized in this research.

2.2.3 Low- and High-competent programming knowledge structure differences

The categories of basic and advanced are used in this thesis (Table 2.2) because these two main parts are considered crucial for programming knowledge. The majority of studies classify programming knowledge according to apparent and functional knowledge, as presented in Table 2.1 above. And even other studies that do not follow exactly the same classification have these two categories imbedded in their other classifications, as illustrated throughout the literature (subsection 2.2.1). Furthermore, this categorization is more suited to the kind of test used in this thesis. Furthermore, Atwood et al. (1978), Adelson (1984), and Ye and Salvendy (1996) have pointed to different kinds of programming knowledge and varying levels of programming expertise. Thus, it is fundamental to state the different organization of programming knowledge between low- and high-competent, as knowing these differences will help us in distinguishing the level of programming expertise.

There are three major differences between low- and high competent in terms of programming knowledge: abstract/concrete; chunk size and structure; and hierarchical structure. Firstly, according to my classification (Table 2.1), basic and advanced, it is assumed that experts have both basic and advanced knowledge, while low-competent have only basic knowledge. This is based on several studies which support this suggestion. McKeithen et al. (1981), Adelson (1981, 1984), and Ye and Salvendy (1996) agree that low-competent have apparent knowledge, whereas high-competent have functional knowledge. These studies are considered the base for the programming knowledge categorization used in this thesis, as illustrated in section 2.2.1.1.

Secondly, according to the idea of chunk size and structures, it is presumed that experts will comprehend a program as separate chunks with a size larger than one line of code. Low-competent will understand the program as separate lines, or even separate chunks of each line of code. This has been demonstrated by several researchers, such as Adelson (1981), Barfield (1986), Pea and Kurland (1984), Shneiderman (1976), and Soloway and Ehrlich (1984). High-competent have larger chunk sizes than the low-competent, and this is a significant indication of programming competence.

Lastly, another difference between high- and low-competent concerns the hierarchical structure of the program code (Atwood et al., 1978; McKay & Shneiderman, 1976; Shneiderman et al., 1977). In terms of memorization tasks, Adelson (1981) and Barfield (1986) indicate that high-competent can not only recall much more information than low-competent, but also they have better structure of chunks, which is also useful for answering this thesis's research question. In addition, it is assumed that high-competent understand the program better, which corresponds to a shallower hierarchical chunk structure. Put another way, the hierarchy of the more competent participants will tend to be shallower than lower competence participants for the same content (i.e., program code). This was concluded by Curtis et al. (1984) and Atwood et al. (1978), who recognized that high-competent programmers can recall information in greater depth of understanding than low-competent. Thus, low-competent individuals have smaller chunk sizes and a greater number of hierarchy layers, whereas high-competent have larger chunk sizes with a smaller number of hierarchy layers. This is of considerable value in this work because the measures of programming competence used in this thesis attempt to assess the differences between low- and high-competent hierarchical chunk structures.

To conclude, there are various perspectives on programming knowledge classifications, as illustrated earlier in this section; however, this thesis focuses on programming knowledge classified into basic and advanced.

2.3 Programming tasks

It is important to consider which tasks have previously been given to test-takers in order to assess their programming comprehension. Different programming tasks are used in order to measure programming comprehension, whether in educational assessments or empirical studies. Also, it is crucial to examine which measures have previously been used in the assessment of programming comprehension. This section considers approaches used in the literature in order to guide the design of my own approach, and to demonstrate the novelty of this thesis's methodology. These include methods using automated programming assessment systems and those used in research.

2.3.1 Kinds of task that have been used to measure programming comprehension

Study	Text-entry questions	Yes/no questions	Subjective rating	Flowchart	Composition	Debugging	Modification	Multiple Choice	Fill-in-the-Blanks	Summary	Think-aloud	Comments	Recall	Copying/Transcribing
(Shneiderman, 1976)													*	
(Shneiderman, 1977)		*	*		*	*	*	*				*	*	
(Soloway & Ehrlich, 1984)									*				*	
(Pennington, 1987)		*		*			*			*	*		*	
(Adelson, 1981)											*		*	
(McKeithen et al., 1981)											*		*	
(Barfield, 1986)													*	
(Ben Shneiderman et al., 1977)				*	*	*	*							
(Adelson, 1984)	*			*				*						
(Miara, Musselman, Navarro &	*		*					*						
(Rambally, 1986)	*		*					*						
(Ye & Salvendy, 1996)								*						
(Liu, Xu, & Cui, 2012)					*									
(Obaidellah, 2013)	*	*			*									
(Sarkar, 2015)	*													
(Dimitri, 2015)						*								
THIS THESIS			*											*

Table 2.3 Kinds of programming comprehension task

Researchers have used various types of programming task. In this section, different task types, for instance comprehension, modification, multiple choice, and recall, are outlined and considered. Table 2.3 above presents tasks that were used to assess programming comprehension in sixteen different studies. The rows represent the studies, while the columns denote task types. The symbol * is used to indicate that a specific task (e.g., modification) was considered in that specific study (e.g., Shneiderman (1977)). To sum up the table, comprehension studies have been more frequently applied than recall studies, and it is noteworthy that some of the recall studies applied comprehension tasks as well. No previous study has utilized transcription as a task to measure programming comprehension.

The measures which were used in the studies are presented in the following subsections, 2.3.1.1 (comprehension studies), 2.3.1.2 (recall studies), and 2.3.1.3 (combined comprehension and recall studies), and assessment equipment is discussed in section 2.3.2 (measures of programming comprehension).

Shneiderman (1976) specified four tasks which can be used to measure programming comprehension:

- Comprehension: measuring participants' understanding of an executable program
- Composition: giving participants specifications and asking them to write a program
- Debugging: giving them a program which contains errors and asking them to locate the errors
- Modification: in which participants are given a program written in an executable way and asked them to modify it

Different measurement techniques are required for each type of tasks, because of the specific differences in the character of the tasks (Shneiderman, 1976).

Pea and Kurland (1984) and Pennington (1987) outline very similar aspects that could be employed in order to measure low- and high-competent programming comprehension in a problem-solving manner: understanding the specifications, planning solutions, writing the program, comprehension, and debugging. These classification systems are similar to the tasks outlined by Shneiderman (1976) and Shneiderman and Mayer (1979), however they pay attention to the pre-task phase (i.e., understanding the specifications and planning the solutions). All of these classification systems are represented in Table 2.3, which focuses on actual problem-solving activities.

2.3.1.1 *Comprehension studies*

Various studies have used comprehension tasks to measure programming comprehension. Some studies used comprehension questions only, such as Adelson (1984), Miara et al. (1983), Rambally (1986), Sarkar (2015), and Ye and Salvendy (1996). Ye and Salvendy (1996) presented high- and low-competent graduate students with a program code written in the C programming language, then provided them with comprehension questions similar to multiple choice questions to measure their programming comprehension. The questions were written according to the five abstraction levels (as explained in section 2.2.1). They used the following measures: total test time in minutes, minutes spent to respond to each level of question, and overall number of wrongly answered questions. To calculate these, they utilized a software package. They found no significant distinction between high- and low-competent in total performance time. They found differences in function level but no significant differences in other abstraction levels. The overall conclusion emphasized previous studies' classifications of programming knowledge, in which high-competent had better abstract knowledge than low-competent.

Sarkar (2015) performed an eye tracking study which attempted to investigate the impact of syntax highlighting on programming comprehension times, and whether its effects can differentiate programmer expertise. The experiment was applied on graduate computer science students. Participant expertise was measured using a questionnaire. The participants were asked to mentally calculate the output of function definition lines of code, having been provided with a set of arguments (i.e., text-entry questions). Sarkar found that the participants were capable of disregarding highlighted keywords completely. Moreover, he found considerable improvement in task completion time when using syntax colouring for low-competent, but the impact of colouring became low when used with high-competent. This means that syntax highlighting assists low-

competent in comprehension but does not assist high-competent, as they already understand the syntax.

Rambally (1986) and Miara et al. (1983) calculated program comprehension using the mean and the variance of a comprehension quiz and subjective rating scores. They conducted their studies on novices (less than three years of programming experience) and experts (three or more years of experience) using the Pascal programming language. In Rambally's (1986) study, participants were given three printed programs to assess the comprehensibility of program code, as well as to try to make control structures more visible and simpler to keep track of, while utilising a colour-coded formatting approach. The functions were coloured in the first program. Different blocks were coloured in the second program, and the last was black and white. The study found that the minimum level of program comprehension was achieved by all participants when using the black and white scheme. The maximum comprehension level was obtained while using the functional coloured scheme, hence some colouring might have been related to function concepts. Thus, Rambally concluded that high-competent perform better than low-competent, and that colour coding techniques have a statistically apparent impact on programming comprehension. Meanwhile, Miara et al. (1983) conducted their study utilizing four levels of indentation (0, 2, 4, 6 spaces). They found that high-competent performed better than low-competent, and that in order to maximize program comprehension, a reasonable (2 or 4 spaces) level of indentation should be used. Participants found it difficult to work with the program without indentation as well as with too much indentation.

Soloway and Ehrlich (1984) studied high- and low-competent performance on a fill-in-the-blanks task (they removed one line of the code and replaced it with a blank), using two versions (plan and non-plan (programming rules were broken)) of the same ALGOL PL program. They used a Newman-Keuls test to calculate the difference and the percentage of subjects' correct answers. In brief, they

found that high-competent exhibited better performance than low-competent, and the plan versions had more correct answers than non-plan, even though the differences between the two versions were not constant. There was a significant difference in accuracy between the plan and non-plan versions. Finally, they indicated that high-competent became low-competent when PL rules were broken. On the whole, program plans and rules of programming discourse considerably influence programming comprehension.

Adelson (1984) asked low- and high-competent participants to answer questions based on a flowchart and a program code which were presented to them. Comprehension and question times were recorded using a stopwatch. Adelson used a Newman-Keuls test to measure the difference according to the following: task response time in minutes, error rates, and delay time in minutes. He found that high-competent understood the program abstractly, whereas low-competent understood it concretely. Multiple choice and fill-in-the-blanks tasks were also applied in studies such as those by Shneiderman (1977), Soloway and Ehrlich (1984), and Ye and Salvendy (1996).

Other studies have utilized composition, debugging, and modification. Liu et al. (2012) and Obaidellah (2016) used composition, whereas Dimitri (2015) utilized composition and debugging. Shneiderman et al. (1977) used all three tasks. Others combined one or more of these three tasks with comprehension questions, such as Obaidellah (2016) and Shneiderman et al. (1977). Obaidellah (2016) studied first-year computer students. She used two Java questions, and firstly presented students with a question asking about the name of operation. Secondly, she presented them with a flowchart or program code, then asked them to draw a flowchart or compose code. Obaidellah used hand marking and measured comprehension and composition of code and flowchart, by calculating students' success rate. She found that participants did better in interpreting the code into the flowchart than in the reverse task. Shneiderman et al. (1977) conducted four experiments; the first two utilized only novice and the other two used intermediate participants. Students' solutions were

marked by hand. In a composition test, participants were asked to draw a flowchart and compose a simple Fortran program code. There was found to be no difference between the groups' performance. Secondly, in a comprehension experiment, participants were asked to write the output of different input values. Again, the authors did not find any difference in participants' performance within the same form (including both flowchart and code). However, they found that the second version (which did not start with a flowchart) was more difficult than the first (which started with a flowchart). Thirdly, in a debugging experiment, the stimulus was a long program printed with its output. The authors again did not find any difference, as participants preformed badly with the flowchart. Finally, in a modification experiment, two types of flowcharts were utilized, detailed and high-level. Each participant was requested to record time spent on each task. The authors found no difference between the groups' performance. Therefore, what can be concluded from this series of experiments is that the use of a flowchart did not have a significant advantage in programming comprehension.

The previously listed comprehension studies suggested, for this thesis, that participants' programming expertise is best measured using a questionnaire, as Sarkar (2015) did. Obaidellah's (2016) study proposed evaluating programming skill for a group with comparable levels of competence (i.e., first-year computer students). Furthermore, a reasonable amount of indentation (2–4 spaces) could be implemented in this thesis, as Miara et al. (1983) concluded that this is helpful in increasing program understanding. These comprehension studies suggest that the overall time spent performing a task is a significant element in determining programming comprehension.

Shneiderman (1977, p.465) pointed out that the "Ability to debug, modify, hand simulate execution or respond to questions about the program all have their weaknesses as comprehension metrics". None of the previously described tasks are totally satisfying for a variety of reasons, one of which is that modifying a program can take a long time.

2.3.1.2 *Recall studies*

Memorization (recall) studies were conducted in the late 1970s and during the 1980s. In general, during a memorization task, test-takers are provided with a program code for a set length of time, then this program is removed and they are given another set amount of time to write/type it again. Adelson (1981), McKeithen et al. (1981), Soloway and Ehrlich (1984), Barfield (1986), and Pennington (1987) utilized memorization tasks as a practical method of judging programmers' comprehension. Shneiderman (1977) suggests that a recall task is a good measure of programming comprehension.

Researchers have utilized recall tasks to examine the distinction between high- and low-competent programming knowledge structures in different ways. For instance, Barfield (1986) and Soloway and Ehrlich (1984) asked participants to memorize then recall programs literally. Barfield (1986) undertook a recall experiment using three versions (executable, random lines, random chunks) of the BASIC program. He applied it to expert, intermediate, novice, and naïve participants. Participants were asked to memorize then recall the programs on a blank sheet of paper. Correctly written lines were marked, and the correct number of lines were recoded. Barfield utilized the ANOVA and Scheffe tests. Next, he calculated the mean and standard deviation (SD) of the number and percentage of recalled lines. His findings were similar to those of Shneiderman (1976), in that the greater expertise participants had, the more chunks (in the random chunks version) and lines (in the random lines version) were recalled. He also found that random chunks were recalled more than random lines, and that executable versions of the program were recalled more often than randomized ones, similar to what McKeithen et al. (1981) and Shneiderman (1976) found.

In contrast to Barfield (1986) and Soloway and Ehrlich (1984), Adelson (1981), in a multi-trial free recall experiment, presented each line of code separately to novice and expert participants in random order. They then had to recall as many lines as possible in any order. The pause that

occurred before each response was recorded. Adelson (1981) performed mean and variance analysis on number of items recalled, chunk size (i.e., amount of objects in a burst of recall), inter-response times (participants were given eight minutes to recall), and pauses before every response (if an item was evoked by less than a 10-second pause from the previous item, it counted as a member of the recent chunk). What Adelson found from participants' keyboard-typed responses and inter-response times was that the difference remained steady over trials, high-competent chunk size was larger and recalled more chunks than low-competent, and high-competent grouped the code lines according to the program they referred to. Thus, high-competent perceived the program lines' functionality, while low-competent grouped the lines in terms of their syntactic purpose. Hence, low-competent understanding of the program was shallow. The current research differs from Adelson (1981) in that I utilize a freehand copying (i.e., transcription) task of a precisely ordered Java program block of code, whereas Adelson utilized keyboard typing in a recall task of randomly ordered Polymorfic Programming Language code. Also, I calculate the pauses which are produced before each transcribed stroke in milliseconds and not before each entire response in seconds.

McKeithen et al. (1981) utilized Chase and Simon's (1973) technique with programming, as conducted by Shneiderman et al. (1977). McKeithen et al. (1981) conducted two experiments on novice, intermediate, and expert participants using the ALGOL W PL. In the first study, participants were asked to memorize executable and randomly ordered programs on blank paper. The authors conducted a 't' test and used, in their first experiment, the number of lines correctly written in their proper relative order as a measure. In their second experiment, recall orders were examined using R&R (Reitman-Rueter technique for deducing tree constructions from recall orders). Experts recalled more chunks than novices, as concluded by many studies, such as Chase and Simon (1973) and Adelson (1981). However, when they were presented with the scrambled version, both experts

and novices performed equally well, because experts found it difficult to recall the code lines when not grouped in meaningful chunks. In the second experiment, McKeithen et al. (1981) presented participants with reserved words and asked them to recall the words. They were asked to memorized them, then recall them out loud. The trials were tape recorded and data were not gathered until all participants accurately recalled all words. What they found was that novices recalled words that were apparently close to each other, such as having the same initial letter or being of the same length, story construction, and common language series, whereas experts sorted words based on their function.

This thesis's experimental stimuli (i.e., program code) were presented in executable order rather than random order. This suggestion was made based on the results of some previously listed studies, such as those by Barfield (1986), McKeithen et al. (1981), and Shneiderman (1976), which show that using a meaningful sequence of program lines or a set of reserved words is more effective than using a random order in distinguishing between high- and low-competent participants. Furthermore, Adelson's (1981) study implies that inter-response time is a critical factor in determining programming understanding.

2.3.1.3 *Combined comprehension and recall studies*

Some studies have combined comprehension and memorization tasks in one experiment, such as those by Pennington (1987) and Shneiderman (1977). Shneiderman (1977) performed two paper-based experiments, the first a modification and memorization test on participants of the same level using the Fortran PL. Participants were asked to modify, then comprehend and recall the program, and finally to answer subjective multiple-choice questions. Shneiderman recorded the number of correct multiple-choice answers. Mean scores were also calculated for participants' performance. The results revealed a strong relationship between recall measures (term grades and no. of lines attempted) and modification measures, using the Pearson correlation matrix. In the second

experiment, he calculated the average scores for the following: no. of correct multiple-choice answers, no. of lines attempted, no. of lines perfectly recalled, and the percentage of the functional accuracy of the program (note: only correctly written lines were marked correct). Eventually, it was concluded that a modification task is a valid measure of participants' programming comprehension. The second experiment was a comprehension and recall test, for which participants of slightly different levels were recruited. Shneiderman found that it is effective to use comprehension questions before asking participants to recall a program.

Pennington (1987) also conducted two experiments on experts utilizing the Fortran and Cobol PLs. In the first experiment, the first two trials started with yes/no comprehension questions and ended with a recall test. The researcher measured recognition speed by calculating the mean of the recorded responses and latency response times in seconds. She measured recognition accuracy by calculating the average of correct responses (i.e., comprehension rates) and incorrect responses (i.e., error rates). Error rates indicted that control flow was much easier to understand than dataflow, state, and function. In the second experiment, participants were asked to comprehend the program. Comprehension was tracked by recording the line of code that was located in the centre of the computer screen. Participants were then asked to type a summary and answer yes/no comprehension questions. After that, they were asked to complete a modification task, type a summary, and complete further yes/no comprehension questions.

The think-aloud technique is used in some comprehension, recall, and combined comprehension/recall studies. It was utilized by McKeithen et al. (1981), for instance, who asked participants to recall aloud a list of reserved words which they had been asked to study at the beginning of the experiment. It was also used by Pennington (1987), who requested that participants think aloud while understanding the program.

In conclusion, tasks commonly utilized in previous studies are comprehension questions (multiple-choice (Shneiderman, 1977; Ye & Salvendy, 1996), fill-in-the-blanks (Soloway & Ehrlich, 1984), write output values (Sarkar, 2015), write function/method name (Obaidellah, 2016), yes/no questions (Pennington, 1987)), composition and debugging (Dimitri, 2015), modification (Shneiderman, 1977; Shneiderman et al., 1977), and memorization and recall (Adelson, 1981; Barfield, 1986; McKeithen et al., 1981; Soloway & Ehrlich, 1984). None of these studies used transcription as a task to measure programming comprehension. Further, despite the fact that comprehension questions such as writing a specific output value and multiple-choice questions are easier to mark than composing or modifying a program code, they are not precisely accurate when compared with measuring programming performance in milliseconds. Although there are studies that have used time as a measure, such as Barfield (1986), Pennington (1987), Sarkar (2015), Adelson (1981), Rambally (1986), and Liu et al. (2012), their timescale is quite long (i.e., seconds and minutes). Thus, their idea of using time to measure programming comprehension is useful but not so useful for the purposes of this thesis, where the pauses are much shorter. In concrete terms, pauses in studies similar to this, such as Cheng (2015) and Zulkifli (2013), are along the lines of 100 milliseconds to a few seconds. What is noteworthy here is not only the time magnitudes, but also the time variation according to task type. It is clear that time measures can significantly vary when comparing, for instance, a composition task with a transcription task. In the former, time durations aggregate many actions, such as editing, adding, or erasing, whereas transcription within pause analysis assesses individual actions. Thus, time variability is lower in this context.

The above listed combined studies aimed to measure programming comprehension and differentiating levels of programming competence by combining comprehension and recall (i.e., memorization) tasks. One of the ways in which this inspired this thesis was that it led me to ask the participants to answer subjective questions after finishing the task, as Shneiderman (1977) did (in

this thesis, a subjective rating of the familiarity of the experiments' stimuli). Also, in these studies, researchers not only evaluated participants with varying degrees of programming competence, but also investigated individuals with similar levels of expertise, which inspired me to test two distinct programming competence level groups in the first and second experiments. In addition, as in the third experiment, students from the same level group were tested.

Ultimately, recall/memorization and comprehension tasks were not used in this thesis because they are harder (i.e., time and effort consuming) than copying (transcribing) and the examiner could obtain partial responses, meaning the results will not be sufficient, efficient, or accurate. The focus is on how to produce temporal chunk signals that can be clearly spotted, in order to differentiate high- from low-competent. The suggested method of achieving this is to ask participants to copy exactly the same stimuli. In copying, all participants should produce exactly the same response, thus the difference will only be the time (pauses). The similarity of all participants' output is a crucial aspect of this thesis. In addition, the previously listed studies showed that there are lots of measures of programming comprehension that work; however, are there other measures which might be an alternative and perhaps better?

2.3.2 Assessment equipment (mode of production)

Various assessment equipment has been used to measure programming comprehension, such as paper-pen, keystrokes, and smart-pen. This section discusses the equipment utilized by previous researchers.

2.3.2.1 *Paper and pen*

Generally, programming processes consist of major phases, as illustrated in section 2.3.1. This section discusses the paper-inking pen, Livescribe smart-pen, and graphics tablet-inking pen techniques. In addition to using paper-inking pens in educational assessments, several empirical

studies, such as those by Barfield (1986), McKeithen et al. (1981), Miara et al. (1983), Obaidellah (2016), Rambally (1986), Shneiderman (1976, 1977), and Shneiderman et al. (1977), have used it as a tool to measure programming comprehension by asking participants to write their solutions on blank paper using a simple inking pen, while doing different comprehension (i.e., composing, modifying, debugging, fill-in-the blanks, multiple choice, etc.) or recall tasks.

Livescribe smart-pen technology (high-detailed resolution behaviour measurement) is similar to the traditional ink pen in its purpose. However, the Livescribe smart-pen has an advantage in that it allows researchers to record every pen stroke as it digitizes the writing, and thus gives a time-stamped registry of the writing. That time can be used to calculate certain quantitative experiment features, as in the studies by Rawson and Stahovich (2013), Stahovich and Lin (2016), and Rawson et al. (2017). They used the Livescribe smart-pen in their empirical studies as a computer-based assessment technology in order to test students' homework activity and compare it with their course grades. Their findings, based on timestamped numeric pen stroke records, showed that students' course grades correlated positively with the quality of and amount of time spent doing homework.

Finally, the graphics tablet-inking pen technique is different from the Livescribe smart-pen in that it utilizes a graphics tablet (with paper attached to it) and a special inking pen. This method is exploited in various disciplines: to measure second language learners' competence, via pause analysis of copied sentences (Zulkifli, 2013); to measure mathematical equation competence through pause analysis of copied math equations (Cheng, 2014, 2015). The studies by Cheng (2014, 2015) and Zulkifli (2013) are considered the foundation for this thesis in using pause analysis to measure program code transcription, via use of the graphics tablet-inking pen technique to measure programming comprehension.

2.3.2.2 *Keystrokes (typing)*

The idea of recording keyboard presses to measure programming quality or performance is an idea applied by Adelson (1981), Pennington (1987), and Ye and Salvendy (1996). Participants' responses, error rates, and latency response times were recorded via a control program and software package in all these studies. Recently, in a study by Liu et al. (2012), this technique (referred to as keystroke logging), was used with low-level performance data, recording number of keystrokes in seconds, to measure the difference between high- and low-competent programming performance in the software development and cognitive field. They found no relation between the number of keystrokes and programmers' performance. The aim in this thesis is to use equipment that can measure programming comprehension using a more precise time scale (milliseconds). Keystroke logging depends on participants' typing abilities, whereas using pen and paper is one of the learning basics for any educated person. Therefore, keystroke logging was not used in this thesis.

Finally, the previously mentioned methods are time-consuming; one of the disadvantages of using them is the question of what happens if a student barely writes anything in an exam or can barely write a program, but still knows quite a bit about what the commands are (i.e., they might be quite a good programmer). This is an issue that should be examined, and a method that allows anybody, no matter how experienced, to provide a complete response that we can evaluate is worth employing. As a result, the transcription (copying) task was used in this thesis as a means of providing a complete response. This thesis also uses the graphics tablet-inking pen, but it will be different from previous studies measuring programming comprehension in that it will combine a digital pen and an A4 paper sheet with a graphics tablet to measure programming comprehension quickly and efficiently.

2.4 Chunk structures and pause analysis

This subsection will demonstrate the following: the chunking process as a fundamental concept in the cognitive domain; the history of the application of the pause analysis methodology and the possibility of applying it in the programming domain, and the relation between pauses and cognitive processes; the relationship between transcription as a task and working memory; the potential signals and measures of the chunk structures such as keystroke logging and Graphical Protocol Analysis; and finally, proven studies of micro-behavioural temporal metrics reflecting chunk structures.

2.4.1 Chunking as cognitive processing in general (other domains)

'Chunking' is one of the most essential cognitive processes; it is sometimes called 'association'. It is a basic mechanism by which information is linked together. If someone is trying to remember a mobile number, they will try to remember it as separate groups, such as "123 456 7891" When someone is trying to remember a shopping list, they will group the items based on their relationships, such as "pasta and rice" as one chunk and "lettuce and tomato" as another chunk. It is important to clearly define the chunking process, not only because it is a significant indication of programming competence, but also because it is important in understanding how transcription tasks work. In the field of psychology, Cowan (2001, p.145) defines the chunk as a "collection of concepts that have strong associations to one another and much weaker associations to other chunks concurrently in use". This procedure includes aggregating input information into 'chunks', which are simple to handle as individual units. The idea of chunking is applied to perception and memory (Miller, 1956; Gobet et al., 2001; Jones, 2012); when individuals see things, they see them in groups, and when a difficult question is received, it is tackled by splitting it into fractions (Shneiderman, 1976): this is known as chunking. Things are stored in the memory in the form of chunks which are grouped together and then built into a chunk hierarchy in the memory. When

memorizing, individual mental powers are geared to conceptually ‘group’ as much information as possible (i.e., individual chunk structures), which facilitates the process of memorizing. The writing process involves perception, memory, and output (i.e., motor behaviour). During transcription (i.e., writing), which is a higher-level process, individuals try to retrieve chunks from memory, then break them down into their component parts in order to complete the motor behaviour which results in marks on the paper.

During learning, the size of chunks increases, with the merging of sub-chunks (Cowan, 2001). The chunk structures of learners gradually change; hence chunk structures are quite different for low-competence participants and high-competence participants. Many experimental studies have indicated the validity of using chunking as a cognitive process to distinguish high- from low-competent in diverse domains, and have agreed that high-competent individuals use chunks that demonstrate functional units. One of the most frequently investigated domains is chess (Chase & Simon, 1973; Gobet & Simon, 1996), where the players perceive structure in such positions and encode them in chunks. Another common area is electronic circuit symbol drawing, as illustrated in Egan and Schwartz (1979) and Lane, Cheng, and Gobet (2000), where there was obvious "chunking" or grouping into functional units, analogous to Chess masters' recall of chess positions. Another sector is in explaining the age-related differences in performance while performing a non-word repetition test (Jones, 2012), where task behaviour improves with time as a result of chunking, which is consistent with children's performance. Jones found that chunking generates apparent changes in areas such as short-term memory capacity and processing speed, which are frequently highlighted as processes of child development. Another field is measuring mathematical competence via copying equations (Cheng, 2014, 2015; Cheng & Rojas-Anaya, 2007). Chunking has also been used in measuring second language learners’ competence (Zulkifli, 2013). This indicates

the feasibility of using chunking as a cognitive process to measure expertise levels in other domains (Gobet et al., 2001).

2.4.2 Pause analysis discovery

Since the 1970s, writing researchers have varied from studying written documents to employing real-time techniques (Schilperoord, 2001). Studies have begun to focus on implicit cognitive processes that happen during the writing process. Hence, to comprehend the cognitive processes underlying writing in general, many studies have applied different methodologies and utilized different tools in order to capture pauses.

The initial approach used to analyse the processes writers are involved in while writing was the use of cognitive status of pause (Schilperoord, 2001). After this kind of studies, video recording and think aloud methods were suggested and used by Matsuhashi (1981, 1987). Video-graphs allowed more features to be captured, such as hand motion and facial expressions, because participants were asked to think aloud. A decade later, researchers such as Schilperoord (1996) and Spelman Miller (2000) improved techniques for real-time writing capturing. Their new technique indicated pauses or temporal signals, and is used in many studies, such as Cheng (2014), Cheng and Rojas-Anaya (2005, 2006, 2008), Spelman Miller (2000, 2006), and Zulkifli (2013), as well as this thesis.

Until now, programming knowledge studies, as illustrated in section 2.3, have not utilized quick, sufficient, efficient, and accurate methods to measure cognitive processes in programming. Using pause analysis in a copying task, which is considered a novel technique in the computer programming field, permits the capturing of cognitive knowledge such as control (control flow). Ordinary tests assess knowledge only by providing scores, as presented in section 2.3.1.

2.4.2.1 *Relation between pauses and cognitive processes*

In the cognitive science field, pause analysis has been used in many domains, such as measuring mathematical competence through copying equations (Cheng, 2014, 2015; Cheng & Rojas-Anaya, 2007), writing (Cheng & Obaidellah, 2009; Cheng & Rojas-Anaya, 2005, 2006, 2008; van Genuchten & Cheng, 2010) and measuring second language competence (Zulkifli, 2013), in order to study cognitive aspects. As the pause analysis method has proved to be efficient and effective in revealing underlying cognitive processes (i.e., chunk structures) in the previously listed studies, it can also be used in the programming domain. Thus, this study aimed to examine the reliability and efficiency of using pause analysis in measuring programming comprehension using a copying/transcription task. Until now, computer programming comprehension has not been measured using this procedure.

A positive relationship exists between length of pauses and depth of chunk hierarchy. In this thesis, pause length was used to measure cognitive processes; however, the hierarchy structure produced by the pauses will not be demonstrated, because this is beyond the scope of this thesis, as producing a hierarchical knowledge structure for each participant requires further individually repeated experiments.

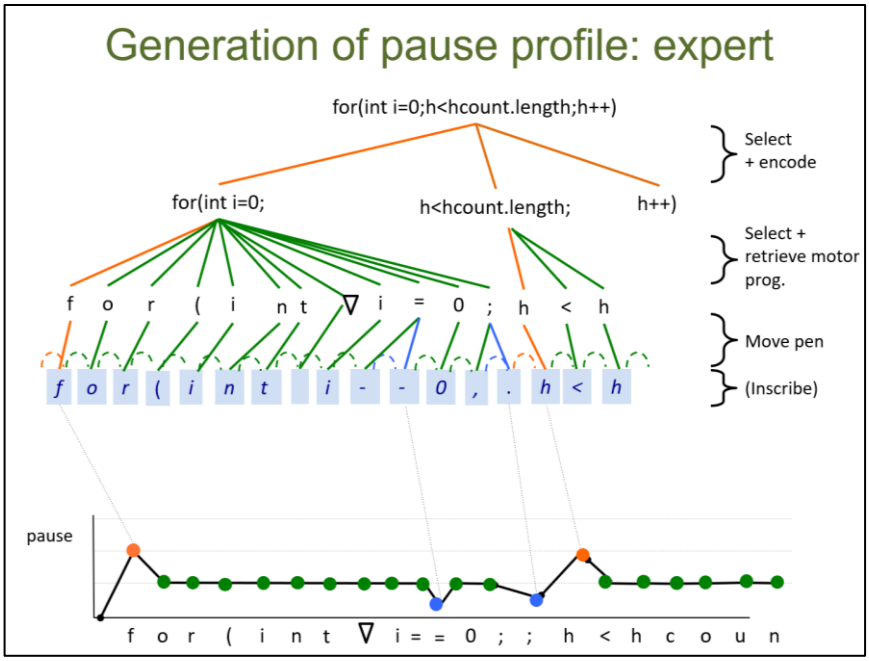

Figure 2.1 A high-competent generation of a pause profile

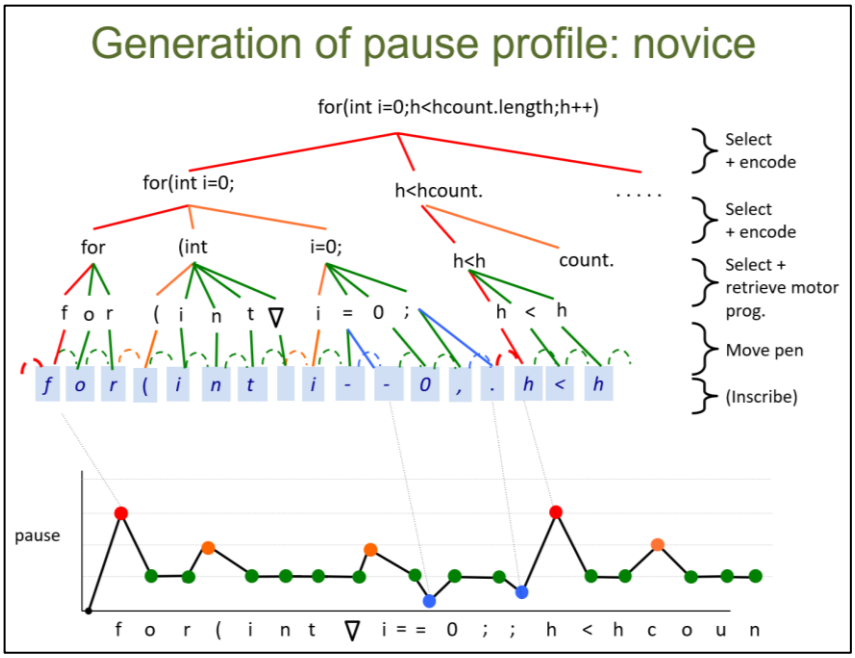

Figure 2.2 A low-competent generation of a pause profile

Figure 2.1 and Figure 2.2 above demonstrate how pauses are generated while a participant is inscribing a line of program code in the experiment. The figures show hypothetical cases which were

adapted from Cheng & Rojas-Anaya, (2008). The same code statement is presented to a high- and a low-competent individual. It is assumed that the high- and low- will break the statement into two different chunk structures. First of all, both of them will select and encode the first chunk they are going to write. Low-competent undertake more selection and encoding than high-competent, and this is because the number of chunks used by low-competent is greater than for high-competent, as high-competent have larger chunks (i.e., a larger number of characters). Then, participants inscribe the chunk letter by letter.

In Figure 2.1 above, a low-competent first chunk is "`for(int i=0;`", which is broken down into three parts: `for`, `(int`, and `i=0`; these three parts are split based on the appearance of the words, not based on their functions (i.e., the exact meaning of the program code statement), and this is because the low-competent understanding of the program is superficial. Furthermore, while copying, spaces are replaced by an inverted triangle, and this is because spaces are important in chunking behaviour; for high-competence participants, copying with or without spacing makes no difference in their chunk structures, but for low-competence participants, having a space is a major indication of having a new chunk, since spaces are typically employed to divide program code into distinct units. Hence, nothing will be recorded if the participant leaves it blank, and as this study requires recording the time, I used the inverted triangle.

The line graphs following each hierarchy diagram in both Figure 2.1 and Figure 2.2 above show that a high-competent takes three steps before he/she can start writing, whereas a high-competent takes two steps, which means a low-competent will have longer pauses than a high-competent. The number of peaks is different between the two graphs, which is a sign of different levels of comprehension. The number of pauses is greater and they are longer for low- than for high-competent, which therefore produces different temporal patterns in task performance and reflects

varying chunk structures. In concrete terms, the greater a person's expertise, the lower the number of pauses produced.

The aim of this thesis is to test whether such a temporal chunk signal can be used to measure programming knowledge (i.e., individuals' chunk structures) and to develop a tool for this. I wished to analyse differences in temporal patterns and develop a comprehension measure. Each participant produced hundreds of pauses, from which a measure of programming comprehension was derived. I evaluated possible measures such as mean, median, and Q3.

2.4.3 Transcription and working memory

Gonzalez et al. (2011) compared copying and tracing, and concluded that copying required a greater amount of memory. Thus, in terms of this thesis, as copying requires resources from working memory, the number of resources needed is based on a person's programming competence. Moreover, working memory space is restricted (Spelman Miller & Sullivan, 2006; Spelman Miller, 2006), and the space will be available based on how a person processes information. For instance, if a person is competent in writing programs, he/she would not require working memory space to remember, for example, a command used to print out a specific sentence or number on the screen. Hence, this person could produce program statements automatically. On the other hand, a person who is unfamiliar with programming composition would require additional memory space, which delays the copying process, and in order to be capable of copying the next statement, there should be free working memory space (Zulkifli, 2013).

However, so far, there have been no studies that use copying a program code as a task specifically for measuring programming comprehension. All previous research clearly indicates that chunk structures are regarded as a fundamental way for individuals to store information about

programming. Thus, having access to individuals' chunk structures may allow measurement of individuals' Java programming competence.

2.4.4 Signals and measures of chunk structures

Writing research saw an evolution in technology with the emergence of the keystroke logging technique as a new framework for logging writing, which permits researchers to explore the cognitive processes behind writing (Schilperoord, 2001; Spelman Miller, 2000). Smart-pens and a combination of smart-pens and Graphical Protocol Analysis (GPA) have also been used to investigate the cognitive processes of writing via pause analysis (Cheng & Rojas-Anaya, 2008; van Genuchten & Cheng, 2010; Zulkifli, 2013). The two subsections that follow introduce the keystroke logging and GPA techniques that have been used in the past, and consider their suitability for trying to assess chunk structures in the programming domain.

2.4.4.1 *Keystroke logging*

Keystroke logging has emerged as a robust computer-based technique for recording writing activity. Writers type on the keyboard while the computer records the typing (Spelman Miller & Sullivan, 2006). It electronically records procedures such as number of keypresses and cursor motion as the writer uses a word processor. This allows the user to spot pause duration and location, and series of writing behaviours. A logfile is produced which contains a highly detailed temporal feature that provides rich data. Many programs based on keystroke logging have been used, such as JEdit, used with the Macintosh operating system, and Inputlog (Leijten & van Waes, 2005). The utility of keystroke logging is well presented by Spelman Miller (2000).

Previously, text production or composition indicted high-level writing processes (plan, translate, revise), and writing was recognized as a problem-solving activity (Zulkifli, 2013). Because of different keyboard expertise, researchers found it hard to distinguish between planning, translation, and

revision. Recently, low-level writing processes such as copying have been used by Cheng (2014) and Zulkifli (2013), and are used in this thesis.

Keystroke logging has proved its usefulness in different disciplines, such as measuring programming performance (Liu et al., 2012). Despite the fact that the keystroke logging methodology is very close to GPA, the current study does not use it for several reasons (mentioned at the end of the next subsection).

2.4.4.2 *Graphical Protocol Analysis: Why I decided to use GPA*

Graphical Protocol Analysis (GPA) is a technique that concentrates on pause duration (i.e., temporal chunk signals). GPA uses a graphics tablet, and utilizes pause measures to examine whether chunks can be detected using writing and drawing pause analysis. Pauses can be recorded at distinct levels throughout the writing process: within a letter, referred to as level zero (L0), among letters (L1), among words (L2), among phrases (L3), and among sentences (L4) (Zulkifli, 2013). Cheng and Rojas-Anaya (2008) and van Genuchten and Cheng (2010) confirm that in working memory, the hierarchical structure of chunks is revealed via graphical production of pauses. Thus, generating a temporal chunk signal can supply information about chunk structures. However, although applying pause levels through writing can be done for drawing and sentences, it is not appropriate for program code, because sentences have clearly separated chunks (i.e., meaningful words), but programs do not.

GPA is superior to keystroke logging in regards to the aim of this thesis to measure programming comprehension for several reasons, as follows:

Firstly, GPA has been successfully utilized in freehand writing copying tasks to measure maths competence, as in Cheng's (2014) study, and to measure second language learners' competence, as

in Zulkifli's (2013) work. Cheng and Zulkifli's research is considered the basis for using GPA as a technique for measuring programming comprehension for this thesis.

Secondly, there are big strategic/operational differences in the way individuals type (i.e., some people use the hunt and pick technique, while others use touch typing, whereas others do half and half). So, there is a significant difference between individuals that has nothing to do with chunking. Previous pilot experiments have shown that these individual typing differences tend to make the temporal chunk signals unclear. GPA utilizes freehand writing, which is considered essential expertise for everyone who is educated (Zulkifli, 2013), and the targeted population for this thesis are educated adults who have already learned how to write.

Thirdly, there is a lot more variability in the measures (i.e., temporal chunk signals) in typing because there are other factors that influence the timing between successive actions, such as the variable distance between the keys. Sometimes two keys are next to each other and can be pressed with the same hand; other times, the keys are far apart and individuals may use separate hands to press each key. As a result, inter-keypress pauses (the pauses between key presses) do not appear to be reliable. Thus, freehand writing provides richer data than keyboard typing (Cheng & Rojas-Anaya, 2005; Zulkifli, 2013). Furthermore, when people write by hand, they only move the pen a small distance each time, whereas when they type, the distances between keys are quite variable, which is why a specially designed response grid is used in this thesis. The temporal chunk signals generated during natural handwriting may be considered more distinguishable and reliable.

2.4.5 Studies of the micro-behavioural temporal measure as reflection of chunk structures

The advantage of using the pause analysis technique for understanding cognitive processing and assessing competence has been shown in various studies and disciplines, such as: text production studies (Schilperoord, 1996, 2001; Spelman Miller, 2000, 2006; Wengelin, 2006), freehand drawing

(Cheng, McFadzean & Copeland, 2001; Cheng & Obaidellah, 2009; Roller & Cheng, 2014), writing words and sentences (Cheng & Rojas-Anaya, 2006; Cheng & Rojas-Anaya, 2008; van Genuchten et al., 2009; van Genuchten & Cheng, 2010), and mathematical equation copying (Cheng, 2014, 2015; Cheng & Rojas-Anaya, 2005, 2007). These studies highlight the value of using this approach to assess competence and comprehend cognitive processes in general. This inspired the current project to investigate the feasibility of using pause analysis as a technique to measure programming comprehension.

The cognitive processes underlying drawing have been examined by previous studies such as Cheng, McFadzean, and Copeland (2001), and Cheng and Obaidellah (2009), using temporal chunk signals (TCS). Roller and Cheng (2014) used Graphical Protocol Analysis (GPA, illustrated in section 2.4.4.2), graphics tablets, and freehand writing to reflect cognitive processes while drawing complex diagrams. Previous drawing studies suggest that utilizing the GPA methodology to measure the cognitive processes, which could be extracted via TCS measures, underlying programming (i.e., programming comprehension).

In terms of writing words and sentences, van Genuchten et al. (2009) used TCS to examine whether dyslexia affects children's writing of sentences from memory. They recruited 109 children between seven and ten years old. After familiarization, participants recalled 24 sentences which varied between familiar, unfamiliar, and non-words. Graphics tablets, traditional pens, and special software were used to record pen movement and to extract the position of the pen, as also used by Zulkifli (2013), who aimed to measure second language learners' competence via the copying method and not writing from memory as van Genuchten et al. (2009) did. Van Genuchten et al. (2009) also compared working memory, short-term memory, and automated working memory assessment, and the means and medians of the pauses of various kinds of sentences were calculated. The results showed that working memory problems in dyslexic children have little effect

on their writing. It was also found that the mean duration of the pauses presented differences between age groups in working memory performance.

The other study, van Genuchten and Cheng (2010), examined whether writing sentences from memory can distinguish between five different levels of chunking. Thirty-two adult English native speakers were recruited and a graphics tablet was utilized to record pauses between five levels: sentence, phrase, word, letter, and stroke. The stimuli consisted of eight English sentences which were specifically written to include the five listed levels. Each stimulus contained three to four sentences, two to three phrases, four to eight words, and letters that could have one or more stroke. The stimuli were presented in jumbled form, and written on paper and attached to the tablet; participants recalled the sentences on paper taped to the tablet containing horizontal rows of rectangles; each letter should be written in each rectangle. Participants were asked to write '#' as in van Genuchten et al. (2009) at the beginning of each sentence, to make sure that writing was well underway before they started writing the first letter. Five measures were calculated for each stimulus. The median was calculated for each level. It was found that TCS between the written elements reflected the five levels in writing sentences. There are several implications for the programming domain. This finding implies that the five levels can be utilized in sentence writing and drawing (P. Cheng & Obaidellah, 2009), whereas they are not suitable in programming, as chunk structures are not obviously divided like words in sentences.

In the process of writing from memory, as seen in van Genuchten et al. (2009) and van Genuchten and Cheng (2010), it is quite clear that chunking behaviour is apparent in temporal signals. Because program code transcription, the task used in this thesis, is also a writing process, there is reason to believe that programming chunks could manifest as temporal signals.

Using a maths competence assessment, Cheng and Rojas-Anaya (2007) successfully demonstrated that copying and pause analysis is a valid method of distinguishing between and measuring different levels of mathematical formula writing. They recruited four participants with vastly different levels of expertise, in order to maximize differences in participants' levels of chunking. The stimulus consisted of sixteen formulas of varying complexity, which participants looked at for two minutes before starting copying. A standard graphics tablet was used to record strokes, in order to analyse pause duration between written strokes. The authors conducted a test comparing the mean, long pause duration (LPD), and long pause count (LPC) as three potential measures of chunking (to find which would be an effective measure of chunking). LPD is the duration of the pause, thus it shows the actual magnitude of the pause, whereas LPC is the number of pauses which are greater than a specific threshold. Cheng and Rojas-Anaya (2007) found that LPD was better at differentiating levels of competence. Furthermore, they concluded that the most expert participants had a smaller number of chunks (i.e., chunk size is larger), whereas the low-competent had a larger number of chunks. Thus, as chunking in this study is manifested as pauses, this suggests the possibility of applying the copying and pause analysis method to the measurement of competence in the programming domain. It also implies that because they employed a limited number of equations, there is no requirement for large programs to be used to assess programming comprehension.

Cheng (2014) asserts the feasibility of using GPA with freehand copying in measuring mathematical competence using the third quartile measure to see whether it is an improvement when compared with the LPD measure used by Cheng and Rojas-Anaya (2007). Twenty adults were recruited, with diverse levels of maths expertise in order to maximise disparities in chunking levels. Their level of expertise was measured using a questionnaire with three parts, focusing on maths qualifications, problem solving with multiple choice answers, and a confidence rating of those answers. The stimuli consisted of eleven copying items, three for practicing and eight for copying. A standard graphics

tablet, A4 paper, the tablet's inking pen, and specially designed software (SMouseLog) were used as the experiment tools. Each stimulus was presented on a 5 cm card above the graphics tablet. Pause measures normalized the pause third quartile (Q3) and pause interquartile range (IQR) for each individual test per participant. Considering all pauses, strong correlations were found using the Q3. For the number stimuli and equation stimuli these were approximately -0.55 and -0.70, respectively. Further, the author found both pause Q3 and pause IQR were better measures than LPD. There are several implications for programming competence assessment. Not just Cheng (2014) but also Cheng and Rojas-Anaya (2007) and van Genuchten and Cheng (2010) suggest the possibility of using GPA with freehand copying in measuring high frequency data, which implies the possibility of using it in the current research domain (i.e., programming). As maths equations are more similar to program code than copying words and sentences, and drawing geometric diagrams, this suggests that the method may indeed work for program code. The general configuration of stimuli, materials, and procedure is a good basis for the design of a programming comprehension experiment. For example, combining a relatively small number of short stimuli is sufficient for a fairly accurate measure. Pauses again appear to be a sufficient general measure of competence. Although normalization did not improve the Q3 measure in general, the results with the higher competence participants suggest that further investigation is worthwhile.

Cheng (2015) demonstrated the feasibility of using the 'centre-click' copying method for measuring mathematical competence by utilizing paused-based temporal chunk signals in a copying task, and investigating whether utilizing other forms of interaction such as a mouse can be used instead of freehand writing and drawing. Twenty-two adults from various maths levels were recruited. Participants completed the same questionnaire utilized in Cheng (2014). In this experiment, a typical mouse, response grid, and the SMouseLog software were used. Symbols in the grid were grouped according to mathematical meanings, which helped high- more than low-competent. The stimuli

involved twelve items, eight formula items, and four practice items. Formulas were varied in their level of difficulty and scrambled when presented to the participants. However, practice items, which included all grid symbols, were presented in the correct order. Participants were trained to click a return to centre (RTC) symbol located in the middle of the response grid each time they clicked on any symbol, in attempt to make the mouse distance uniform when moved to reach each symbol, as Cheng found no correlation between the pauses and maths competence when the distance was not controlled. Items were separately presented, written on a card placed at the same height as the grid. The author examined first quartile (Q1) and Q3 of the following: symbol, RTC, ALL, which added a symbol to RTC, and subtracting a symbol from RTC. He showed that Q3 is sufficient as it showed different levels of competence, whereas Q1 showed that all participants were at the same level of competence. For all the participants, strong correlation was found using the Q3 symbol, which is -0.52, followed by Q3 ALL = -0.50 and Q3 RTC = -0.10, which is the worst. Adding and subtracting RTC and symbol were in the middle. Basic mathematical skills were normalized, calculated by subtracting basic Q3 from Q3 ALL. Stimuli varied in difficulty level, suggesting that this could be implemented in this thesis. Q3 again appears a good general measure of competence. The centre-click method may not appropriate for the programming domain as a program consists of a variety of characters, numbers, letters, symbols, and punctuation, which are hard to aggregate in a reasonable-size grid. Thus, it was not used in this study.

The studies by van Genuchten et al. (2009), Cheng and Rojas-Anaya (2007), Cheng (2014), and Zulkifli (2013) all suggest the value of the GPA method for measuring high cognitive skills, and thus the value of measuring programming competence via graphics tablets, traditional pens, an especially designed response sheet, and the special software which records pen movement. Van Genuchten et al. (2009) utilized freehand writing in a memory task. The specifically designed response sheet (to prevent cursive writing) could be utilized, as program code contains a variety of characters and it is

assumed that it will make pauses between characters apparent. Starting with '#' appeared to be a good idea to implement in this thesis, because the use of '#' at the beginning is to make sure the first pause associated with a character of the stimulus has a meaningful pause associated with it. On the other hand, when beginning transcription of the stimulus with the very first letter of the stimulus it would not be possible to calculate the first pause duration.

In conclusion, the work of Cheng and Rojas-Anaya (2007), Cheng (2014), and Zulkifli (2013) measuring maths and second language learners' competence revealed evidence that measures depending on the pauses (i.e., temporal chunk signals (TCS) of micro-behaviours) that occur in freehand copying reflect learners' chunks structures. Therefore, there is a theoretical base to utilize these TCS of micro-behaviours for measuring competence in the programming domain. Put another way, TCS can be used to evaluate high-level cognitive skills involved in programming. In addition, van Genuchten et al. (2009) and van Genuchten and Cheng (2010) have shown that writing from memory allows use of pauses as a measure. Whereas, in transcription, participants keep looking back at the stimulus, and when they look back at the stimulus, they pick up a number of chunks to copy. So, by using transcription other measures of competence can be produced, more than just pauses. These include the number of times participants view the stimulus, the number of characters per view, or the writing time, which are the hide and show (HS) measures. This thesis's experiments are the first to utilize HS measures.

A limitation of the previous work, which was conducted in labs (Cheng, McFadzean, and Copeland, 2001; Cheng and Obaidellah, 2009; van Genuchten et al., 2009; van Genuchten and Cheng, 2010; Cheng & Rojas-Anaya, 2006; Cheng & Rojas-Anaya, 2008; van Genuchten et al., 2009; van Genuchten & Cheng, 2010; Cheng, 2014, 2015; Cheng & Rojas-Anaya, 2005, 2007; Zulkifli, 2013), is that only pause measures were implemented, and these only look at the output motor writing behaviour, not

perception. As chunking is used for perception as well as for writing output, it might be useful to look at chunking during perception as well.

All the previously discussed work shows that chunk signals occur not only when writing linear notations as in maths and natural language learning, but also when drawing. Hence, it is considered a general phenomenon, and there seems to be no reason why it would not be expected to work with copying (i.e., transcribing) program code.

2.5 Design space

A primary goal of this thesis is to maximise the difference between high- and low-competence participants' chunk signals. As a result, several concerns were raised about the setup of the thesis experiments. These concerns can be summarised as follows:

1. What type of task would be the most suitable method of measuring Java programming comprehension? Comprehension (i.e., composition, modification, debugging, mutable-choice, fill-in-the-blanks, subjective ratings, etc.), recall (i.e., memorization), combined comprehension and memorization, or a copying (transcription) task?
2. What is the best production mode to employ for the specified task, freehand writing or keyboard typing?
3. What kinds of stimulus modifications are possible, complete program code, parts of program code, or code (reserved) distinct words?
4. Should the stimuli be presented all at once, line by line, or word by word?
5. What modes of stimulus transcription or presentation are possible, hide-show, view-display, or both?

6. What is the best way of familiarising participants with the experimental stimuli? Pre-exposure is used before beginning the real activity. Alternatively, the stimulus might be revealed immediately at the start of the task.
7. Might using different supplementary aids in programming maximise the difference between high- and low-competence participants' chunk signals? These might include the use of flowcharts, different levels of comments, memorability, modularity, syntax highlighting and colouring, the use of indentation (i.e., layout), and arbitrarily displayed stimulus content orders.

Table 2.4 below shows different factors that were considered when designing the experiment assessment method and stimuli. The role of this table is to represent the variables investigated in this thesis's experimental design.

No.	Factor	Thesis design factors
1	Task type	Transcription & Subjective rating
2	Production mode	Free-writing
3	Stimulus manipulations	Produce errors
		Code fragments
		Whole program
		Line by line
		Reserved words
4	Stimuli presentation	Concurrent
		Hide/show
5	Stimuli familiarization	Immediate exposure
6	Stimuli augmentation	Syntax highlighting
		Indentation/layout
		Modularity
		Memorable variable names
7	Participants' competence levels	Same level (naïve, novice, intermediate or expert)
		Naïve, novice, intermediate & expert
		Naïve, novice & intermediate
		Novice and expert

Table 2.4 Thesis design space aspects

2.5.1 Aspects that may influence programmers' cognitive processes

There are many aspects that may affect reading code, thus influencing comprehension of it. Learners who are weak in reading program statements are commonly weak in solving code problems (Lister et al., 2004). Thus, it could be concluded that participants who cannot read code clearly will not be able to compose code correctly (Obaidellah, 2016). This section includes programming aspects (i.e., supplementary aids) that may affect programmers' understanding of a program, such as: different stimulus (i.e., program code) presentation mode, syntax highlighting, indentation, memorability, flowcharts, and code scrambling. In the context of the present study, the use of these

supplementary aids may differentiate high- from low-competent; only those that will separate the two groups will be useful.

In the following subsection, the overall design of the new assessment method that is evaluated in this thesis is illustrated in terms of the following aspects, which may affect programmers' cognitive processes: (1) different modes of transcription (i.e., stimulus presentation modes), (2) syntax highlighting, (3) indentation, (4) memorable variable names, commenting, and modularity, (5) flowcharts, and (6) code scrambling.

2.5.1.1 *Stimuli presentation modes*

Two main stimuli presentation modes (i.e., different modes of transcription) are used in the design of this research's experiments: hide and show (HS) and view display (VD). VD, where the stimulus is always apparent, is the appropriate mode to provide the $pause_{Q3}$ measure. HS, where the stimulus becomes visible only when a participant clicks a special button, is the suitable presentation mode to provide the other temporal chunk signals, characters per view (i.e., view-numbers), writing-times, and view-times.

2.5.1.2 *Syntax highlighting*

Syntax highlighting (or syntax colouring) means making parts of the program code syntax obvious by colouring them with a different colour to the rest of the code statements. Sarkar (2015) investigated the impact of syntax colouring in a comprehension task and whether its effects can differentiate programmer expertise. Sarkar found that using syntax colouring improved programming comprehension. He found that it enhanced program comprehension speed in low- more than in high-competent. Dimitri (2015) states that if programming expertise increases, the time needed to understand the highlighted task decreases. He found that using syntax highlighting increased debugging and writing speed. Thus, the intellectual overhead required to comprehend

highlighted program code is lower than the mental overhead required to understand plain program code (Sarkar, 2015). Rambally (1986), Dimitri (2015), and Sarkar (2015) conclude that syntax highlighting and function (control flow) colouring significantly improve programming understanding. Their findings reinforce the theory that when trying to understand a program code which includes extra visible cues, the required overhead will be less. Sarkar's (2015) study led me to refrain from using syntax highlighting in my experiments as it would have lessened the difference between high- and low-competent. What I am precisely focusing on is finding techniques that will differentiate students with good and poor programming comprehension in relation to chunk signals.

2.5.1.3 *Indentation*

Indentation means leaving a specific space before starting to write a program statement; it is used in order to better structure code blocks. Indentations are not compulsory in Java and most programming languages. Figure 2.3 below presents an example of a block of Java program code including indentation.

```
for(int h=0; h<hCount.length; h++)
  {
     System.out.println (h+hCount[h]);
  }
```

Figure 2.3 Indentation example

With regards to indentation use, McKeithen et al. (1981) kept indentation in both program versions, normal and scrambled, as they are vital to the meaning of the ALGOL W programming language. Barfield (1986) also kept them in BASIC programs, which were presented to participants as executable orders, random lines, and random chunks, but he did not consider indentation in counting correct lines. However, Pennington (1987), who used the Cobol and Fortran programming languages, removed the indentation. Shneiderman (1976), who used Fortran, also eliminated

indentation from experimental programs. As Shneiderman (1977) and Miara et al. (1983) showed, although participants prefer the indentation, there is no considerable effect on their performance measures in general. For this thesis, as indentation is determined by the logic of the program code, it could be used, as I am trying to manipulate techniques that could discriminate high- from low-competent in terms of chunk signals. Thus, it is assumed that indentation will strengthen the chunk signal for high- more than low-competent, because it will make the program functions clearer, hence will help experts to understand the code as they already have good programming knowledge. On the other hand, it will not have a considerable effect for the low-competent individuals, because their programming knowledge is superficial. Thus, the use of indentation differentiates programming comprehension levels.

2.5.1.4 *Memorability*

In a series of studies, McKay and Shneiderman (1976), Shneiderman (1976, 1977), Shneiderman et al. (1977), Shneiderman and Mayer (1979), and Rambally (1986), found that memorable variable names, commenting, and modularity had a noteworthy influence on program comprehension, whereas using flowchart diagrams did not.

Commenting is used to simplify and explain the role of the program code statements, and can be used for one or more statements. Comments are not executed and are not considered a part of the actual program. Shneiderman (1977) differentiated two kinds of comments, low-level comments and high-level comments. He found that high-level comments assist in evolving the hierarchical structure, whereas low-level comments prohibit it, because they are like repeating code lines whose function is already obvious to experts. In addition, they are distracting to readers. On the other hand, Miara et al. (1983) and Rambally (1986) did not use comments at all in their experiments. In this thesis, comments were not used because it is difficult to systematically write comments.

Regarding modularity, Shneiderman and Mayer (1979) presented participants with three program formats: modular, where each module has a clear function, nonmodular, which is an undivided serial program, and random modular, which is a program divided into subdivisions without obvious function. Shneiderman and Mayer (1979) and Rambally (1986) found that an unclear breakdown of a program could make it more difficult to understand. However, while providing a random modular format to the superior students in the class, the latter accomplished high grades, despite the difficulty of the program. A further result was that "modular program design facilitates the chunking process". Hence, measuring program comprehension using a modular program is difficult for low-competent but helpful for high-competent. With regards to this project, modularity may be manipulated because it will help high-competent understanding, and thus make the difference between low- and high-competent obvious by strengthening the chunk signal, which is an indication of a higher level of programming expertise.

2.5.1.5 *Flowcharts*

Shneiderman et al. (1977) and Shneiderman and Mayer (1979) found no difference when using a flowchart to measure program comprehension. Shneiderman et al. (1977) indicate that the use of a flowchart is more beneficial when used with comprehension rather than composition tasks. However, they also concluded that the use of the flowchart neither benefits nor harms students. Obaidellah (2016), while trying to measure students' programming comprehension by asking them to translate a Java program code to a flowchart and a flowchart to a program code, concluded that participants did better in interpreting the code into the flowchart than in the reverse task. So, Obaidellah's results agree with those of Shneiderman et al. (1977). Overall, the aid provided by a flowchart is not dependent on the existence of the flowchart only, but also and mainly on the kind of information utilized to help learners in building their programming knowledge.

Therefore, based on previous studies, it can be concluded that the use of a flowchart does not have a significant advantage in programming comprehension, hence a flowchart would not be useful in this particular thesis, as I am looking for a technique that discriminates between students with good and poor programming comprehension via chunk signals. On the whole, utilizing flowcharts will not differentiate high- from low-competent individuals in terms of temporal chunk signals.

2.5.1.6 *Code scrambling*

Scrambling an executable program code means changing its statements' order. They are not presented to the participant in the normal executable order, thus there are no logical, functional, or control flow relations between the statements. Code scrambling obviously makes it harder to read and understand the overall program function because it is not presented as usual. Barfield (1986), Shneiderman (1976), and McKeithen et al. (1981) found that when participants were presented with a scrambled version, both high- and low-competent performed equally. This is because experts were used to grouping various parts of a program according to their meaning, so they found it difficult to recall the code as its lines were not grouped in meaningful chunks. However, when they were presented in the executable version, participants' ability to remember increased with an increase in expertise (Barfield, 1986; Shneiderman, 1976).

Summing up, previous studies indicate that low- and high-competent act alike when using scrambled versions. In addition, Shneiderman (1976) and Zulkifli (2013) (who used the pause analysis while copying technique to measure second language learners' competence, the same as this thesis's technique) found no benefit from using random/scrambled versions. This thesis's main aim is to find a technique that clearly differentiates High- from low-competent. All this implies that using scrambled program code in this thesis is useless.

To conclude, in order to measure programming comprehension rapidly and efficiently, it was decided to use temporal chunk signal analysis in a freehand writing transcription task utilizing the graphics tablet-inking pen technique. I decided to use natural handwriting, which is supposed to be a more accurate and precise measure than keyboard typing, because people have different typing abilities. Its feasibility and benefits have been demonstrated in previous studies, and as a result, I decided to use it in this thesis.

2.6 Summary

Three major domains have been covered in this section, studies of: programming knowledge; programming tasks; chunk structure and pause analysis. Based on a review of the literature in these domains:

- There is a need for an effective programming assessment approach.
- It is clear that programming involves chunking.
- Pause analysis in relation to chunk structure in a copying task has been successfully applied to measure competence in other domains, therefore it might potentially be used to measure programming comprehension.

The overall target of this thesis was to develop a novel approach for evaluating computer programming comprehension by analysing the temporal chunk signals that arise in the task of code transcription. To accomplish this target:

- This research focused on freehand writing by evaluating adults' programming skills.
- This research methodology may suggest an alternate approach to measuring learners' programming skill over typical methods (i.e., composing program code, debugging, or modification).

3 Experiment 1: Measuring Java comprehension

Chapter content:

- An introduction that includes the experiment hypothesis and key questions.
- The experiment methodology:
    - General experiment design.
    - Classification of participants.
    - Experiment implemented materials.
    - The experiment procedures.
    - Stimuli design.
- The experiment results:
    - Independent measure of competence.
    - Evidence of the role of chunking in Java transcription task.
    - Regression analysis for the behavioural measures.
    - Further evidence of the role of chunking (Content analysis).
- Discussion:
    - How the experiment builds on previous findings.
    - The relation between the behavioural measures.
    - Content analysis.
    - Suggestions and future work.
- Summary.

3.1 Introduction

The goal of this experiment is to see if the previously developed chunk-based methods for measuring competence via copying in domains such as maths and second language learning can be applied to programming. According to the literature review, this technique has never been applied to programming. As stated in Chapter 2, chunking is important in the doing and learning of programming (e.g., Shneiderman, 1976; McKeithen et al., 1981; Pennington, 1987). In this experiment, the aim is to measure participants' existing programming knowledge. It is assumed

that: (1) Java programming knowledge is hierarchically represented in the learners' memory (as demonstrated in section 2.2.2 in Chapter 2); (2) High-competent will read the code for meaning, thus they copy it knowing the meaning of it, whereas low-competent may copy it without understanding the meaning (as explained in section 2.2.3).

In previous transcription experiments (e.g., Cheng, 2014, 2015; Zulkifli, 2013), the stimulus has typically been shown on a card or computer screen near the writing tablet, allowing participants to alternate their gaze between the stimulus and the tablet. What is new for this experiment is the use of the hide and show (HS) presentation mode to record when the participant moves between the stimulus and the tablet. When the participant presses a special button, the stimulus appears on the computer screen. To write, the participant must release the button, and the stimulus is masked. This extends the repertoire of techniques that may be used to assess chunk structures with a method that targets the processing of several chunks, at a 10 s timescale, which makes available the behavioural measures: (a) *view-numbers*, (b) *writing-times*, and (c) *view-times*.

This method (HS) contrasts with a previous method that analyses elements within a single chunk to collect Q3 pauses – I call it view display (VD) mode, where stimuli are shown all the time. Regrettably, I was unable to record pauses for this experiment due to a technical issue with the experimental device. As a result, I will not refer to Q3 pauses or the VD presentation mode throughout this chapter.

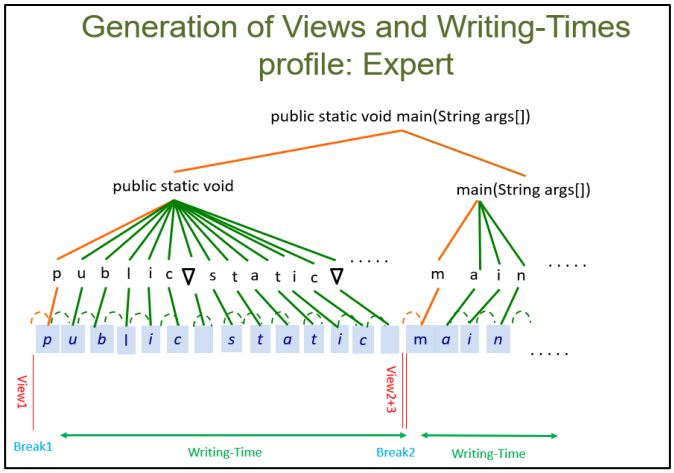

Figure 3.1 Generation of view-numbers and writing-times profile: High-competent

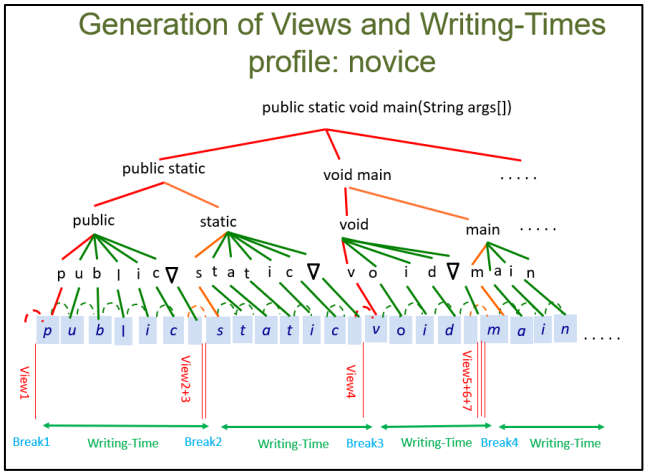

Figure 3.2 Generation of view-numbers and writing-times profile: Low-competent

Figure 3.1 and Figure 3.2 show the behavioural measures that are produced during the transcription of a line of Java code: view-numbers (the total number of views of the stimulus in a trial, obtained by counting the number of button presses to view the stimulus), writing-times (the time spent writing between two consecutive views), and view-times (the length of each look at the stimulus, obtained by subtracting the time when they release the button from the time when they press the button).

As mentioned in the first chapter, I used the Java programming language because it is commonly used. Java is particularly important to this thesis because it is the primary programming language taught to undergraduate students at the University of Sussex (i.e., my participants), and I had full access to these students.

The same Java code statement is shown in both the above figures for a high- and low-competent. Based on their chunk structures, the high- and low-competent have different writing-times and view-numbers. The high-competent has a longer writing-time than low-competent because the high-competent chunk size (i.e., number of characters) is larger and has fewer chunks than the low-

competent. As a result, the high-competent has fewer views of the stimuli than the low-competent (view-numbers). It is believed that the view-times are the same for both the high- and low-competent (the reasons will be explained later). *Views* in Figure 3.1 and Figure 3.2 refers to any time a person presses the button to display the stimuli (i.e., there could be several views without writing), whereas *break* refers to the time between stopping writing and starting writing again, which could include one or more views. The majority of the participants have the same number of views and breaks. However, because some of the participants have more than one point of view, I consider all of those points to be one break, as revealed by my additional analysis (i.e., content analysis), and which was not evident from the previous analysis. I chose to address view-numbers (i.e., views) rather than breaks in this chapter because I examined both and found no difference. Further, in order to conduct the experiment, multiple decisions must be taken and evaluated in order to determine the effectiveness of the assessment of programming comprehension and the distinction between high- and low-competence participants.

Therefore, the experiment aims to explore whether:

(1) Could chunk-based measures of comprehension be extended to domains other than mathematics and natural language learning?

(2) Is it possible that handwriting a program code provides powerful and stable temporal chunk signals that can be used to assess programming comprehension?

(3) Can programming comprehension be accurately measured using view-numbers, writing-times, and view-times?

Various predictions can be made for the HS measures.

Assuming that the size of a stimulus remains constant, I predict:

H1) View-numbers: more competent participants will have a smaller number of views of the stimulus in a trial because their chunk size is bigger.

As more competent participants' chunks contain more content, I predict:

H2) Writing-times: for more competent participants, the duration of written answers after each stimulus view will be longer because their chunks are bigger and each character takes the same time to write.

This suggests that writing speed is independent of target domain expertise, which is plausible for adult participants. Now, as the time to perceive a chunk is approximately constant (Chase & Simon, 1973), and if working memory capacity is independent of competence, then I predict:

H3) View-times: the amount of time spent on each individual view of the stimulus will not be directly related to competence.

I used two levels of stimuli difficulty (explained in detail later): simple stimuli, which contain frequently used components of Java, introduced earlier during instruction (in the students' programming module); and difficult stimuli that include more specialist expressions that are presented later in the course. So, I predict:

H4) Performance on simple stimuli will be superior to performance on advanced stimuli, with fewer view-numbers and longer writing-times, but there will be no effect on view-time.

Note that H3 is framed negatively, so care is required to interpret data that might support it. In particular, the magnitude of other effects must be strong enough to indicate that the possibility of the absence of an overall view-time effect is not due to a lack of statistical power. The underlying pattern of view-time data can also be examined for supporting evidence.

Clearly, the predictions are based on strong assumptions, so no reliable measurements of competence can be achieved until the effects of chunking produce significant temporal signals.

What was done in this experiment that was not applied in previous studies (e.g., Cheng, 2014, 2015; Cheng & Rojas-Anaya, 2007; van Genuchten et al., 2009; Zulkifli, 2013) is content analysis.

3.2 Method

There are numerous measures of programming competence, as stated in the literature review; for this experiment, I employed the following measures: view-numbers, writing-times, and view-times, collected in HS presentation mode, where the stimuli are only visible if a participant continues to press a special button.

The following sections describe the methodology used in this experiment (general experiment design, classification of participants, materials, procedures, stimuli design).

3.2.1 General experiment design

The experiment design includes: (1) a within-participant factor, with each person transcribing a simple and a difficult section of Java program code; (2) a mixed factor, involving low- and high-competence participants as well as simple and difficult stimuli (explained below). The order of these trials was counter-balanced. The trials were preceded by two practice stimuli.

(Originally, the experiment was a counter-balanced 2x2 design with a view display (VD, i.e., where the stimulus is constantly shown on the screen) factor to provide pause distribution measures for comparison. Unfortunately, an obscure software-hardware interaction on the experimental computer was found during the analysis. As the original counter-balancing does appear to have affected the reported conditions, for clarity, the experiment is presented as just a single factor.)

3.2.2 Classification of participants

The participants were 24 adults from the School of Engineering and Informatics at the University of Sussex. Recruitment spanned first-year undergraduate students through to members of faculty, to obtain a good range of programming expertise. Ages ranged from 19 to 59 years (mean=25, SD=8.51); there were fifteen male participants and nine female participants. They received £8 for participating.

3.2.3 Materials

There were two main stimuli categories: the first is the practice stimuli and the second is the Java stimuli. The practice stimuli consisted of series of simple statements, such as 'Computer Science', 'Programming Course', 'JAVA Programming Language'. In the Java stimuli, each stimulus was made up of nine lines of code separated into three distinct blocks. Each stimulus had the same number of lines, and the total number of characters varied by less than 5%. Two simple (S) and two difficult (D) versions of the stimuli were created by consulting the course content of the students. The expressions in the simple stimuli were a core part of their Java instruction in their first year, for example, '`Public class person{`' is exploited as an example of a simple statement stimuli since it was introduced early on in the undergraduate Java course. The expressions in the difficult stimuli are more specialist items that would only have been seen by the better performing students, for example, '`g.setFont(f);`' is used as an example of the difficult stimuli because not all participants can comprehend it, as it is not taught until later in the course.

The experiment was conducted using a standard graphics tablet (Wacom Intuos3) connected to a PC running a logging program specially written in our lab. Participants wrote with an inking pen on a response sheet. The response sheet was printed with a grid of 25 lines, each consisting of 42 spaces for writing individual characters. The sheet was designed for non-cursive writing in order to provide

rich inter-stroke pause data (see parenthetical note in the general design section above). Participants adapt easily to this writing style, and it does not seem to have a negative impact on other aspects of their performance (Cheng, 2014; Zulkifli, 2013).

Following the experiment, the participants completed a questionnaire with four parts (using an online survey platform). Part 1 included biographical questions relating to educational level. Part 2 assessed programming experience in general with five graduated rating items, such as 'I can develop programs using more than one object-oriented programming language'. Part 3 assessed Java programming expertise using eight graduated items, such as 'I am familiar with both objects and classes in Java'. Part 4 assessed participants' familiarity with the four specific Java stimuli provided to them during the experiment. Participants were asked to judge what their degree of familiarity would have been for each item prior to the experiment, on a 5-point Likert scale.

3.2.4 Procedures

Participants were instructed to use their preferred hand to hold the pen, and instructed to: begin writing at the beginning of each line, even for indented code; start writing as soon as the stimulus is revealed; copy the code as quickly and as accurately as they can; continue writing without correcting if they made a mistake; draw an upside-down triangle symbol (inverted capital delta) in place of spaces because spaces are important, and as I needed to record the time, nothing would be recorded if the participant left it blank; to start each trial with a hash (#) in order to ensure the validity of the pause for the first symbol; to start writing from the beginning of the line (write it line by line); to continue writing the same code statement in the next line in the response sheet, and when the same line of code exceeds a line in the response sheet, then start a new line for the next code statement; to hold down the special key to reveal the stimulus, with their preferred hand, ensuring that they write only when the stimulus key is released. The participants quickly became

fluent in the practice trials, so these conditions were not an especial burden (several of these conditions were needed for the pause measurements). Similar trial requirements were successfully used in previous experiments, so it is clear that they do not in themselves undermine the reliability of the results. The participants finished the experiment within an hour.

3.2.5 Stimuli design

In order to achieve a high level of experimental control, it was decided to use small code fragments. Each fragment consisted of three lines. In addition, as Miara et al. (1983) found, it is difficult to comprehend programs without indentation or with excessive indentation, so it was decided to keep indentation in this experiment stimuli because it will help high-competent and may not affect low-competent as they do not understand the program anyway. However, all participants were asked to ignore the indentation while writing and start from the beginning of each line. This is because I wanted better chunking signals, and because the number of indented spaces varies between each line of code, and I wanted participants to concentrate on the code material rather than counting how many spaces are at the beginning of each code line. The experiment had two factors: the first is participants' familiarity as low- and high-competence participants (between-subjects design); the second factor is stimuli difficulty, with the stimuli being either simple or difficult (within-subjects design). Four versions (S1, S2, D1, D2) were produced, as listed in

Table 3.1 below.

The next subsection discusses the two factors and justifies the selection of each stimulus. Each participant was presented with the four stimuli versions (i.e., two simple and two difficult). Two versions (S and D) were in VD presentation mode and the other two versions were in HS presentation mode.

Stimuli Version		Stimulus Content	No. Of Characters	No. Of Lines
S1	A	public class Person{ private String name; private int age;}	57	3
	B	public int Balance(){ int amountRefund; return amountRefund;}	61	3

	C	for(int h=0;h<hCount.length;h++)
		{
		System.out.println(h+hCount[h]);}
			66	3

		Total:	**184**	**9**
S2	A	public class Person{ public String name; public int age;}	55	3
	B	public void Balance(){ System.out.println("#"); Total += balance;}	64	3

	C	int h=0;
		while(h<hCount.length){
		System.out.println(h+hCount[h]);h++;}
			68	3

		Total:	**187**	**9**
D1	A	<body> Hello! The time is now <%=new java.util.Date()%> </body>	61	3
	B	int[] numbers= {1,1,3,5,8,13}; for(int item:numbers)	54	3

	C	cont=f.getContentPane();
		button=new JButton("Yes");
		cont.add(button);
			67	3

		Total:	**182**	**9**
D2	A	<body> <%! private static boolean visited=false; %> </body>	59	3

	B	int summation=0;
		for(int counter:arrayTall)
		summation+=;
			54	3

	C	Font f=new Font("S",Font.PLAIN,6);
		g.setFont(f);
		g.drawString("T",1,9);
			69	3

		Total:	**182**	**9**

Table 3.1 Stimulus versions (S1, S2, D1, D2) and the total number of characters and source code lines

3.2.5.1 *Stimuli difficulty factors: Simple (S) & difficult (D)*

The stimuli were chosen following a review of the students' programming modules. Java is the base language used for programming assignments in nearly all first-year modules. As shown

in

Table 3.1 above, the number of program code lines in every stimulus is identical. The number of characters in each stimulus version, e.g., S1 A and S2 A, is very similar; the total number of stimuli characters, e.g., S1, S2, D1, and D2, is also quite similar, with no more than 5% difference between them.

The reasons for selecting each stimulus are as follows:

1- S1 A & S2 A: consist of instance variables and data types. These stimuli declare class and instance variables of the same class, which is considered a fundamental concept. Students study it at a very early stage and must use it in any Java program. Therefore, it is categorized as S.

2- S1 B & S2 B: contain a return and void methods concept which is introduced to the students early in the course. A method is a set of program statements written to perform a specific operation. Students frequently use it because it helps in code reusability. For instance, *main* is a vital method in Java, as when a program is compiled and run, it is the first method to be executed. Thus, it is also a simple stimulus.

3- S1 C & S2 C: utilize embedded concepts: arrays and controls (iterations) using *while* and *for* loops for the same code. These notions are introduced to the students early in the first year. These stimuli are considered simple because they consist of simple and not nested loops. They also include frequently used code words.

4- D1 A & D2 A: the servlet JPS concept (i.e., creating dynamic web pages in Java) is introduced to third-year students in their first semester but is not covered in the first year. Also, it is not frequently used. Thus, it is considered difficult.

5- D1 B & D2 B: the iteration concept is introduced early in the course. However, this enhanced loop construction is not frequently used in the first and third years, especially for low-component programmers, unless students do programming as an activity beyond the course. In D1, the Fibonacci sequence is utilized in writing the array element as it is considered common knowledge for senior students and faculty, more so than for low-competent. Consequently, these stimuli are classified as difficult. It is assumed that high-competent will not find it difficult to transcribe and may recognize it as one chunk.

6- D1 C & D2 C: utilize the graphical user interface (GUI) notion which is introduced later in the first year, thus students do not practice them and it can be said that they are only somewhat familiar with it. Thus, it is classified as difficult. It is predicted that high-competent will produce shorter pauses than low-competent in this part.

3.3 Results

This experiment's results will concern the following: (1) the independent measures; (2) the test of the predictions associated with each of the main hypotheses; (3) further analysis of the effect of chunking; (4) the content analysis, which provides more detailed evidence for the claims that chunking occurs.

3.3.1 Independent measure of competence

(r)	Education level	General Programming	JAVA	Familiarity
Education level				
General Programming	0.366			
JAVA	0.183	0.759		
Familiarity	0.181	0.734	0.849	

Table 3.2 Correlations between competence measures (n=24, Pearson correlation, 1 tail, critical value is 0.472 at p<.01)

Questionnaire responses were coded to obtain independent competence measures against which to compare the chunk-based measures. Education level was scored on a scale from one to six (1 = first year undergraduate student, six = faculty member); an example of a question regarding this is "Have you learnt more than three programming languages?". General programming and Java experience were scored by giving one point for each positive answer related to the measure, so had scales from one to five and one to eight, respectively. Examples of the yes/no questions which were included in the questionnaire regarding general programming are "I am familiar with all the following programming concepts: declaring variables, conditions and iterations, and printing out a sentence on the screen" and "I can develop programs using more than one object-oriented programming language". Examples of yes/no questions regarding Java experience are "I am familiar with the 'main' method in Java" and "I am familiar with using GUI in Java". Ratings of familiarity were scored from 0 (low) to 4 (high), so with the four stimuli, the overall scale runs from zero to twelve. Participants were asked "Before you started the experiment, how familiar would think you were with these statements?". Table 3.2 above presents correlations between all combinations of the competence measures, and is unsurprising. Education level is only weakly (and not significantly) correlated to the other measures. General programming experience has a strong positive relation

to both Java experience and familiarity. The correlation between Java experience and familiarity is particularly strong. All this suggests that both Java experience and familiarity are specific to Java, rather than wider programming competence, and that either is suitable to serve as an independent measure. As the actual pattern of results is equivalent with either measure, just the analyses of the comparison using familiarity will be reported here.

3.3.2 Evidence of the role of chunking in a Java transcription task

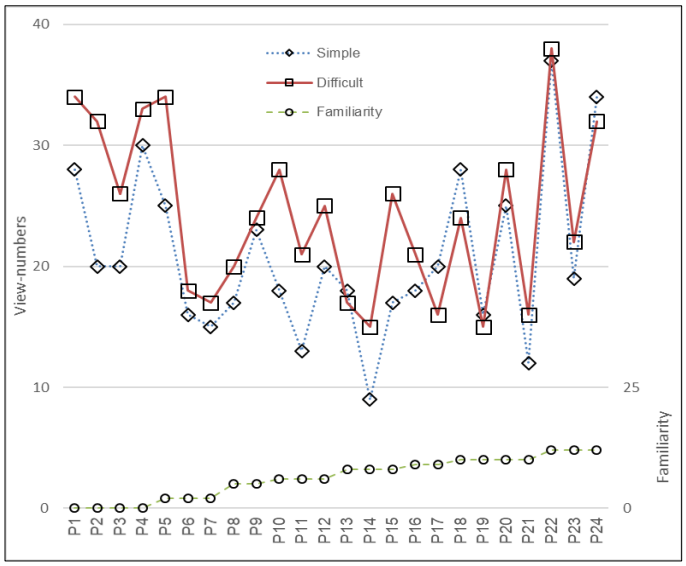

Figure 3.3 Total number of views for participants, ranked in order of familiarity, across simple and difficult stimuli

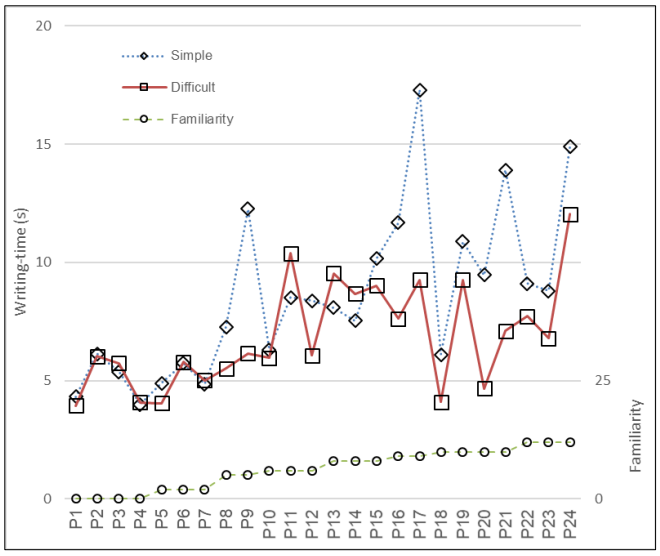

Figure 3.4 Median writing-times for participants, ranked in order of familiarity, across simple and difficult stimuli

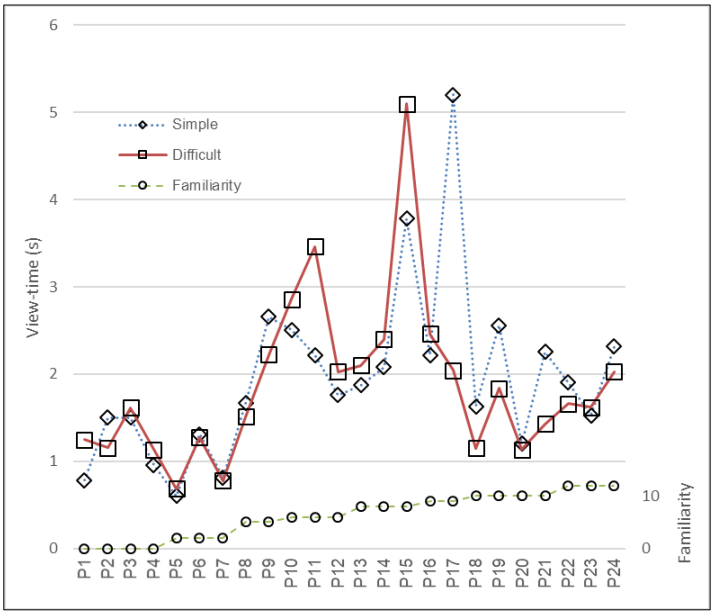

Figure 3.5 Median view-times for participants, ranked in order of familiarity, across simple and difficult stimuli

The view-numbers, writing-times, and view-times were computed from the logs of each participant for each trial. Figure 3.3, Figure 3.4, and Figure 3.5 above show the total view-numbers, median

writing-times, and median view-times for participants, ordered by their familiarity scores though simple and difficult stimuli, and we can see a fairly linear increase in the degree of familiarity across all participants. A binary split of participants' familiarity scores conveniently creates two equal size groups, with low familiarity scores exclusively below 6 and high score exclusively above 8. Further, there is a lot of variability between the participants. The overall trend of view-numbers is a decrease, as in Figure 3.3, where it is clear that the blue dotted line (simple stimulus) tends to be lower than the red solid line (difficult stimulus). However, there is an increase in writing-times, as in Figure 3.4, and the red solid line (difficult stimulus) tends to be lower than the blue dotted line (simple stimulus). Figure 3.5 shows that the general view-times trend is towards going up.

Figure 3.6, Figure 3.7, and Figure 3.8 below aggregate the data across low- and high-competence participants and across simple and difficult stimuli by showing the mean of the total view-numbers, the mean of the median writing-times and the mean of the median view-times.

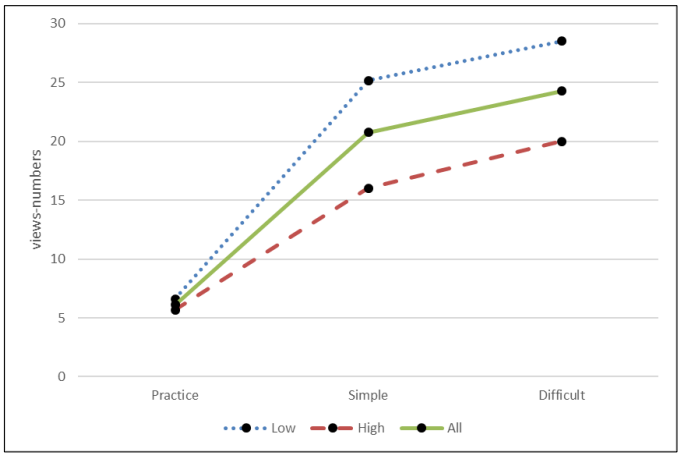

Figure 3.6 Mean view-numbers across stimuli difficulty types and levels of competence

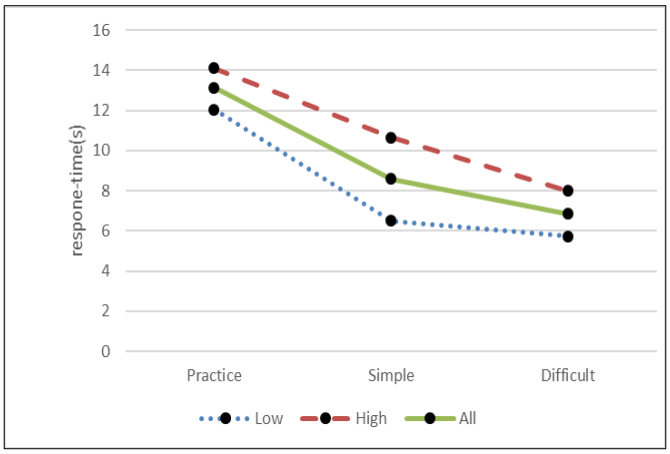

Figure 3.7 Mean of median writing-times across stimuli difficulty types and levels of competence

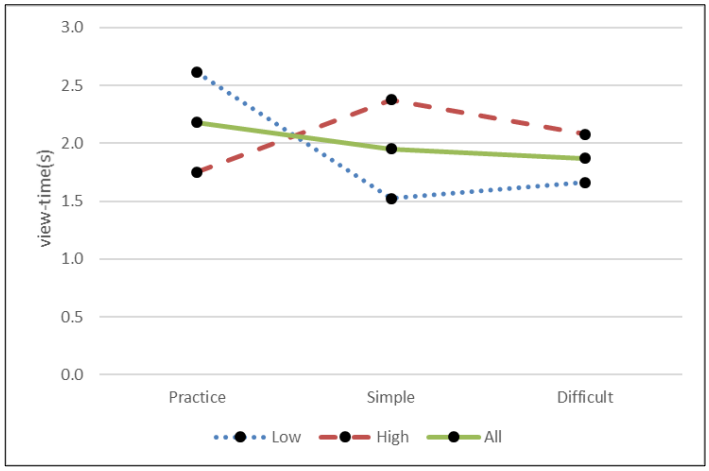

Figure 3.8 Mean of median view-times across stimuli difficulty types and levels of competence

The first thing to note is that in the total view-numbers (Figure 3.6) the value for practise items is significantly lower than for the Java stimuli, but it is essentially equivalent at low and high competence levels (6.6 and 5.7, respectively). Similarly, the mean of the median writing-times (Figure 3.7) for the practise items is somewhat longer than for the Java stimuli, and while the value is greater for higher and lower competence participants (means of 14.2 s and 12.1 s), it is not significant (by a t test; t=1.09, df=22, 1 tail, p=.24). These findings reassures that there is an impact of transcribing the Java stimuli that extends beyond simply transcribing any stimuli.

Consistent with prediction H1, which concerns the increase in the number of views with a decrease in the Java competence of the participants, Figure 3.3 and Figure 3.6 show that the high-competence participants required fewer views than those with low competence, which is significant at both levels of stimuli (simple: 16.3 vs 25.2, t=4.40, p=.0002; difficult, 20.0 vs 28.5; t=4.05, p=.0005; both df=22, 1 tail). Consistent with prediction H4, which predicted that the performance on simple stimuli would be superior to advanced stimuli with fewer view-numbers, Figure 3.3 and Figure 3.6 show that the simple stimuli require fewer views than the difficult stimuli across all participants (20.8. vs 24.3; t=4.05, p=.0003; df=22, 1 tail). For high-competence participants the view-numbers are still significant despite the small group size (16.3 vs 20.0; t=2.88, p=.016; df=10, 1 tail), but not for the low-competence participants (25.2 vs 28.5 s, t=1.8, p=.1, df=10, 1 tail).

Consistent with prediction H2, which focuses on an increase in writing duration with an increase in Java competence, Figure 3.4 and Figure 3.7 show that the high-competence participants had longer writing-times than those with low competence, which is significant at both levels of stimuli (simple: 10.7 vs 6.5 s, t=3.86, p=.0008; difficult, 8.0 vs 5.7; t=3.14, p=.005; both df=22, 1 tail). Consistent with prediction H4, the difficult stimuli had shorter writing-times than the simple stimuli across all participants (6.9. vs 8.6 s; t=3.29, p=.002; df=22, 1 tail). For high-competence participants only the writing-time is still significant despite the small group size (10.7 vs 8.0 s; t=3.0, p=.013; df=10, 1 tail).

Turning to prediction H3, which concerns the absence of an overall effect of view-times on Java competence, Figure 3.5 does not show a clear overall upward or downward pattern across all degrees of stimulus difficulty. If anything, the overall pattern is in contrast to the trends in Figure 3.3 and Figure 3.4. Figure 3.8 above reveals that high-competence participants have longer view-times than those with low competence, but this is not significant for the difficult stimuli (2.1 vs 1.7 s; t=1.50, p=.15, df=22, 1 tail), and marginally significant for the simple stimuli (2.4 vs 1.5 s; t=2.62, p=.02, df=22, 1 tail). Further, comparing the view-times on the practice stimuli with the Java stimuli

view-times we see they are similar, whereas for view-numbers and for writing-times the practice values are quite different to Java stimuli values, as noted above. Consistent with prediction H4, Figure 3.5 shows that there were nearly equal numbers of participants with longer view-times for the simple stimuli and longer view-times for the difficult stimuli. In terms of the means across all participants (Figure 3.8), no significant differences occur for the simple stimuli (1.5 vs 1.7, t=1.03, p=.3, df=22, 1 tail) nor for the difficult stimuli (2.4 vs 2.3; t=1.21, p=.25; df=22, 1 tail).

In summary, all predictions are supported in terms of view-numbers, writing-times, and view-times.

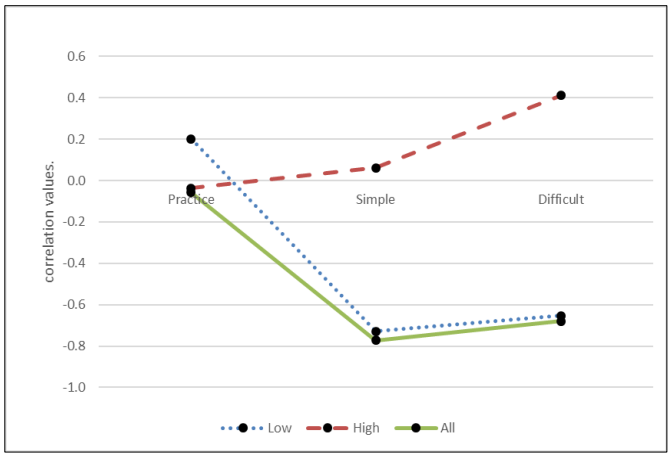

Figure 3.9 Correlation of view-numbers with familiarity across simple and difficult stimuli for all participants, low and high competence

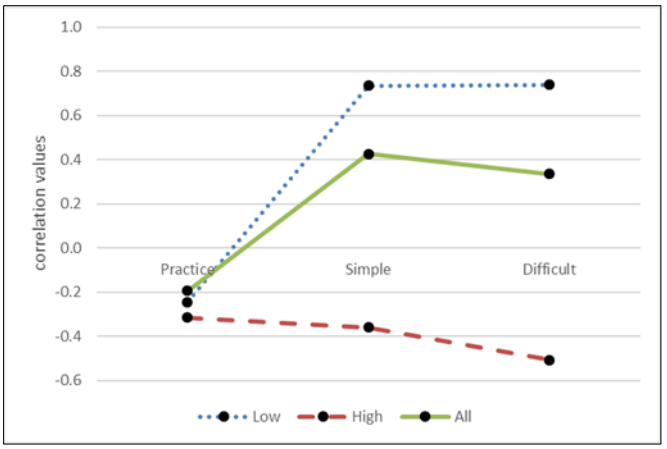

Figure 3.10 Correlation of writing-times with familiarity across simple and difficult stimuli for all participants, low and high competence

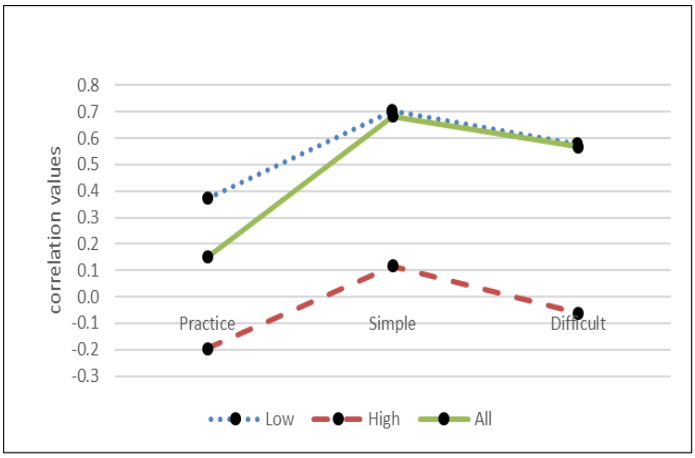

Figure 3.11 Correlation of view-times with familiarity across simple and difficult stimuli for all participants, low and high competence

Figure 3.9, Figure 3.10, and Figure 3.11 show the Pearson correlations of familiarity scores with, respectively, view-numbers, writing-times and view-times, which confirm what appears in Figure 3.3, Figure 3.4, and Figure 3.5 and further support the four predictions H1, H2, H3, and H4. The scale ranges are not the same. For correlations over all participants (solid (green) line in Figure 3.9–Figure 3.11) the critical value is 0.344 for significant correlations at $p<.05$, and 0.472 at $p<.01$ (1 tail, df=22).

For correlations with just high competence or low competence (dashed or dotted lines) the critical value is 0.497 at $p<.05$ and 0.658 at $p<.01$ (1 tail, df=10).

As expected, none of the correlations for the practice items are significant. With view-numbers (Figure 3.9), across all participants the negative correlations are strongly significant: the number of views decreases with an increase in familiarity score (i.e., competence), which matches prediction H1. The result is similar when just the low-competence group is considered, but correlation for the high-competence participants is positive but not significant. For writing-times, the pattern of results is similar but the direction of the correlations is reversed (Figure 3.10): writing-time increases with an increase in familiarity score, consistent with prediction H2. For the whole group and the low-competence sub-group, the correlations for the difficult stimuli are lower than for the simple stimuli.

The view-time correlations (Figure 3.11) for the whole group and the high-competence sub-group are not significant, but the correlations of the low-competence participants are strong for both Java stimuli difficulty levels, simple and difficult.

In summary, correlations between view-numbers, writing-times, and view-times are overall consistent with the four predictions, with some divergence in detail. In particular, view-numbers and writing-times did not differentiate high-competence participants. Also, view-times did unexpectedly differentiate low-competence participants, who need more views with increasing competence.

3.3.3 Regression analysis for writing-times and view-times against view-numbers

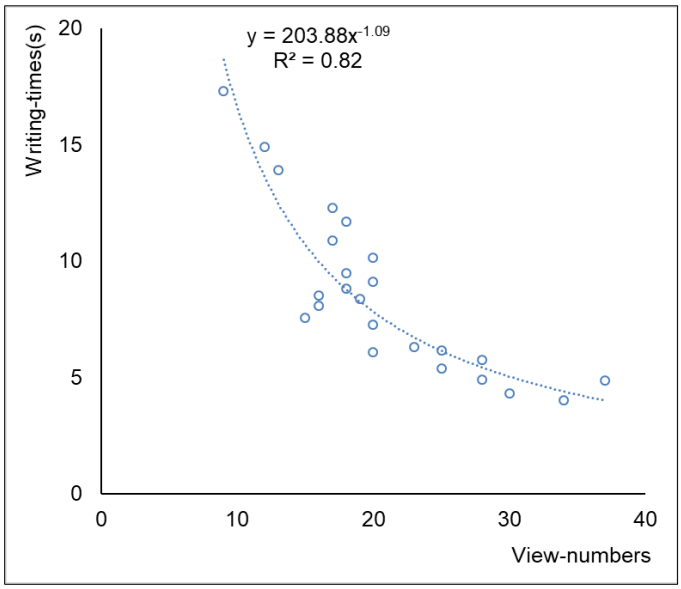

Figure 3.12 Relation of writing-times to view-numbers (simple stimulus)

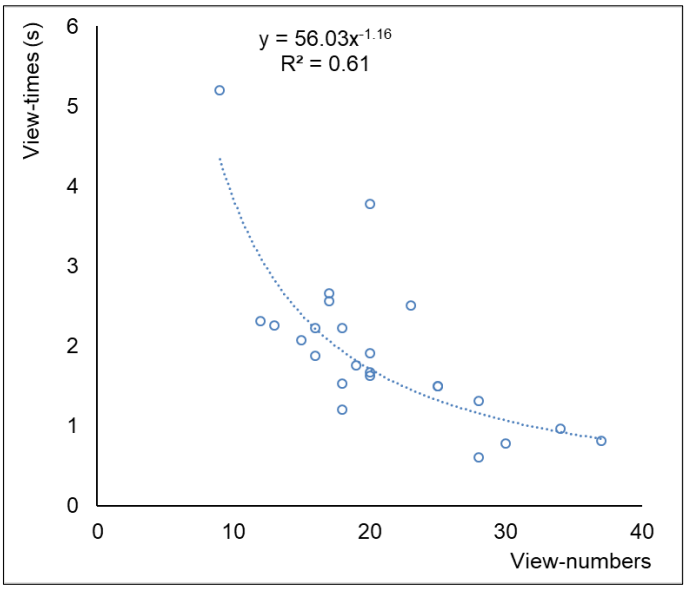

Figure 3.13 Relation of view-times to view-numbers (difficult stimulus)

	Writing-times & view-numbers			View-times & view-numbers		
	Practice	Simple	Difficult	Practice	Simple	Difficult
Index, i	-0.95	-1.09	-0.97	-1.05	-1.16	-1.24
Constant, C	57.9	203.9	136.5	8.9	56	83.7
R^2	0.459	0.818	0.747	0.603	0.615	0.623

Table 3.3 Parameter of best-fit power relation for writing-times and view-times to view-numbers

The relationship between the three main behavioural measures (view-numbers, writing-times, view-times) are examined because a systematic relationship between them could provide further support for the hypotheses and more precise chunk-based explanations of the results. View-numbers and writing-times are both predicted to be dependent upon chunking processes; as a result, there should be some kind of stable and formal relationship between them. View-times are

not expected to be chunk-dependent, so no regular relation between them and view-numbers (or writing-times) is anticipated.

To investigate further whether chunking might explain the differences between writing-times and view-times, scatter plots of these variables against view-numbers were plotted for all participants in all the conditions of the experiment. Figure 3.12 above plots writing-times versus view-numbers for the simple stimuli and Figure 3.13 above plots view-times against view-numbers for the difficult stimuli. Table 3.3 above presents the parameters of the best fit equations for both relations of writing-times and view-times against view-numbers across all stimuli, along with the R^2 values.

The pattern of data in Figure 3.12 has a particularly distinctive shape, which is also apparent in the graph for the difficult stimuli. Hence, the power law for writing-times versus view-numbers is notable, over both stimuli difficulty levels, simple and difficult: a power law curve was perfectly fitted to the data and the R^2 values are large. This implies that writing-times is the more reliable measure to predict view-number. The relation between view-times and view-numbers is less clear, with lower R^2 values.

3.3.4 Further evidence of the role of chunking in Java comprehension

When participants study Java, they begin to recognise and group common code words and syntax. When they see the stimulus, they recognise these common code words as a single chunk. Content analysis was carried out to provide more detailed evidence for the claims that chunking occurs (as presented in Chapter 2), as well as to support the claim that chunking does not occur in random positions, but that it happens at the border of meaningful things.

I investigated the identity of each stroke that participants transcribed in order to know where exactly the breaks were in the stimulus, and as a result what the exact contents of the participants' chunks are. For example, a low-competent may break the `While(h<hCount.length){` stimulus

as follows, `While--- (h --- <h --- Count --- .length --- ){`, whereas an high-component may break it as `While(h< --- hCount.length){`. The chunk's contents can be obtained by locating the beginning of each view, extracted from the written logs of the participants.

The main goal of conducting a content analysis is to determine which content categories had the most breaks. Since each category has a unique code, the output of this analysis will show the number of breaks that occurred for each category in each stimulus. This will then help to determine whether or not breaks occur at the border of meaningful things (explained in more detail later on). To do this, four major data sets were used: (1) a stroke log; (2) a view log; (3) a model file that includes the sequence of the whole stimuli characters; and (4) an actual log, written characters of individual participants. The first two logs are from the written logs; the stroke log includes the times when the pen is on the paper and when it is picked up, as well as the precise positions of each stroke using X and Y coordinates (i.e., tablet and response sheet). The view log contains the number of times a participant viewed the stimuli (i.e., breaks) as well as the precise time for each view. The model file contains the expected written characters for each stimulus. Finally, there is the actual written character log, which includes the times and positions of each stroke for each participant's trial.

So, in the end, there were four coded (i.e., model) stimuli that were combined with the model files (as explained earlier). The file containing the real written characters was then compared and adjusted with the model file for each stimulus for each participant. At this point, and as previously reported, I could specifically compare each character in the model files with each actually written character. As a result of this procedure, I knew which exact characters were associated with each break, because I could see where each participant wanted to view the stimulus and the content of each chunk for each participant. Some participants' individual logs included problems such as missing characters, wrong letters, or the insertion of additional characters that did not appear in the

model files. It is worth noting that this analysis was performed manually and required a substantial amount of time and effort.

A coding scheme was established for different content categories in the stimulus, as shown in Table 3.4 below. The coding scheme relates to sets of characters that are likely to be chunks that someone familiar with Java would see as meaningful. Each unit under the text or punctuation mark categories is assigned a unique code; for example, the 'beginning of a variable type' such as *int* or *float* in the actual log is assigned the code '3000' and the punctuation '[' is assigned the code '100', etc.

Some participants on occasion view a stimulus without any associated writing before the next view. It is worth noting that content analysis allowed me to compute a view-related measure; the pattern of results obtained with this measure is essentially identical to that obtained with view-numbers. This type of view was not apparent in my initial research, but I discovered it after conducting the content analysis. All this implies the exact location of each participant's views (i.e., breaks) in each stimulus, thus the precise content of participants' chunks of each stimulus is acquired.

CODE	Categories	Calculated
Text		
1000	Beginning Reserved Word	Yes
1001	Within Reserved Word	
2000	Beginning Var	Yes
2001	Within Var	
3000	Beginning Var Type	Yes
3001	Within Var Type	
4000	Beginning Access Modifier	Yes
4001	Within Access Modifier	
5000	Beginning text	Yes
5001	Within text	
Punctuation Marks		
100	[	Yes
101	]	
110	{	Yes
111	}	
120	(	Yes
121	)	
130	<	Yes
131	>	
200	;	Yes
210	:	Yes
220	.	Yes
230	" (open)	Yes
231	" (close)	
240	,	Yes
300	/	Yes
310	+	Yes
320	%	Yes
330	!	Yes
340	=	Yes
600	Strokes within characters	Yes
700	Capitals within reserved words or variables	Yes
Spaces	Space	Yes

Table 3.4 Coding scheme for various types of stimulus content (code content categories)

No.	Abbreviation	Definition
1	No.Bs(S)	The number of breaks for this particular category
2	BsPerCate(D)	The number of breaks for this particular category / the number of times this particular category exists in the trial *100

Table 3.5 Definition of each content analysis measure, S means simple and D means difficult

Correlations	Reserved & Variables	PunctuationMarks	CapitalsWithinWords	StrokesWithinChars	Spaces
BreaksNo.(S)	-0.32	-0.72	-0.42	-0.13	-0.6
BreaksNo.(D)	-0.18	-0.63	-0.62	-0.09	-0.37
BreaksPerCate(S)	-0.36	-0.59	-0.37	-0.17	-0.55
BreaksPerCate (D)	-0.11	-0.49	-0.62	-0.08	-0.39

Table 3.6 Correlation between familiarity scores and number of breaks associated with each category (n=24, Pearson correlation, 1 tail, critical value is 0.472 at p<.01)

The BreaksNo. and BreaksPerCate measures were introduced to help in achieving the main goal of this analysis (i.e., chunking happens at the border of meaningful things), via counting the break numbers and percentages of each content category. Before looking deeper into the findings of this analysis, it is important to consider the meaning of the two measures BreaksNo. and BreaksPerCate, as seen in Table 3.5. When analysing the data, all symbols such as < and ; are combined as one category, PunctuationMarks. Reserved words and variable names were integrated as one group (Reserved & Variables) as both are combination of alphabetic characters. Thus, there are five codes for the content categories: Reserved & Variables (such as class, age); PunctuationMarks (such as [, {>); CapitalsWithinWords (such as hCount and JButton); StrokesWithinChars (such as writing the letter t in 2 stages '–' and 'I'); and Spaces, as shown in Table 3.6 above. Further, when it says 'yes' in the last column of the Table 3.4, it implies that this category is taken into account in the final calculation of the categories that have been assigned breaks.

After aggregating this enormous amount of detailed data, correlation values between participants' familiarity scores and the number of breaks associated with each code content category ('S' and 'D' in Table 3.6 mean simple and difficult stimulus), as shown in Table 3.6. Specifically addressing the five content categories in Table 3.6, regarding (1) punctuation marks, there is a significant negative relationship between familiarity and BreaksNo. that is correlated with punctuation marks for both simple and difficult stimuli. To put it another way, the less qualified the participants are, the more breaks identified with punctuation marks they take. Furthermore, BreaksPerCate. and BreaksNo. have similar correlation values, but BreaksNo. has slightly more values. Regarding (2) capitals within reserved words and variables, there is a robust negative relation between both BreaksNo. & BreaksPerCate and participants' familiarity (i.e., there are more breaks associated with capitals for low-competence participants than high-competence participants). Low-competence participants tend to split up single variables or reserved words (i.e., which are considered as a single word) more than high-competence participants, only for difficult stimuli, although nevertheless there is a weakened relation for simple stimuli. The D stimuli are more difficult for low-competence participants than the S stimuli. In terms of (3) the spaces category, there is a strong negative relation between participants' competence and BreaksNo., which is associated with spaces (i.e., low-competence participants are more likely to have more breaks associated with spaces than high-competence participants). Again, this is similar with BsPerCate.

On the other hand, there is no chunking effect in the (4) Reserved & Variables and (5) StrokesWithinChars code content categories (i.e., baseline condition); as shown in Table 3.6, they have no significant correlations with smaller values. In other words, for all participants, there is a weak relationship between the number of breaks and Reserved & Variables and StrokesWithinChars.

In summary, as previously noted (earlier analysis at the beginning of the Results subsection), low-competence participants have more view-numbers (i.e., breaks) than high-competence participants for both simple and difficult stimuli; accordingly, the correlations between the number of breaks and Java familiarity in Table 3.6 are generally negative, which is unsurprising. In other words, as Java competence increases, the number of breaks associated with each code's content category decreases. Moreover, I found, as shown in Table 3.6, that the breaks are more associated with the punctuation marks than with Reserved & Variables for both simple and difficult stimuli. More precisely, breaks are more closely aligned with the PunctuationMarks, CapitalsWithinWords, and Spaces categories, and hence these categories give a stronger indicator of Java competence than Reserved & Variables and StrokesWithinChars.

3.4 Discussion

The overall purpose of this experiment was to see whether previously proven chunk-based approaches to evaluating competence by copying in domains such as maths and second language learning can be extended to programming. The aim was also to see if the view-numbers, writing-times, and view-times measures (HS technique) can extend the repertoire of techniques that have been used (i.e., VD mode (pauses)) to assess chunk structures.

This section will answer the three questions and go through the four hypotheses raised in the introductory part of this chapter. It is important to note that the analysis here will concern view-numbers, writing-times, and view-times (HS behavioural measures) only. This section includes four subsections: (1) how this experiment extends similar studies; (2) the evidence of the role of chunking in a Java transcription task; (3) the content analysis outcomes and implications; (4) suggestions and implications for future experiments.

3.4.1 How this experiment builds on previous findings

This experiment contributes a method for evaluating competence in programming using a transcription task and measures with timescale of 10 s. This extends the range of techniques beyond the pause distribution measures of previous work (e.g., Cheng, 2014, 2015; Cheng & Rojas-Anaya, 2007).

This experiment extends previous researches findings in three ways.

First, allowing the user to expose and hide the stimuli at will allows two alternative temporal chunk measures to be captured: view-numbers, the total number of views of the stimulus in a trial; writing-times, the median duration of writing time between views. Predictions H1, which concerns the decrease in the number of views with an increase in Java competence, H2, which focuses on increasing the length of written responses as the Java level increases, and H4, which refers to the superior output on simple stimuli compared to advanced stimuli, with fewer view-numbers and longer writing-times, are well supported by converging evidence. These behavioural measures are strongly correlated with the independent measure of competence (familiarity). No support for view-times as a suitable measure of competence was obtained, as predicted in H3, despite the relative strength of the effects for the view-numbers and writing-times measures. The HS measures' correlations that were found in this experiment are at the same level as the pause correlations from previous studies (Cheng, 2014; van Genuchten & Cheng, 2010; Zulkifli, 2013).

Second, the experiment shows that measures based on temporal chunk signals are applicable beyond mathematics (algebraic formulas) and natural language, in a domain that happens to share some characteristics of both those domains. This is not surprising, but it is certainly reassuring that the approach works for programming. So, I can answer 'Yes' to my first question, 'Could chunk-

based measures of comprehension be extended to domains other than mathematics and natural language learning?'.

Third, in contrast to the single-line stimuli used in the previous experiments mentioned above, the present stimuli were larger (nine lines). A greater amount of data per trial means that single trials can yield strong usable correlations with competence, without the theoretical problem of deciding how to aggregate data from multiple trials or the practical problems associated with switching between multiple trials.

3.4.2 The relationships between the dependent behavioural measures

Overall, for all participants, correlation of view-numbers and writing-times with the independent Java competence measure (i.e., familiarity) are strong, and this also holds for the low-competence group. The greater the familiarity scores, the smaller the view-numbers, and the longer the writing-times. Furthermore, it is clear from Figure 3.3 and Figure 3.4 that there is considerable variability between participants, such that some of the best low-competence participants have better scores than many of the high-competence participants, and vice versa. Clearly the development of a real educational test of programming competence must address the accuracy and sensitivity of the measures.

The correlations between view-numbers, writing-times, and view-times are overall consistent with my four predictions, with some divergence in detail. View-numbers and writing-times, in particular, differentiate low-competence participants but not high-competence participants. In other words, the difficulty of the difficult stimuli may be insufficient to differentiate those within that group. This seems plausible, in hindsight, as the range of difficulty captured in the stimuli was based on the undergraduate Java programming curriculum, but a proportion of the participants were drawn from more senior groups. This plateauing was also seen in previous studies (Cheng, 2014, 2015). One

implication of this is the importance of designing stimuli with a sufficient range for the target test group.

For view-times, there are two unexpected findings (which contradict hypothesis H3). The first is the clear positive correlation of view-times with Java competence for low-competence participants; the correlations are not significant for the whole group and the high-competence group. High-competence participants had longer view-times than low-competence participants for both simple (marginally significant) and difficult (not significant) stimuli. The second is the increase in view-times with decreasing view-numbers (Figure 3.13); theoretically, there should be no connection between the two. My current assumption is that disparities in view-times are caused by different stimulus encoding procedures utilised by participants, which vary depending on the type of stimulus content. However, the relations between view-times and familiarity are not strong as for view-numbers and writing-times.

The clear negative relation between writing-times and view-numbers (Figure 3.12 and Table 3.3) supports the chunk-based hypotheses underlying predictions H1 and H2, which concern the decrease in view-numbers and increase in writing-times as Java familiarity scores increase. The poor fit of such a power law for view-times vs view-numbers is consistent with prediction H3, which claims that view-times have no impact on Java competence. This means that the basic process in transcription tasks appears to be the selection of chunks from the stimulus, with more competent participants keeping more characters – since they have larger chunks – and that the time required for writing is proportional to the number of characters. Nevertheless, Figure 3.3 and Figure 3.4 show much individual variability, so a useful line for future work would be to investigate the possibility of separately measuring the working memory capacity of participants in order to consider whether there is a need to devised methods to normalize for them.

Finally, this experiment allows me to respond ‘yes’ to the questions mentioned in the introduction subsection of this chapter: is it possible that handwriting program code provides powerful and stable temporal chunk signals that can be used to assess programming comprehension? Can programming comprehension be accurately measured using view-numbers, writing-times, and view-times?

3.4.3 Content analysis

Content analysis was used to supplement the first analysis, allowing me to determine where participants chose to view the stimulus each time they finished writing the previous chunk content. This ensures that the precise content of each stimulus is revealed for each participant. As a participant’s Java competence increases, the number of breaks associated with each stimulus (Reserved & Variables, PunctuationMarks, CapitalsWithinWords, StrokesWithinChars, and Spaces (i.e., content categories)) decreases (Table 3.6).

Moreover, it was found, as shown in Table 3.6, that the breaks are more associated with the punctuation marks than with Reserved & Variables for both simple and difficult stimuli. One of the key distinctions between low- and high-competence participants is that low-competence participants had more breaks associated with punctuation than high-competence participants. More precisely, breaks are more closely aligned with the PunctuationMarks, CapitalsWithinWords, and Spaces categories, and hence these categories give a stronger indicator of Java competence than Reserved & Variables or StrokesWithinChars. To explain why these three categories (PunctuationMarks, CapitalsWithinWords, and Spaces) have more breaks than the others, consider the following: (1) for PunctuationMarks, this is not unexpected given that recognisable ordinary words, that have everyday meanings other than in Java, can be recognised by anybody, while punctuation is more difficult to remember; (2) for CapitalsWithinWords, if a capital character

appears inside a word, low-competence participants break the word because they recognise it as a different (i.e., new) word, while high-competence participants recognise it as a whole word; finally, (3) for the Spaces category, low-competence participants were more likely than high-competence participants to take more breaks that are correlated with spaces. In other words, highly competent participants can recognise, for example, three words divided by spaces as one block rather than three distinct words, since they understand the connections between the words and, as a result, understand the general context. Low-competence participants, on the other hand, interpret them as three disconnected words because they do not understand the links between the words (i.e., the whole meaning). As a result, participants with low competence had more breaks correlated with spaces than participants with high competence.

To sum up, the experiment has verified that it is feasible to use view-numbers, writing-times, and view-times displayed using the hide and show technique in a simple freehand transcription task as a method of measuring competence in a rich information domain such as programming. The correlations found in this experiment are at the same level as the pause correlations from previous studies (Cheng, 2014; van Genuchten & Cheng, 2010; Zulkifli, 2013). Therefore, in this experiment, it was proved that there is another, complementary way (rather than the pause measure) to obtain a competence measure based on chunking.

3.4.4 Suggestions and future work

This experiment has several implications which are worth considering in further work:

- The level of competence for the target participant group must be considered when designing the experiment stimuli.
- Implement an independent memory test to assess participants' working memory ability apart from their transcribing speed.

- Use stimuli that contain more punctuation marks, capitalization within words, and spaces.

3.5 Summary

- It is maybe possible to use view-numbers and writing-times recorded in HS display mode with simple freehand transcription tasks as a method of measuring competence in such a rich information domain as programming.
- Similar to previous studies, normalising for practise items did not improve the measures (i.e., processing speed is independent of programming competence).
- According to the content analysis, participants break more before punctuations, spaces, and capitals within words.
- There were significant relationships found: view-numbers decrease with the increase of familiarity scores and writing-times increase with the increase of the familiarity scores.
- Significant relationships between view-numbers and writing-times were discovered.

4 Experiment 2: Improved stimuli design

Chapter content:

- An introduction that contains the hypothesis and main questions.
- The experiment methodology:
  - General experiment design.
  - Classification of participants.
  - Experiment applied materials.
  - Stimuli design.
  - The experiment procedures.
- Results part 1:
  - Independent measure of competence.
  - Evidence of the role of chunking in Java transcription task.
  - Regression analysis for the behavioural measures against familiarity scores.
  - PPCS normalisation.
  - Regression analysis for writing-times and view-times against Characters per view.
- Discussion part 1:
  - Checking the behavioural measures performance.
  - Stimuli design.
- Results part 2:
  - Complexity analysis.
  - Discrimination between the two complexity levels across all the behavioural measures.
- Discussion part 2.
- Overall discussion
- Summary.

4.1 Introduction

The first experiment (Chapter 3) showed that assessing the structure of chunks is a potential method for evaluating programming comprehension in a transcription task. It demonstrated that the more familiar participants are with Java, the longer their writing-times and the fewer view-numbers they produce. It demonstrated as well that the independent measure (familiarity) can significantly predict view-numbers and writing-times. On the other hand, due to the previous experiment's shortcomings, the following points were considered for the current experiment. (1) I planned to collect Q3 pause data from the view display (VD) mode, as this was not collected in the previous experiment due to an obscure software-hardware interaction on the experimental computer, which resulted in unrecorded pauses. (2) I redesigned the stimuli, since the advanced stimuli were inadequate to distinguish participants within the high-competence group, and the content analysis was applied (Chapter 3), indicating that the distinction between participants would be clearer if the stimulus contained more punctuation and a reduced number of letters/terms, i.e., variable words and reserved names.

Characters per view would be used to calculate view-numbers in this experiment (characters per view is calculated by dividing the total number of characters of a stimulus by the view-numbers per trial). This is due to the fact that, unlike the previous experiment, the stimulus length of this experiment varies. Thus, in order to have equivalent stimulus length, characters per view is used here, and thus the characters per view metric equals view-numbers.

The first experiment is replicated here in terms of experimental design, through use of the same presentation factor, hide and show (HS; where the stimulus is only made visible when a participant presses a button), to obtain the view-numbers (the total number of views of the stimulus in a trial), writing-times (the time spent writing between two successive views), and view-times (the duration

of each look at the stimulus). In addition to HS, view display (VD; where the stimulus is visible at all times) is used as a presentation method in this experiment to capture participants' pauses. Participants, as in VD, press the special button just once to make the stimulus visible before beginning to transcribe the program code. This approach ensures the reliability of the length of the pauses before each character. What's new in this experiment is that I attempted to recruit a larger sample of participants by recruiting more undergraduate students.

Since the previous experiment's results matched the views and writing-times generation model (see section 3.1 in Chapter 3), I planned to test it again in this experiment to see if better correlations between behavioural measures and familiarity can be obtained with the new stimuli design in this experiment. Thus, the hypotheses H1, H2, H3, and H4 are the same as in the first experiment (Chapter 3) and will be repeated here for convenience.

Given that the size (i.e., the number of characters) of each Java code stimulus is set, I predict:

H1) Characters per view for a particular stimulus in a trial will be higher for more competent participants.

Since the chunks of a more knowledgeable participant contain more information, I predict:

H2) Writing-times after each stimulus view will be longer for more competent participants.

In the first experiment I predicted that there would be no relation between view-times and programming competence, however the results showed an unexpected relation (strong positive correlations of view-times with competence, specifically for low-competence participants). Thus, I make the same prediction again except this time I paid close attention to whether the strange relationship still exists.

Since the time required to comprehend a chunk is roughly fixed (Chase & Simon, 1973), and if the time retained per view is unrelated to competence, then I predict:

H3) View-times will not be directly related to programming competence.

I will make certain that pauses are successfully logged in this experiment, as they were not in the previous one (Chapter 3). Thus, I will be able to compare $pause_{Q3}$ with characters per view, writing-times, and view-times in this experiment. I was seeking to answer the following question:

Is $pause_{Q3}$ a valid measure for assessing programming comprehension in a transcription task?

Thus, this experiment has two more hypotheses (H4 and H6) than the previous one.

As predicted by the pause generation model (section 2.4.4 in Chapter 2) and as a pause is the duration before beginning to write each stroke/character in VD, I predict:

H4) Pauses will be shorter for more competent participants.

In this experiment, I used basic and advanced stimuli again; as basic Java concepts (which were introduced to students in the early stages of their first year) are used for the basic stimulus, I predict:

H5) Participant's performance on basic stimuli will be higher than on advanced stimuli.

According to chunking theory, each person can have his or her own chunk size (i.e., number of elements). Two people, for example, may have the same degree of experience on a topic. However, one has a greater working memory capacity than the other, and the person with a larger capacity can appear to have improved Java comprehension in the tests, not because they are more expert, but because their normal working memory capacity is larger. So, there is a theoretical reason why measuring their chunk size will be beneficial. I hoped to improve the behavioural measures by normalising them using Participant Preferred Cluster Size (PPCS), which is the total number of characters participants transcribe each time they view the stimuli, in order to remove any individual

differences that might influence the measures. The PPCS test uses arbitrary letters and numbers to estimate working memory size, used as an individual baseline for each participant. According to the outcomes of this side experiment (Cheng & Albehaijan, 2020), participants' PPCS varied between two and even up to six elements. I found that the number of elements varies both within a participant and across different participants. I measured PPCS using performance on the PPCS stimuli S1 as a baseline, as justified in Cheng and Albehaijan (2020).

In addition, PPCS tests were applied using the same protocols as the Java test in order to achieve the same temporal chunk signal measurements as the Java stimuli. As a result, PPCS data can be evaluated in the same manner as the Java data. The main difference is in the content of the stimulus; PPCS stimuli are made up of randomly ordered numbers and letters (an alphanumeric string) and are delivered in four forms: S1, S2, S3, and S4 (further explained in the methodology section).

So, I was seeking to answer the following question:

> Can characters per view, writing-times, view-times, and Q3 pauses be improved by normalizing using PPCS?

After assessing participants' basic transcription performance, I predict:

> H6) The behavioural measures will be improved after normalizing them for the PPCS measures.

The relationship between the three HS measures (view-numbers, writing-times, view-times) were evaluated in the previous experiment to provide evidence that the transcription process is based on chunking. This in turn gives more confidence in the interpretation of the results of the competence measures. I found a significant relation between writing-times and view-numbers. Therefore, for this experiment, and as I am using characters per view instead of view-numbers, I predict:

H7) Writing-times will be directly related to characters per view. The longer the writing-times (the more time participants spend transcribing), the more characters per view they will produce.

This chapter includes three main sections: Methodology, Results, and Discussion. It consists of two 'results and discussion' parts. As shown in

Table 4.1, Part 1 uses the difficulty factors basic (B1, B2) and advanced (A1, A2). Part 2 uses the complexity factors simple (B1.1, B2.1 and A1.1, A2.1) and complex (B1.2, B2.2 and A1.2, A2.2). The analysis Part2 was developed because the findings of the analysis in Part 1 were not as good as in the first experiment (Chapter 3), and I attempted to analyse this. The results of this analysis are given in Part 2. The majority of the structure of this experiment is the same as in experiment 1; certain sections have been replicated for ease of reading.

4.2 Method

4.2.1 General experiment design

This experiment includes two main parts: (1) PPCS test and (2) Java tests. For both parts, two presentation factors were applied, VD and HS. The PPCS test consists of eight stimuli (four in VD and four in HS), presented to the participants in the same order, from S1 through to S4.

With regards to the Java test, the experiment is a counter-balanced 2x2 design (two display factors, VD and HS, and two difficulty factors, basic and advanced). It contains four stimuli (each was made up of nine lines divided into two sperate blocks), two basic and two advanced: a within-participant

factor with each participant transcribing basic and advanced stimuli, and a mixed factor with low- and high-competence participants and basic and advanced stimuli.

4.2.2 Participant classification

The participants were 51 adults from the School of Engineering and Informatics at the University of Sussex. Recruitment spanned first-year undergraduate students through to faculty members (with a wide range of Java knowledge), to obtain a good range of programming expertise. There were 28 students who had just begun the Java module, 15 students who had done some Java programming, and some who had completed the first Java module, seven postgraduate students, and one faculty member. Participants ranged from 18 to 59 years of age (mean = 21.922, SD = 6.378), and 35 were male, 15 females, and one not specified. They received £8 for participating.

4.2.3 Materials

The experiment was carried out with a tablet connected to a PC running a logging program written specifically for our lab. Participants wrote with a special inking pen on an A4 response sheet. The sheet was printed in landscape orientation (i.e., crosswise, not lengthwise) with a net of 17 lines, each consisting of 42 spaces for the writing of separate characters. It was designed for non-cursive writing in order to provide rich inter-stroke pause data.

After completing the transcription task, participants completed a four-section online questionnaire designed to offer an independent assessment of their Java competence. Section 1 asked four biographical questions about educational level. Section 2 used four graduated ranking items to measure programming expertise in general, such as “I can develop programs using more than one object-oriented programming language”. Section 3 used eight graduated items to assess the level of Java programming expertise, such as “I am familiar with both objects and classes in Java”. Section 4 assessed participants’ experience with the Java stimuli used in the experiment by asking them to

judge “what their degree of familiarity would have been for each stimulus before starting the experiment”, on a 5-point Likert scale.

4.2.4 Stimuli design

Why was the Java programming language chosen for this research? This was because Java is the base language used for programming assignments in nearly all first-year modules for undergraduate participants. As shown in

Table 4.1 below, four stimuli versions were chosen after consulting the content of the Java modules taken by the student participants. The expressions for the basic stimuli were an important part of their Java curriculum during their first year. The expressions for the advanced stimuli are more advanced items that may have previously been used by the top students. The content analysis results (Chapter 3) helped in improving this experiment’s stimuli design. The new stimuli were designed to increase the number of chunks by replacing variable and reserved words with expressions with more symbols and punctuation.

Regarding stimuli content, the two practice items consisted of a series of simple statements, such as ‘Sussex University’, and ‘Computer Science’. The PPCS test consisted of two trials, each PPCS stimulus containing 40 randomly ordered letters and numbers (presented in one line) such as ‘k 6 m 4 i 7 h 2 z …’, as shown in Cheng & Albehaijan (2020).

For each Java stimulus, there were two code blocks, and each block contained four or five lines. Indentation was kept in this experiment as well, for reasons mentioned in the first experiment (Chapter 3).

Stimuli Version		Stimulus Content	No. Of Characters	No. Of Lines

B1	B1.1	import java.util.Scanner;
		class A{
		public static void main(String[]args){
		Scanner s=new Scanner(System.in);
		double l=s.nextDouble();}}
			122	5

	B1.2	int n,i,j;
		int a[]=new int[n];
		for(i=0;i<(n-1);i++){
		for(j=0;j<n-i-1;j++)}}
			69	4

Total:			191	9

B2	B2.1	import java.util.Scanner;
		class S{
		public static void main(String[]args){
		Scanner b=new Scanner(System.in);
		int n=b.nextInt();}}
			116	5

	B2.2	int num,i,j,tem;
		if(ar[j]>ar[j+1]){
		tem=ar[j];
		ar[j]=ar[j+1];}
			58	4

Total:			174	9

A1	A1.1	InputStream is=null;
		Try{
		is=new FileInputStream(f);
		byte c[]=new byte[2*1024];}
			73	4

	A1.2	Node<T>n=new Node<T>();
		n.setValue(i);
		n.setNext(f);
		if(f!=null)f.setPrev(n);
		if(f==null)r=n;
			88	5

Total:			161	9

A2	A2.1	try{
		Files.createFile(f);}
		catch(FileAlreadyExistsException x){
		System.err.format("n",f);}
			86	4

	A2.2	Node<T>d=new Node<T>();
		d.setValue(i);
		d.setPrev(r);
		if(r!=null)r.setNext(d);
		if(r==null)f=nd;
			89	5

Total:			175	9

Table 4.1 Stimulus versions (B1, B2, A1, A2)

4.2.4.1 *Stimuli difficulty factors: basic (B) & advanced (A)*

The experimental stimuli can be summarised briefly as follows:

1- B1.1 & B2.1: consist of import statement; declaring the class; declaring the main method; declaring an object from class Scanner; input value from the user. These concepts are considered fundamental in Java and students study them at an early stage in their first year. Thus, these stimuli are considered basic.

2- B1.2 & B2.2: consist of declaring variables and arrays; using conditional if; nested (inserting one loop inside the body of another loop) *for* loop. These are presented for first year students, hence these stimuli are considered basic. However, the complexity of the loop structures can make them harder.

3- A1.1 & A2.1: these stimuli include the try and catch block (*try* allows one to test some lines of code for errors while it is being executed, whereas catch allows one to execute a specific code if an error occurs in try), I/O (input/output), and stream (to read from a file or write to a file) notions. These notions are introduced later in the first term of the first year, thus students had not practiced them a lot. Hence, these stimuli are classified as advanced.

4- A1.2 & A2.2: the data structure notion is presented here, which is considered an advanced concept in Java. This notion is presented to students towards the end of their first year, thus it can be said that they are somewhat familiar with it. So, it is classified as advanced.

4.2.5 Procedures

Participants started the experiment by transcribing the practice items, then the PPCS items, and finally the Java stimuli.

Participants were asked to obey the following guidance (similar to the previous experiment's procedures but briefly listed here for ease of reading): (1) hold the pen in their preferred hand, (2)

start writing from the beginning of the line even if there is indentation, (3) start writing as soon as the stimulus is revealed, (4) copy the code as quickly and as accurately as they can, (5) continue writing even if they make a mistake and don't go back to correct it, (6) draw an upside-down triangle in place of spaces, (7) start each trial with hash (#), (8) hold down the special key to reveal the stimulus, with their preferred hand. Participants quickly became fluent in the practice trials, so these specific ways of writing were not an especial burden. Similar trial requirements were successfully used in previous experiments, so they do not undermine the reliability of the results. The participants finished the experiment within an hour.

Spaces are important in Java, and to make sure that participants produced something that indicates a space, they were asked to draw upside-down triangle. As the response sheet contains boxes, there is a temptation that they might miss them out. I did not want spaces to be missed out because it is important to record a time for every character (including blanks between characters). On the other hand, in the PPCS test spaces are included only to show the different numbers of characters and participants are not required to copy the spaces; thus, participants were asked to transcribe the letters and numbers without spaces.

For both PPCS and Java tests, participants started writing each stimulus by drawing # in order to ensure the validity of the pause for the first character. Furthermore, to ensure that participants could not write while the stimulus was apparent in the HS process, they were instructed to use their preferred hand (i.e., their writing hand) to press the special button, so they had to leave their hand off the space bar in order to continue transcribing the stimulus.

4.3 Results and discussion

The experiment's results consist of two parts: part 1 presents the results using the basic and advanced stimuli difficulty factor (similar to the first experiment's analysis), while part 2 examines the same stimuli but utilizing the complexity factor (simple/complex) for both basic and advanced categories in turn, in order to determine whether the complexity of the stimuli may be an additional factor that needs to be taken into account in the design of stimuli.

4.3.1 Results part 1

The results will consider the following: (1) the correlations between the various independent competence measures and justifying the dedicated competence measure (familiarity) for this experiment. (2) The evidence for the role of chunking in a Java transcription task across basic and advanced stimuli (difficulty factor). (3) Hypotheses H1–H5 concerning the predictions regarding the relationship of the behavioural measures (characters per view, writing-times, view-times, and pauses) to the competence measure (familiarity). (4) Hypothesis H6 relating to an improvement in behavioural measure values following normalisation for the PPCS test. (5) And finally, hypothesis H7 relating to the prediction of the relation of writing-times to characters per view.

4.3.1.1 *Independent measures of competence*

Four independent measures of competence were examined (via online questionnaire), similar to what was implemented in the first experiment (Chapter 3); education level, general programming, Java experience, and familiarity (participants' familiarity with the experiment stimuli was assessed by asking them to judge what their degree of familiarity would have been with each stimulus prior to beginning the experiment). However, the questionnaire here is slightly different. Three questions were eliminated from general programming part: the first question was deleted because it does not

give a clear indication of the general programming background, while the other two questions were deleted because there are other equivalent questions.

Education level was recorded on a scale from 1 to 4 (1 = new first-year undergraduates who had just started the course (thus have no Java background), 2 = first-year undergraduates toward the end of the year and second-year undergraduates, 3 = PhD students, 4 = faculty members). Education level has a positive (not strong) relationship with Java experience, which is in contrast to what was found in the first experiment (i.e., weak relation). The existence of the relation between education level and Java experience in this experiment may be because the majority of the recruited participants are new first-year undergraduates.

The general programming questions focus on programming concepts in general, and contain four yes/no questions. The Java experience questions concentrate only on the Java programming language, and include nine yes/no questions. For both parts, the answers were scored by giving one point for each 'Yes' answer. So, the general programming part had a scale from one to four and the Java experience part had a scale from one to eight. Familiarity scores were rated from 0 (low) to 4 (high), so with the four stimuli, the overall scale runs from one to nine.

(r)	Education Level	General Programming	Java	Familiarity
Education Level				
General Programming	0.277			
Java	0.407	-0.166		
Familiarity	0.331	-0.187	0.873	

Table 4.2 Correlation between competence measures (n=51, Pearson correlation, 1 tail, critical value is 0.354 at p<.01, 0.273 at p<.05)

Table 4.2 above presents correlations between all combinations of the measures. General programming experience is only weakly (and not significantly) correlated to the other measures. The correlation between Java experience and familiarity is essentially strong, similar to the first experiment, which suggests that both Java experience and familiarity are specific to the Java, rather than general programming competence. Hence, either Java or familiarity are suitable to serve as an independent measure. Just the analyses with the comparison using familiarity will be stated in this chapter. Familiarity scores (from 1 to 9) were utilized in order to split the participants into two groups, low competence and high competence. A binary split of participants conveniently creates two groups: 31 low-competence participants having scores from 1 to 4 and 20 high-competence participants with scores of 5 to 9. The majority of the new students are in the low-competence group, whereas most of the participants with Java experience (students at the end of their first year, in their second and last year, and PhD students) are in the high-competence group.

4.3.1.2 *The evidence for the role of chunking in the Java transcription task*

The four dependent behavioural measures were calculated from the logs of each participant. This subsection investigates whether chunking is a significant mechanism (as shown in the first experiment) in the transcription of the Java code generated by this experiment. Is there a disparity in the patterns of the behavioural measures between basic and advanced stimuli across all the participants?

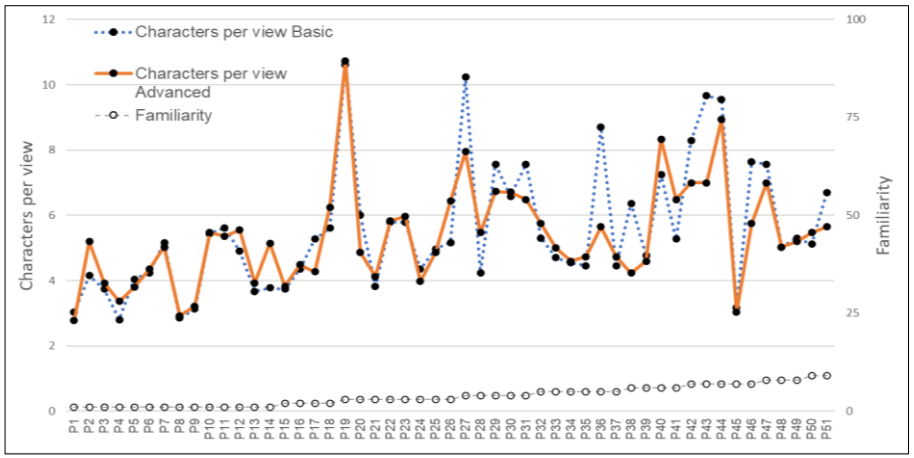

Figure 4.1 Total characters per view for participants across basic and advanced stimuli; participants are ranked by their familiarity scores

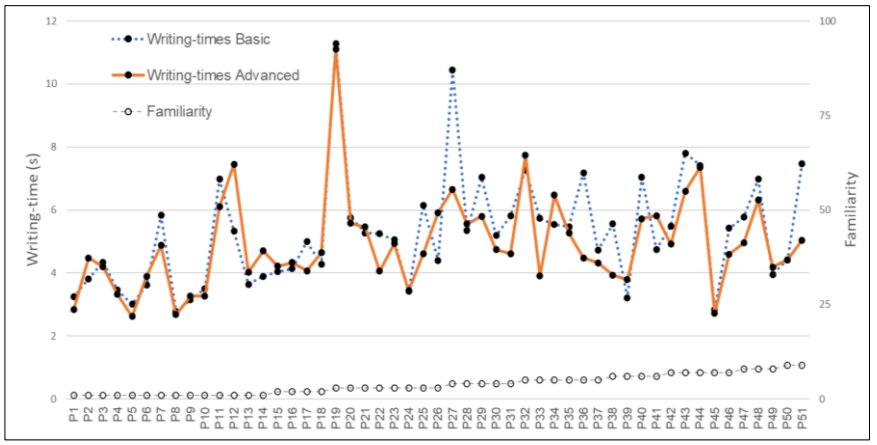

Figure 4.2 Median writing-times for participants across basic and advanced stimuli; participants are ranked by their familiarity scores

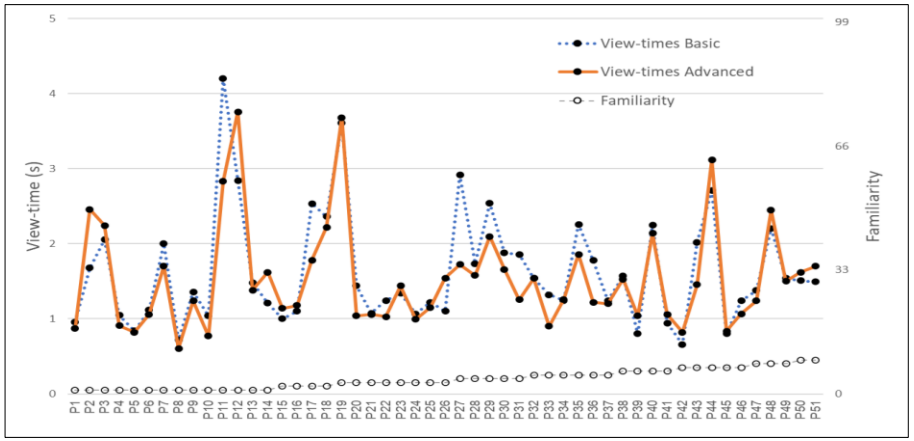

Figure 4.3 Median view-times for participants across basic and advanced stimuli; participants are ranked by their familiarity scores

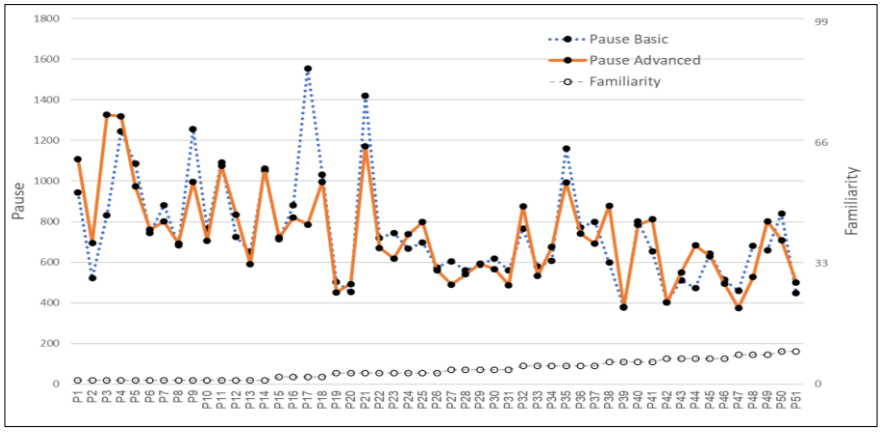

Figure 4.4 Q3 pauses for participants across basic and advanced stimuli; participants are ranked by their familiarity scores

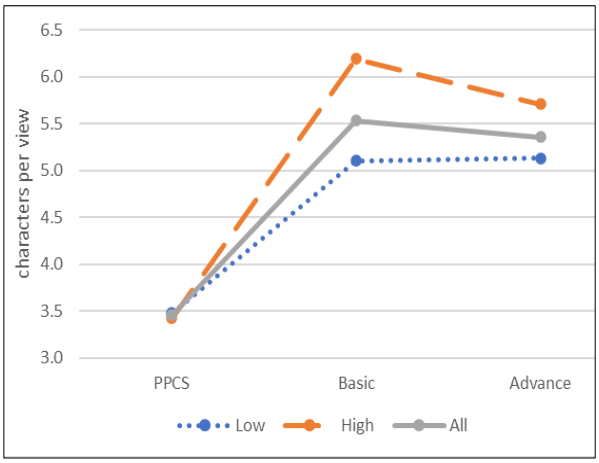

Figure 4.5 Mean characters per view across stimuli types and competence levels

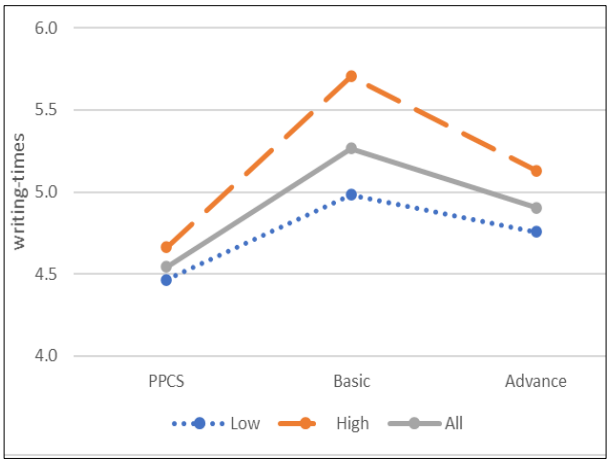

Figure 4.6 Mean of median writing-times across stimuli types and competence levels

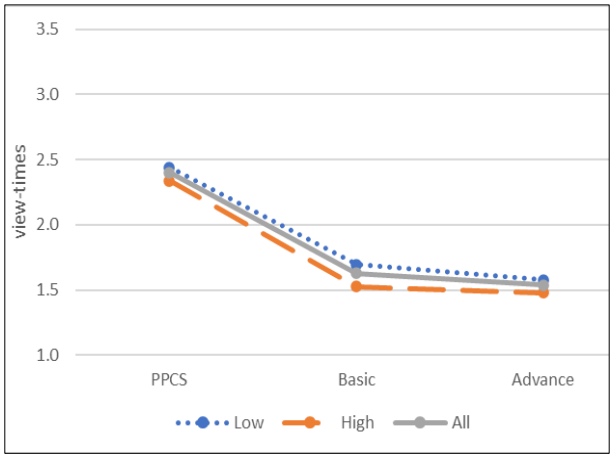

Figure 4.7 Mean of median view-times across stimuli types and competence levels

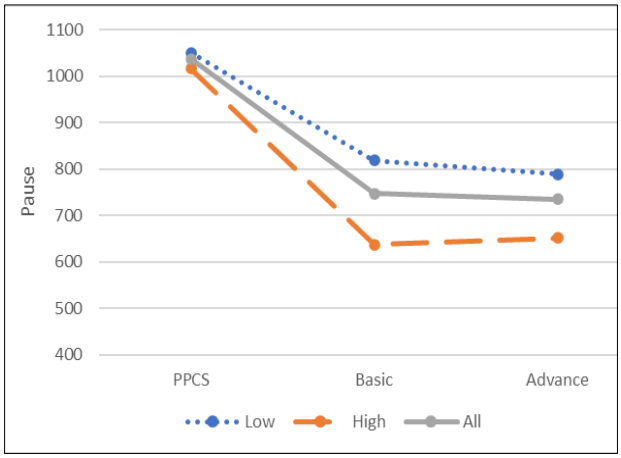

Figure 4.8 Mean of Q3 pauses across stimuli types and competence levels

Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4 show the distribution of the behavioural measures (characters per view, writing-times, view-times, and pauses) for all participants – ranked in line with their familiarity – across basic and advanced stimuli difficulty levels. Overall, there is a general trend in the above listed figures that characters per view and writing-times increase with an increase in the participants' competence levels, whereas pauses decrease, as expected from the hypotheses. In other words, the more competent participants are, the higher characters per view and writing-times they have, and the shorter pauses they produce. View-times do not show a clear overall downward or upward trend for either level of stimuli difficulty. In other words, H1, H2, H3, and H4 are all supported.

Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8 show the mean values for all the behavioural measures across the low and high groups (competence levels), PPCS test, and basic (b) and advanced (a) stimuli (difficulty levels). Generally, Figure 4.5, Figure 4.6, and Figure 4.8, suggest high-competence participants are performing better (more characters per view (SD b=1.87, a=1.41), longer writing-times (SD b=1.45, a=1.28), and shorter length of pauses (SD b=186.25, a=177.57)) than the low-competence participants (characters per view (SD b=1.88, a=1.62) and writing-times (SD b=1.94, a=1.65) and Q3 pauses (SD b=278.50, a=244.69)). However, in terms of the view-times (Figure 4.7), both low- and high-competence participants performed similarly. Mean PPCS values are essentially equal for both low- and high-competence groups for every behavioural measure shown in the figures above. It is worth noting that the mean PPCS values of characters per view and writing-times are lower than the Java stimuli values, whereas they are longer than the Java stimuli for view-times and pauses.

The following paragraphs explain Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8 in greater detail in regards to prediction H5, basic and advanced stimuli (difficulty levels):

- For characters per view and writing-times, consistent with prediction H5, Figure 4.5 and Figure 4.6 show that across all participants, basic stimuli have slightly more characters per view and longer writing-times than the advanced stimuli (i.e., not much difference between basic and advanced), which is only significant for the characters per view basic stimuli (6.2 vs 5.1, t=-2.09, p=.024, df=49, 1 tail). For both characters per view and writing-times, the disparity in the results between basic and advanced stimuli is clearer in the high-competence group than the low-competence group. As a result, despite the small group size, only characters per view is significant for high-competence participants (i.e., when compared with the low-competence group) (6.2 vs 5.7, t=-1.84, p=.041; df=18, 1 tail), and writing-times as well (5.7 vs 5.1s, t=-2.5, p=.012; df=18, 1 tail).
- For view-times, concerning prediction H5, Figure 4.7 shows the view-times mean values are similar for both basic and advanced stimuli across low- and high-competence groups. For all participants, no significant differences occur for the basic stimuli (1.7 vs 1.5, t=-0.81, p=.2, df=49, 1 tail) nor the advanced stimuli (1.6 vs 1.5, t=-0.49, p=.32, 1 tail).
- For pauses, Figure 4.8 reveals that the advanced stimuli had a longer mean pause than the basic stimuli, which is consistent with H5 (advanced: 652 vs basic: 638) for high-competence participants only but not significant (by a t test; t=-0.54, p=.29, df=49, 1 tail). The opposite is the case (advanced: 789 vs basic: 819) for the low-competence participants (which contradicts H5), but also not significant (by a t test; t=-0.86, p=.2, df=49, 1 tail).

As previously noted, there is no discernible variation in the participants' output between basic and advanced stimuli across characters per view, writing-times, and pauses for this experiment. In comparison, the previous experiment showed substantial variation in participants' temporal chunk signals between basic and advanced stimuli. As a consequence of these observations, I conclude that there is something about the stimuli that merits repeating the analysis, but this time based on

the complexity factor of the stimuli (simple and complex) rather than the difficulty factor (basic and advanced).

In summary, Figure 4.5, Figure 4.7, and Figure 4.8 show that PPCS test findings suggest that the effect of transcribing the Java stimuli exists beyond the act of merely transcribing any stimuli. Moreover, the overall data provides support for all the predictions H1, H2, H3, H4, and H5 (H5: except for the low-competence participants pauses), all of which suggest that chunking is a key mechanism in the transcription task in VD and HS settings, which is consistent with the results of the first experiment (except for pauses, as they were not calculated in the first experiment).

The next subsection and its subsections examine the predictions of the relationship of characters per view, writing-times, view-times, and pauses to familiarity.

4.3.1.3 *Regression analysis for the behavioural measures against familiarity*

The seven hypotheses that relate temporal chunk signals to competence are considered in this subsection. A simple linear regression was performed to analyse the relationship between the different variables in greater depth. The results are presented for each behavioural measure against familiarity, to provide additional information about the strength of the relationship between a behavioural measure and familiarity, and to provide information about the reliability of the behavioural measure. The results are also presented for writing-times, view-times, and Q3 pauses against characters per view (in the next subsection) to provide extra evidence that chunking is occurring. The analysis was performed on PPCS, basic, and advanced stimuli for all participants and separately for low- and high-competence participants, to normalize the behavioural measures for the PPCS measure, thus improving the behavioural measure values.

Since the regression analysis was performed independently for low- and high-competence participants, this subsection is separate from the behavioural measurements subsection (4.3.1.2, all

participants together). Generally, predictions H1, H2, H3, H4, H5, and H7 are supported; the orientation of the relationship is as expected, the absolute magnitudes of the gradients are significant, and the data points are a good match to the linear regression line.

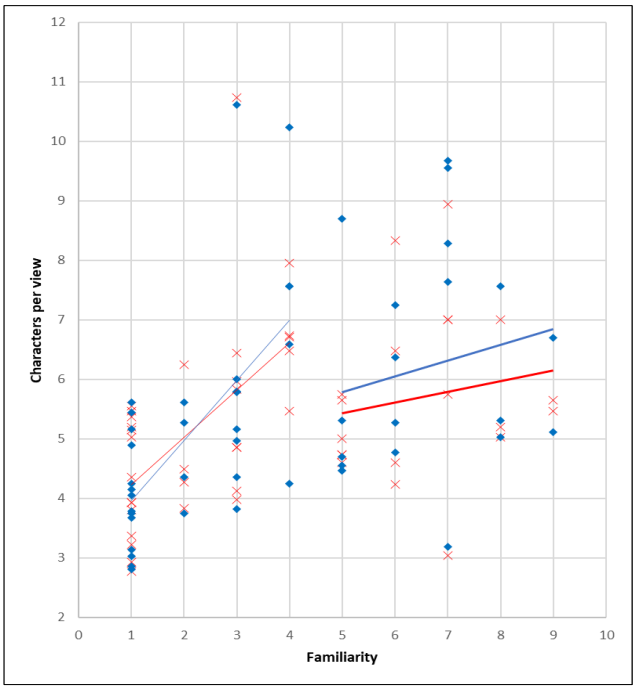

Figure 4.9 Relation of characters per view to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli

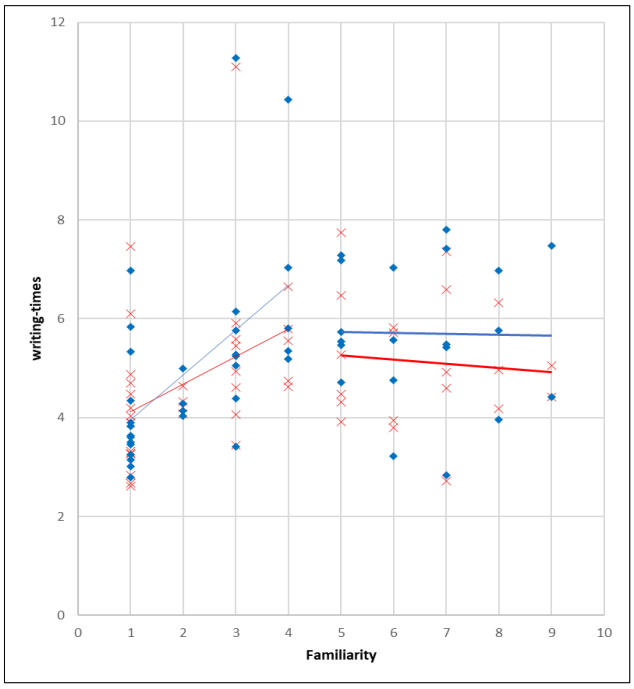

Figure 4.10 Relation of writing-times to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli

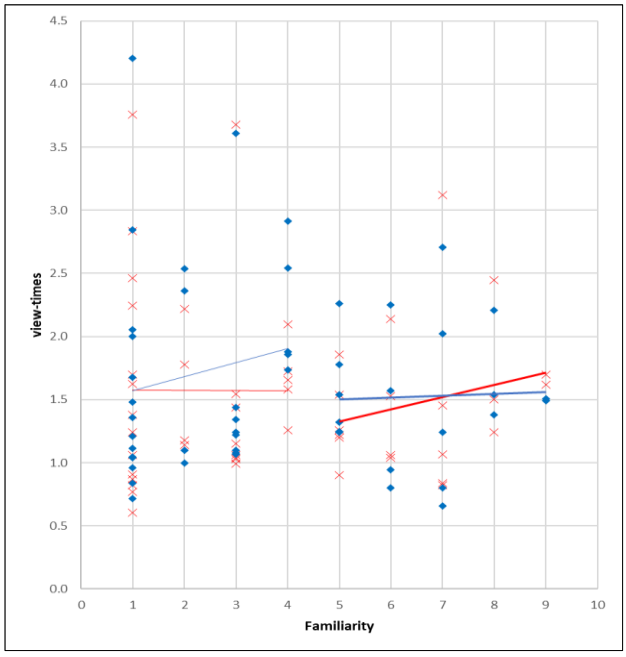

Figure 4.11 Relation of view-times to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli

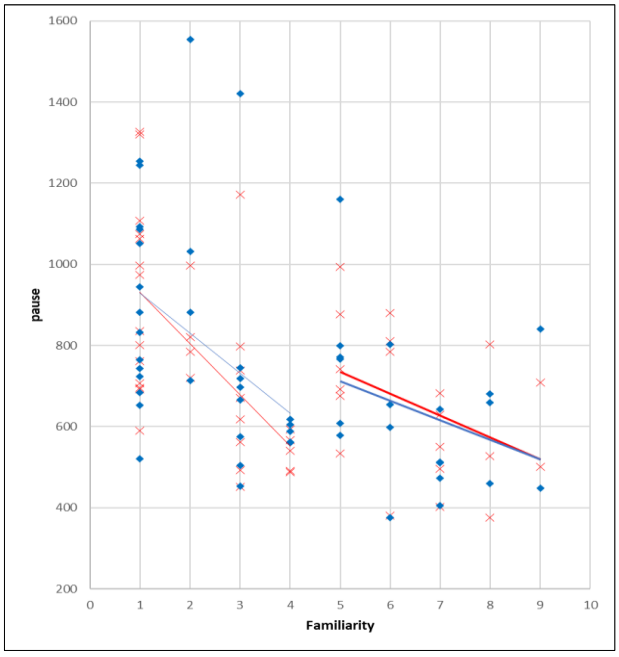

Figure 4.12 Relation of pauses to familiarity for all participants; low- (thin lines) and high- (thick lines) competence groups. Basic (blue diamonds) and advanced (red crosses) stimuli

Figure 4.9, Figure 4.10, Figure 4.11, and Figure 4.12 present the linear regression relationships of characters per view, writing-times, view-times, and $pause_{Q3}$ to familiarity for participants, who are ranked in order of familiarity. They also present the relationships among all participants as well as low- and high-competence groups, across basic and advanced stimuli, separately. The lines for both low and high groups are nearly flat for view-times (which do not differentiate participants' competence levels), as opposed to the steep lines for the other measures, which fits the first experiment's findings and is compatible with H1, H2, H3, and H4. For characters per view, writing-times, and $pause_{Q3}$, the lines are steeper for the low group than for the high group, indicating that the behavioural measures are better at discriminating the low group than the high group. In other words, the stimuli perform better for the low group than for the high group. Accordingly, the R^2 values for the behavioural measures, shown in Table 4.3 below, are slightly greater for characters per view and Q3 pauses than for writing-times, whereas they are very weak – close to zero – for view-times, as predicted. Thus, characters per view and Q3 pauses are the more reliable measures in predicting Java competence. R^2 values are lower (weaker relation) for the high group than for the low group. Thus, the stimuli do not work well for high-competence participants. Another thing to note is that, for both characters per view and writing-times, the distinction between the basic and advanced stimuli is clearer for the high-competence group than for the low-competence group. The critical R^2 values for characters per view, writing-times, view-times, and $pause_{Q3}$ are as follows: for all participants, the critical value is 0.23 for significant R^2 at p<.05, and 0.322 at p<.01 (1 tail, df=49). The critical value for R^2 with only high-competence participants is 0.296 at p<.05 and 0.409 at p<.01 (1 tail, df=29). The critical value for R^2 with only low-competence participants is 0.378 at p<.05 and 0.516 at p<.01 (1 tail, df=18).

stimuli name		PPCS	Low PPCS	High PPCS	B	A	Low B	High B	Low A	High A	B-PPCS	A-PPCS
N		*51*	31	20	51	51	31	20	31	20	51	51
df		49	29	18	49	49	29	18	29	18	49	49
char/VN	Constant	-0.03	-0.12	-0.06	0.35	0.22	1.01	0.26	0.79	0.18	0.49	0.25
	Intercept	3.58	3.74	3.81	4.18	4.49	2.95	4.46	3.44	4.53	1.62	1.12
	f	0.60	0.80	0.54	12.66	7.35	19.33	0.69	14.41	0.55	12.43	5.70
	significance f	0.441	0.378	0.471	0.001	0.009	0.000	0.418	0.001	0.468	0.001	0.021
	$R^{2\cdot}$	0.01	0.03	0.03	0.21	0.13	0.40	0.04	0.33	0.03	0.20	0.10
WT MDN	Constant	0.07	0.27	-0.03	0.24	0.13	0.91	-0.02	0.56	-0.09	0.05	0.13
	Intercept	4.28	3.88	4.86	4.33	4.41	3.05	5.81	3.57	5.71	0.17	0.06
	f	1.06	3.57	0.02	6.39	2.29	12.69	0.00	5.44	0.16	3.92	0.68
	significance f	0.309	0.069	0.904	0.015	0.137	0.001	0.948	0.027	0.696	0.053	0.415
	$R^{2\cdot}$	0.02	0.11	0.00	0.12	0.05	0.30	0.00	0.16	0.01	0.07	0.01
VT MDN	Constant	-0.03	-0.01	-0.14	-0.01	-0.01	0.11	0.01	0.00	0.10	-0.85	-0.97
	Intercept	2.53	2.46	3.23	1.68	1.56	1.46	1.43	1.58	0.84	0.02	0.03
	f	0.30	0.00	0.86	0.09	0.09	0.72	0.02	0.00	1.00	0.18	0.42
	significance f	0.588	0.964	0.367	0.762	0.762	0.404	0.877	0.999	0.330	0.675	0.522
	$R^{2\cdot}$	0.01	0.00	0.05	0.00	0.00	0.02	0.00	0.00	0.05	0.00	0.01
Pause Q3	Constant	-24.49	-104.60	-95.70	-49.40	-46.07	-98.84	-48.23	-126.28	-53.88	-228.03	-251.28
	Intercept	1111.35	1239.50	1643.51	938.46	913.03	1029.06	953.50	1057.54	1004.89	-17.22	-14.87
	f	1.48	3.32	3.46	14.34	16.71	6.12	2.53	16.92	3.68	1.10	0.95
	significance f	0.229	0.079	0.079	0.000	0.000	0.026	0.129	0.000	0.071	0.299	0.335
	$R^{2\cdot}$	0.03	0.10	0.16	0.23	0.25	0.17	0.12	0.37	0.17	0.02	0.02

Table 4.3 Parameter of best-fit linear relation for characters per view, writing-times, view-times, and pauses to familiarity

I will look at each behavioural measure individually in the subsections that follow: characters per view, writing-times, view-times, and Q3 pauses.

4.3.1.3.1 Characters per view

Hypothesis H1 focuses on characters per view, the total number of stimuli characters divided by the total number of times the stimulus is viewed per trial, which should increase with an increase in familiarity. Table 4.3 shows linear regression models of characters per view as functions of familiarity scores for all participants with both stimuli difficulty levels at low and high stages of Java competence.

As predicted and as shown in Figure 4.9 and Table 4.3, characters per view increases with familiarity scores (steep lines – good constant values) for all the participants for both basic and advanced stimuli (the more competent participants are with Java, the more characters per view they have); thus, characters per view effectively distinguishes participants' Java competence levels. When splitting the participants into low- and high- competence groups, the low-

competence group has a more stable relation than the high group. Put another way, the low group has a steeper line – greater constant – than the high group. Thus, in general, characters per view can strongly predict Java competence for the low competence group only.

The correlations for all the participants, whether with basic or advanced stimuli, are unlikely to be due to chance. The correlations for low B and low A groups are not due to chance. The R^2 values are lower than the first experiment, but they are significant for low B and low A only. The goodness of fit value for basic is greater than the value for comparable advanced stimuli. Partitioning the participants into low and high groups over basic and advanced stimuli results in better fitted values and higher R^2 values for the low group than for the high group. As a consequence, characters per view are thought to be a good predictor of Java competence in transcription tasks. In conclusion, there is support for prediction H1 and consistency with the previous experiment (Chapter 3).

4.3.1.3.2 Writing-times

Hypothesis H2 focuses on writing-times following each stimulus view. Writing-times is the time spent writing between two consecutive views in HS presentation condition. Table 4.3 reveals linear regression models of writing-times to familiarity scores for all participants with all stimulus difficulty levels at variable levels of Java competence (low and high).

As expected, and presented in Figure 4.10 and Table 4.3, writing-times improve with familiarity scores (steep lines – good constant values); the more competent participants are with Java, the longer their writing-times. So, writing-times differentiate participant competence levels (the basic stimuli are more suitable for the participant levels), but not as well as characters per view. Thus, there is a positive significant relation between familiarity and writing-times. For both basic and advanced stimuli, the low group, in comparison, have a more robust association (steeper line – a higher constant) than the high group. Thus, in general, writing-times will forecast Java

competence strongly (though not as strongly as characters per view; see section 4.3.1.3.1), and predict it more strongly for the low group than for the high group.

The correlations in the case of basic stimuli only (for all participants) are unlikely to be due to chance. The low-competence group correlations for both basic and advanced are unlikely to be random as well. The R^2 values are low; they are significant only for Low B. The goodness of fit value for basic stimuli is higher than that for corresponding advanced stimuli. When participants are split into low and high groups, the low group has much better suited values and higher R^2 values than the high group, for both basic and advanced. Hence, writing-times have some potentials to predict Java competence in transcription tasks. To summarise, there is some support for prediction H2, which is consistent with the first experiment.

4.3.1.3.3 View-times

Hypothesis H3 focuses on view-times and predicts that there is no relation between view-times and Java competence. View-times is the duration of each view of the stimulus in the HS presentation condition. View-times differ from characters per view, writing-times, and Q3 pauses as chunking theory suggests that there is no association between view-times and competence. Table 4.3 reveals linear regression models of view-times as functions of familiarity scores for all the participants with both basic and advanced stimuli at varying levels of Java competence (low and high).

In addition, as expected and as presented in Figure 4.11 and Table 4.3, the lines do not show a clear upward or downward trend (flat lines – poor constant values) in regards to familiarity scores, thus do not discriminate participants' levels of competence well for participants as a whole (basic and advanced) and for both low and high subgroups, which is counter to Figure 4.9, Figure 4.10, and Figure 4.12 (characters per view, writing-times, Q3 pauses, respectively). Hence, generally, view-times cannot be a predictor of Java competence in a transcription activity.

The correlations are all likely to be due to chance. R^2 values are all close to zero (very weak). The scatter of the data points around the regression lines is very high, as opposed to the scatter of the data points around the regression lines of other behavioural measures. So, view-times is thought to be an unreliable predictor of Java competence in a transcription task. To put it briefly, there is support for prediction H3 and uniformity with the first experiment (Chapter 3).

4.3.1.3.4 $Pause_{Q3}$

Hypothesis H4 focuses on pauses, the duration of the pause before transcribing a stroke. Table 4.3 shows linear regression models of pauses as functions of familiarity scores for all participants with both stimuli difficulty stages at low and high levels of Java competence.

As predicted and presented in Figure 4.12 and Table 4.3, pauses decrease with the increase of familiarity scores (inverse relation) – steep lines, good constant values, for both basic and advanced stimuli (the more competent participants are with Java, the shorter the pauses they produce). Hence, Q3 pauses efficiently distinguish participants' Java competence levels. The low-competence group has a more stable association than the high group. To put it another way, the low group has a steeper line – a higher constant – than the high group. Thus, in general, Q3 pauses will strongly predict Java competence, and predict it more strongly for the low group than for the high group.

The correlations for basic and advanced (all participants) and for low B and low A subgroups only are unlikely to be accidental. The R^2 values are strongly significant for all the participants with both levels of stimuli difficulty, and remain so when just the low-competence group is considered. Whereas the R^2 values for the high-competence group are likely to be due to chance. The data fits very well for both basic and advanced (all participants). Splitting the participants into low and high competence over basic and advanced stimuli causes better fitted values and higher R^2 values for the low group than for the high group, as with characters per view (Figure 4.9) and writing-times (Figure 4.10). For the low-competence group, it is noted that

the goodness of fit value for the advanced stimuli is greater than for comparable basic stimuli (the opposite of characters per view and writing-times); this may be because there is an exception in the details of the design of the stimuli (or perhaps the VD measure is more accurate because it picks every single stroke in ms, while HS measures may consist of a set of characters, letters, or words). Thus, the Q3 pauses measure is considered to be a better indicator of Java skill in transcription tasks. To sum up, there is support for prediction H4.

The overall results for characters per view, writing-times, view-times, and $pause_{Q3}$ are consistent with the predictions, but with some divergence in details. Characters per view and Q3 pauses are recognized as more reliable predictors of Java competence, but in most cases, Q3 pauses are a slightly better predictor of Java competence than characters per view.

4.3.1.4 *PPCS normalization*

Hypothesis H6 concerns the improvement of the behavioural measures values after normalizing them for the PPCS (Participants' Preferred Cluster Size) test. It is worth recalling that the PPCS stimuli are made up of randomly ordered numbers and letters (an alphanumeric string) and are delivered in four forms: S1(used), S2, S3, and S4, as explained earlier in the chapter.

Table 4.3 shows linear regression models of PPCS as functions of familiarity scores for all the participants and both low- and high-competence groups. As shown in the columns PPCS (all participants), Low PPCS (low-competence group), and High PPCS (high-competence group), there is generally no relation between PPCS results and familiarity scores. For all the behavioural measures, the constant values are very low, and the R^2 values are close to zero. Thus, the PPCS test does not distinguish participants' Java familiarity levels, and hence cannot predict Java competence. All the correlations are likely to be due to chance.

Table 4.3 displays linear regression models of normalised PPCS values for both stimulus difficulty levels (basic (B-PPCS) and advanced (A-PPCS)) as functions of familiarity scores. For characters per view, when comparing the columns B&B-PPCS and A&A-PPCS, the constant values are

slightly greater after normalizing, whereas the R^2 values are slightly lower (almost the same) for both B&B-PPCS and A&A-PPCS. All the correlations are unlikely to be due to chance. Thus, the B-PPCS and A-PPCS measures distinguished participants' Java levels and can predict Java competence, but not as well as the B and A. Normalising for the PPCS test does not improve characters per view values to make them a better measure of Java competence. On the other hand, for writing-times, view-times, and Q3 pauses, the constant and R^2 values are very low after normalizing when compared with the B and A. All the correlations are due to chance. Therefore, the normalized data (B-PPCS and A-PPCS measures) does not differentiate between participants' Java levels and cannot predict Java competence, hence normalising for the PPCS test does not improve writing-times, view-times, or Q3 pauses values to make them a stronger indicator of Java competence.

To sum up, contrary to hypothesis H6, none of the behavioural measures were improved after normalizing, as shown in Table 4.3, B-PPCS and A-BBCS columns. Put another way, individual differences do not impact the effectiveness of the behavioural measures because the values of all behavioural measures do not increase after subtracting each participant's working memory size.

In the following subsection, additional proof of the role of chunking in transcription tasks will be provided via the regression analysis for writing-times, view-times, and Q3 pauses against characters per view. Since characters per view, writing-times, and Q3 pauses are predicted to be influenced by chunking processes, there should be a clear and structured interaction between them. View-times are not supposed to be chunk based, so there would be no expected association between view-times and characters per view.

4.3.1.5 *Regression analysis for writing-times, view-times, and pauses against characters per view*

Prediction H7 concerns the nature of the direct connection between writing-times and characters per view (the more characters per view participants generate, the longer their writing-time while transcribing).

Extra evidence for the role of chunking in freehand transcribing can be gathered by looking at the relation between characters per view and the other measures. The previous experiment (Chapter 3) showed that writing-times were significantly predicted by characters per view. Thus, for this experiment, as all the behavioural measures use chunking, a positive relation between characters per view and writing-times is expected, as both are anticipated to increase with an increase in Java familiarity scores (H1 and H2). A negative relation between characters per view and $pause_{Q3}$ is predicted because pauses are supposed to decrease with an increase in familiarity (H5). For view-times, it is expected that there will be a weaker relation than with writing-times, as view-times are supposed to be not directly related to programming competence (H3).

Scatter plots for writing-times, view-times, and $pause_{Q3}$ against characters per view are plotted below (Figure 4.13, Figure 4.14, and Figure 4.15) for all participants and low- and high-competence groups for both basic and advanced stimuli.

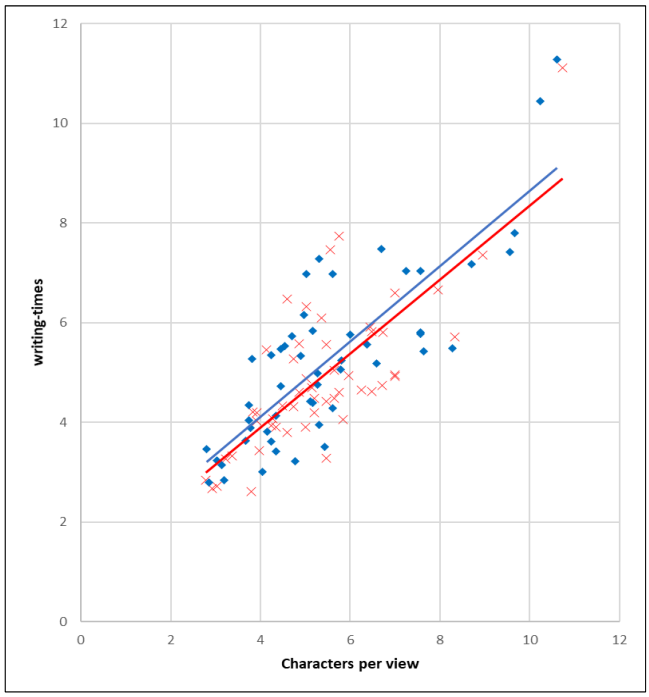

Figure 4.13 Relation of writing-times to characters per view (basic (blue diamonds) and advanced (red crosses) stimuli)

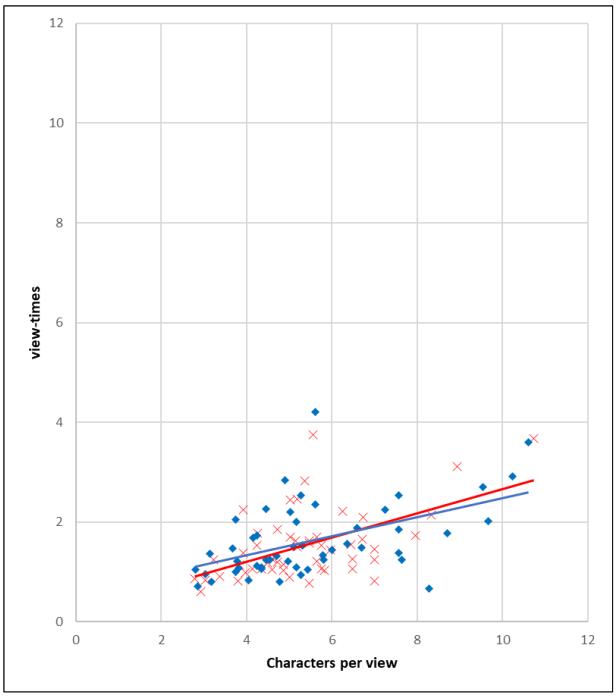

Figure 4.14 Relation of view-times to characters per view (basic (blue diamonds) and advanced (red crosses) stimuli)

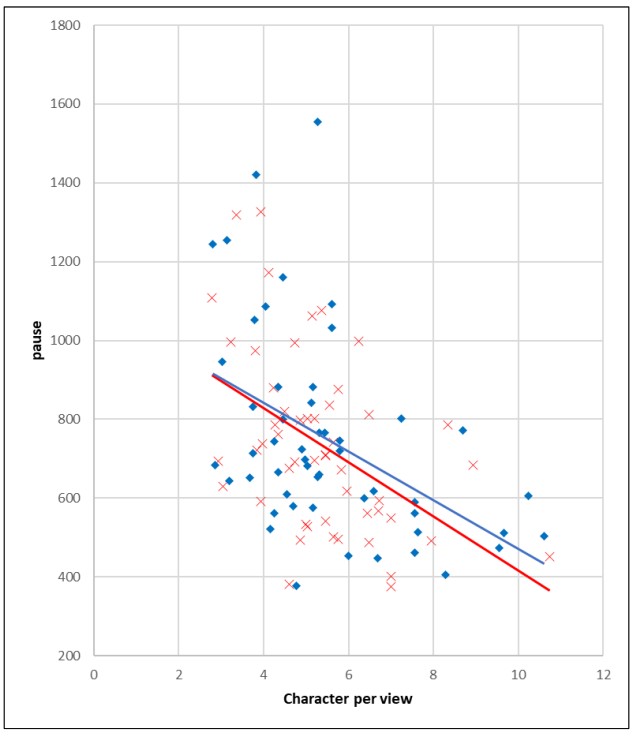

Figure 4.15 Relation of pauses to characters per view (basic (blue diamonds) and advanced (red crosses) stimuli)

stimuli name		PPCS	B	A
N		*51*	51	51
df		49	49	49
WT MDN	Constant	-0.06	0.76	0.74
	Intercept	4.75	1.09	0.93
	f	0.07	99.70	67.56
	significance f	0.789	0.000	0.000
	$R^{2\cdot}$	0.00	0.67	0.58
VT MDN	Constant	-0.01	0.19	0.24
	Intercept	2.43	0.57	0.24
	f	0.00	16.50	20.01
	significance f	0.971	0.000	0.000
	$R^{2\cdot}$	0.00	0.25	0.29
Pause Q3	Constant	-22.54	-61.95	-68.70
	Intercept	1094.71	1090.29	1103.17
	f	0.11	13.14	13.59
	significance f	0.747	0.001	0.001
	$R^{2\cdot}$	0.00	0.21	0.22

Table 4.4 Parameter of best-fit linear relation for writing-times, view-times, and Q3 pauses to characters per view

Figure 4.13, Figure 4.14, and Figure 4.15 above show the linear regression relation of writing-times, view-times, and $pause_{Q3}$ in relation to characters per view for test-takers across stimuli difficulty levels (basic and advanced). A simple linear regression was used to assess whether writing-times, view-times, and $pause_{Q3}$ can predict characters per view.

Hypothesis H7 concerns the relation between characters per view and writing-times. As predicted, writing-times increase with characters per view (the longer participants spent transcribing after each view, the more characters per view they have). Table 4.4 shows linear regressions of writing-times, view-times, and $pause_{Q3}$ as functions of characters per view for both basic and advanced stimuli. In Figure 4.13, Figure 4.14, and Figure 4.15, the lines for both basic and advanced stimuli are steeper (larger constant values) for writing-times than view-times, similar to the first experiment results (Chapter 3) and compatible with H7. All the correlations are unlikely to be due to chance. Further, the R^2 values are greater – stronger relation – for writing-times than for both view-times and pauses. Thus, writing-times are thought

to be a reliable predictor of characters per view in transcription tasks. For writing-times only, the goodness of fit value for basic is superior than the value for the equivalent advanced stimuli. For PPCS, compared with the Java stimuli, the constant and the R^2 values for PPCS are very low (close to zero). The correlations are all due to chance.

Another point to note is that there is no substantial variation in participant results between basic and advanced across any of the figures above (only stimuli difficulty factor is concerned), which encourages the detailed analysis in part 2 (concerning the complexity factor).

In summary, writing-times have remarkably the better constant and R^2 values than view-times and $pause_{Q3}$ (explanations are given below) in relation to characters per view, as shown in Table 4.4. Writing-times has the best fitted regression line to the data (stronger relation), which supports H7 and matches the first experiment (Chapter 3). In other words, writing-times is the more reliable measure for predicting the number of characters per view.

The subsection that follows goes into the findings for the main results (part 1) in more depth, via the seven predictions.

4.3.2 Discussion part 1

This subsection discusses the overall results in two subsections, the first concerning the seven hypotheses which were discussed in the introduction subsection of this chapter. The second subsection compares this experiment with the previous experiment (Chapter 3) in order to answer the specific question addressed in the design of the stimuli for this experiment

4.3.2.1 *Are the behavioural measures performing as expected?*

The previous experiment (Chapter 3) showed that view-numbers and writing-times, obtained in the HS presentation mode, can be used to assess Java programming competence in a simple transcription task. Previous studies have revealed that measures of the distribution of inter-stoke pauses, in the same kind of task (copying, transcription), appear to reflect participants'

chunk structures and thus can be utilized to examine learners' competence (Cheng, 2014, 2015; van Genuchten & Cheng, 2010; Zulkifli, 2013). This experiment extends those findings and differs from the first experiment (Chapter 3) in that: (1) $pause_{Q3}$ is investigated as a temporal chunk signal measure that is available in the view display VD presentation mode, (2) a PPCS test is applied for each participant in both VD and HS presentation modes.

The linear regression analysis showed that the overall relationships of characters per view, writing-times, and $pause_{Q3}$ with the stimuli familiarity scores are strong, especially for low-competence participants group, as revealed in Table 4.3. To put it another way, characters per view, writing-times, and $pause_{Q3}$ have potential as predictors of Java competence, and they predict it more strongly for the low-competence group than for the high-competence group.

Hypothesis H1 predicts a positive relationship between characters per view and familiarity, and hypothesis H2 likewise predicts a positive relationship between writing-times and familiarity. Hypothesis H3 predicts that there is no relationship between view-times and familiarity. The findings are consistent with these predictions and extend the findings of the first experiment (Chapter 3), which showed that participants' temporal chunk signals (characters per view and writing-times) are greatly influenced by participants' Java competence levels (chunk structures). On the other hand, in this experiment, there is little support for view-times as an adequate predictor of competence. This means that any chunking effects that can arise during stimulus viewing would be minimised. Hence, characters per view and writing-times have potentials as indicators of Java competence in transcription tasks.

Hypothesis H4 predicts a negative relationship between $pause_{Q3}$ and familiarity. $Pause_{Q3}$ is greatly shaped by the participant's Java comprehension level, hence the participant's chunk structure. The simple linear regression test, as seen in Table 4.3, shows that $pause_{Q3}$ effectively distinguishes the Java competence of participants. Thus, $pause_{Q3}$ is thought to be a good predictor of Java ability in transcription tasks, which allow me to answer 'Yes' to the question:

Is $pause_{Q3}$ a potential measure for assessing programming comprehension in a transcription task?

Hypothesis H5, which states that participants do better on basic stimuli than on advanced stimuli (difficulty factor), is supported for characters per view and writing-times (which is consistent with the previous experiment) and partly supported for $pause_{Q3}$. This is as expected, since basic stimuli are meant to be easy, and most participants should be able to transcribe them very well. Whereas, for $pause_{Q3}$, the length of the pauses is similar for both levels of stimuli difficulty (basic and advanced) among all participants and both low and high subgroups. This may be because VD and HS modes are two distinct ways of displaying the stimulus. In VD, any character (i.e., stroke) participates as a data point as $pause_{Q3}$ picks up any single bit of punctuation, thus this measure is finer grained (smaller scale). In HS, characters per view and writing-times are a smaller number of data points and a set of characters.

Generally, for all the behavioural measures, the difference in participants' performance between basic and advanced is not as clear as it was in the previous experiment. When taking a close look at the stimuli, and with the benefit of hindsight, the results are not surprising, because the second part of both basic stimuli for this experiment is similar to the advanced stimuli used in the first experiment (Chapter 3). For the HS measures, the disparity in results between basic and advanced stimuli is more pronounced in the high-competence group than in the low-competence group; this is reasonable considering that high-competence participants considered the basic stimuli simpler than the advanced stimuli (even the basic stimuli have a complex part), while low-competence participants found them both (basic and advanced) difficult.

Hypothesis H6, which concerns the possibility of improving (stronger associations with Java familiarity scores) characters per view, writing-times, view-times, and Q3 pause values following normalisation for the PPCS (Participants Preferred Cluster Size) test (Table 4.3 columns B-PPCS and A-PPCS), was not supported. So, the answer is 'No' to the second question of this chapter:

Can characters per view, writing-times, view-times, and Q3 pauses be improved by normalizing them using PPCS? It is worth noting that, also in the first experiment (Chapter 3), I attempted to normalise using practise items, which did not work as well (perhaps because there were only slight differences between participant performances).

Figure 4.5, Figure 4.6, and Figure 4.8 show that for each behavioural measure, the PPCS test has almost the same values, which significantly differ from the Java stimuli values (similar to the relationship between the practise items and the Java stimuli in the first experiment). This adds to the evidence of the Java stimuli's role on measuring competence in transcription tasks. Further, the absence of the PPCS effect may be due to PPCS being unimportant in the transcription task, and/or that the individual variations observed in the PPCS in terms of characters per view, writing-times, and Q3 pauses might not be as significant as the differences observed while actually transcribing the Java stimuli.

Lastly, hypothesis H7 is supported, which concerns the existence of a direct substantial relationship between writing-times and characters per view: the longer participants spent transcribing a chunk, the more characters per view they created, which corresponds to what was discovered in the first experiment (Chapter 3). The clear linear relationship between writing-times and characters per view (Figure 4.13, Table 4.4) underpins the chunk explanations supporting my first and second predictions, H1 and H2. However, view-times (i.e., based on chunking but not related to competence) and characters per view have a weaker relationship, as expected, that not as characters per view do with writing-times (Figure 4.14, Table 4.4), which provides additional support to the third prediction, H3. While transcribing, the participant looks at the stimulus, then does some writing, then returns to look at the stimulus, and has to find where he/she finished the last line and then start looking for the new part. So, something else is interfering with the process, and this could indicate that the relationship between characters per view and view-time is weak (the length of time a person spends viewing the stimulus is

unrelated to the number of characters they transcribe). Further, the linear regression analysis clearly indicates a negative association between $pause_{Q3}$ and characters per view (Figure 4.15, Table 4.4), and supports chunking theories that underpin my fourth prediction, H4. This relationship is weak, as expected, since the Q3 pauses are clearly determined by the participant's chunks and not by the number of characters per view.

To sum up, this experiment is consistent with the first experiment (Chapter 3) in terms of the potential of HS measures as Java competence predictors and how these measures differentiate participants' Java competence levels (more competent participants have more characters per view and longer writing-times) across stimuli difficulty levels (basic and advanced). This experiment suggests that characters per view and $pause_{Q3}$ have potentials as predictors of Java competence in a transcription task, but in most cases $pause_{Q3}$ becomes a better predictor of Java competence than characters per view. This may be because the nature of the transcription task is more suited to the view display (VD) presentation mode; also, as previous research has shown, pauses are a greater reflection of competence and are more closely related to the structure of chunks (Cheng, 2014, 2015; van Genuchten & Cheng, 2010; Zulkifli, 2013), hence, there is a possibility that such a slight difference occurs between the values of $pause_{Q3}$ and the values of characters per view and writing-times. Furthermore, in comparison to the first experiment and consistent with H3, no evidence of the unusual relation between view-times and programming competence was found here. In addition, the disparity in participant performance between basic and advanced (difficulty factors) is not as substantial as in the first experiment. So, one of the things that really seems to be an important factor when designing the stimuli is the stimuli complexity factor (explained in the next subsections). Consequently, the analysis of basic (simple and complex) and advanced (simple and complex) variations will be applied in the detailed analysis provided in the following subsection (Results part 2).

4.3.2.2 *This experiment's stimuli design*

As mentioned earlier in the stimuli deign subsection, two main factors were focused on while designing this experiment stimuli: (1) consulting the Java module content for the student participants; (2) consulting the content analysis results (Chapter 3), which concern increasing the number of chunks via providing more expressions with symbols and punctuation, capital letters with a variable, and spaces. So, has trying to increase the density of these expressions made this experiment's stimuli (part 1) more discriminating between those with high competence? The answer is 'No'; the explanations are as follows.

This experiment supports the previous experiment's findings (Chapter 3) in terms of: (1) the substantial variation in temporal chunk signal HS metrics (characters per view and writing-times) as a result of participants' familiarity. (2) The curves in Figure 4.1, Figure 4.2, and Figure 4.4 propose that characters per view, writing-times, and $pause_{Q3}$ have plateaued for the high-competence participants, which suggests that advanced stimuli will not be adequate to differentiate the high-competence group. This may be because some participants had more Java knowledge, and the stimuli were structured based on the undergraduate Java curriculum. This curve, once again, resembles that of comparable research (Cheng, 2014, 2015; Albehaijan & Cheng 2019). Thus, careful selection of the level of difficulty in the test stimuli will be important for the design of tests using this method. As a consequence, this experiment's redesigned stimuli again did not distinguish participants with high competence levels. So, another question to ask is: 'What other factors may be important in the design of the stimuli that impact its difficulty?'

In the following subsection, Results part 2, the stimuli are viewed according to the complexity factor (simple and complex) rather than the difficulty factor (basic and advanced, as in Results part 1 above), for two main reasons: (1) Why is there no difference in participants' performance between basic and advanced stimuli? Especially for $pause_{Q3}$ which contradicts prediction H5. (2) The other reason, stemming from reason 1, is to examine whether considering the difficulty

factor alone is sufficient when designing the experiment's stimuli, or whether the complexity factor should also be considered.

4.3.3 Results part 2

For the second part of the analysis, I predict that the complexity of the stimulus could be an additional aspect that must be considered in stimuli design. Hence, in this section, I will re-examine the existing experimental stimuli by using the complexity factor (simple and complex). I will consider the following: (1) analysing the experiment stimuli based on the complexity factor by measuring the percentage of different stimuli components; (2) the discrimination between the two complexity levels (simple and complex) across all behavioural measures.

4.3.3.1 *Stimuli complexity analysis*

Consulting the Java module content was considered while designing both the previous experiment's (Chapter 3) and this experiment's stimuli. However, this experiment's stimuli differ from the previous experiment in that more effort was made to increase the density of the syntax (i.e., more punctuation), more variables including capital letters in the middle of words, more spaces, and shortened variable names; this was done for all the stimuli difficulty levels (i.e., all versions). This was done based on the findings of the content analysis (Chapter 3), which showed that, in general, participants choose to see the stimulus (i.e., took breaks) more often before transcribing the following categories: punctuation marks, capitals within variables or reserved words, and spaces.

According to this complexity analysis (part 2), the more *PunctuationALL* in a stimulus, the more complex it is. Therefore, the stimuli were classified as simple or complex (based on the stimulus complexity factor (syntax)) for both the basic and advanced categories. I compared *PunctuationALL* (i.e., summation of brackets and punctuation marks) and *textAll* (summation of reserved words and variable names) in order to determine which stimulus is simple and which is complex.

Table 4.5 shows the percentage of the number of characters of each stimuli version (B1.1, B1.2, B2.1, B2.2, A1.1, A1.2, A2.1, A2.2) according to the content measures. The content measures consist of two main groups, *PunctuationAll* and *textAll*. In more detail, *PunctuationAll* concerns the summation of the two percentages *brackets* (total number of brackets e.g. {[<) and *symbols* (total number of symbols e.g., + - *!) for each stimulus version, as in equation 1 below. Whereas *textAll* concerns the summation of the two proportions *ReservedWords* (total number of characters in each Java reserved word e.g., main, try, class, for) and *VariableNames* (total number of characters in each variable name e.g., tem, ar, n, s) for each stimulus version, as in equation 2 below.

$$PunctuationAll\ proportion = brackets\ prportion + symbols\ proportion \quad (1)$$

$$textAll\ proportion = ReservedWords\ proportion + VariableNames\ proportion \quad (2)$$

Concerning the degree of complexity, for the basic stimuli (B1, B2), it is obvious from Table 4.5 below that for the simple stimuli (B1.1, B2.1), the *PunctuationALL* percentage is much lower than the percentage for *textAll*, whereas for the complex stimuli (B1.2, B2.2), the *PunctuationALL* proportion is slightly lower than the *textAll*. In terms of the advanced stimuli (A1, A2), for the simple stimuli (A1.1, A2.1), the *PunctuationALL* proportion is much smaller than the *textAll*, while for the complex stimuli (A1.2, A2.2), the *PunctuationALL* proportion is lower than the *textAll* but not as much as it is in the simple. In brief, the degree of complexity between the two levels of basic (B1.1+B2.1 vs B1.2+B2.2) is much stronger than the degree of complexity between the two levels of advanced (A1.1+A2.1 vs A1.2+A2.2). Put another way, the distinction in the participants' performance between simple and complex will be clearer for the basic stimuli (B1, B2) than for the advanced stimuli (A1, A2). Thus, the basic category will better differentiate simple and complex than the advanced category.

Stimuli Versions		Stimulus Content	No. of characters	Brackets ()<>{}[]""	. ; = + * > < - !	PunctuationALL	Reserved words	Variable names	TextALL	PunctuationALL+TextALL

B1 (Basic)	B1.1 (Simple)	`import java.util.Scanner;`
		`class A{`
		`public static void main(String[]args){`
		`Scanner s=new Scanner(System.in);`
		`double l=s.nextDouble();}}`
			122	0.10	0.07	0.17	0.80	0.03	0.83	1

	B1.2 (Complex)	`int n,i,j;`
		`int a[]=new int[n];`
		`for(i=0;i<(n-1);i++){`
		`for(j=0;j<n-i-1;j++)}}`
			69	0.19	0.29	0.48	0.26	0.26	0.52	1

B2 (Basic)	B2.1 (Simple)	`import java.util.Scanner;`
		`class S{`
		`public static void main(String[]args){`
		`Scanner b=new Scanner(System.in);`
		`int n=b.nextInt();}}`
			116	0.10	0.08	0.18	0.78	0.03	0.82	1

	B2.2 (Complex)	`int num,i,j,tem;`
		`if(ar[j]>ar[j+1]){`
		`tem=ar[j];`
		`ar[j]=ar[j+1];}`
			58	0.24	0.19	0.43	0.09	0.48	0.57	1

A1 (Advance)	A1.1 (Simple)	`InputStream is=null;`
		`Try{`
		`is=new FileInputStream(f);`
		`byte c[]=new byte[2*1024];}`
			73	0.11	0.10	0.21	0.64	0.15	0.79	1

	A1.2 (Complex)	`Node<T>n=new Node<T>();`
		`n.setValue(i);`
		`n.setNext(f);`
		`if(f!=null)f.setPrev(n);`
		`if(f==null)r=n;`
			88	0.18	0.16	0.34	0.52	0.14	0.66	1

A2 (Advance)	A2.1 (Simple)	`try{`
		`Files.createFile(f);}`
		`catch(FileAlreadyExistsException x){`
		`System.err.format("n",f);}`
			86	0.14	0.07	0.21	0.74	0.05	0.79	1

	A2.2 (Complex)	`Node<T>d=new Node<T>();`
		`d.setValue(i);`
		`d.setPrev(r);`
		`if(r!=null)r.setNext(d);`
		`if(r==null)f=nd;`
			89	0.18	0.16	0.34	0.51	0.16	0.66	1

Table 4.5 Proportions of the no. of characters for stimuli content categories

4.3.3.2 *Discrimination between the two stimuli complexity levels across all behavioural measures*

The structure of the following subsections will be slightly different from the structure of Results part 1, behavioural measure correlations subsection, as both basic and advanced have been broken down into simple and complex. For ease of reading and comparison between basic (simple and complex) and advanced (simple and complex) complexity categories, I present each behavioural measure (characters per view, writing-times, view-times, and pauses) separately.

The focus in this subsection is mainly on the discrimination between the two complexity levels (simple and complex).

The next four subsections demonstrate the relations of characters per view, writing-times, view-times, and Q3 pauses with familiarity.

4.3.3.2.1 Characters per view

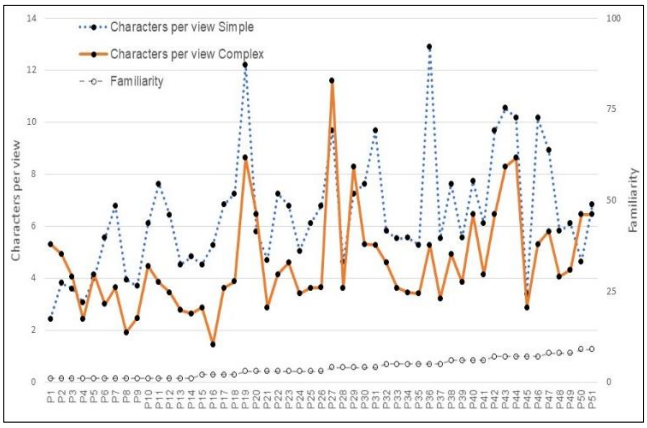

Figure 4.16 (Basic) Total characters per view for participants across simple and complex stimuli; participants are ranked by their familiarity scores

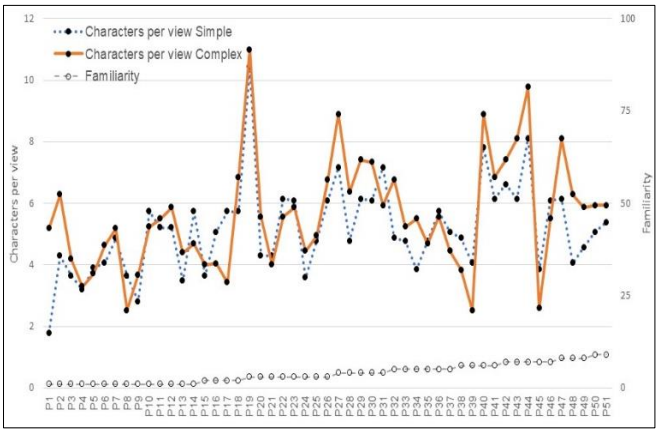

Figure 4.17 (Advanced) Total characters per view for participants across simple and complex stimuli; participants are ranked by their familiarity scores

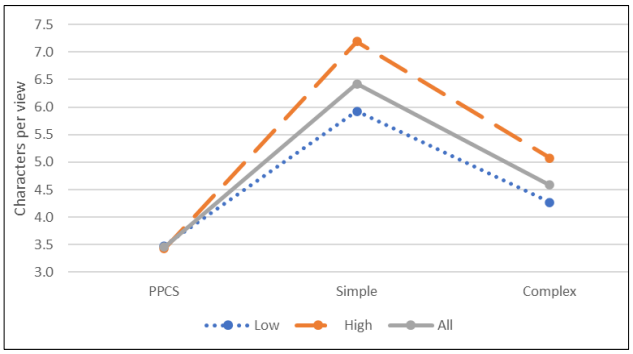

Figure 4.18 (Basic) Mean characters per view across stimuli types and competence levels

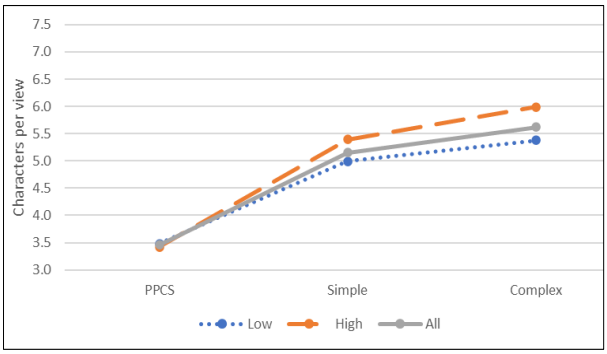

Figure 4.19 (Advanced) Mean characters per view across stimuli types and competence levels

Figure 4.16 and Figure 4.17 above show the distribution of characters per view for all participants – ranked according to their familiarity scores – across simple and complex stimuli. Figure 4.18 and Figure 4.19 present characters per view mean values across the two complexity levels and low- and high-competence groups, for basic and advanced, respectively.

Charters per view	Main analysis part1		Detailed Analysis part2			
Competence levels	Basic	Advanced	Basic		Advanced	
			(simple)	(complex)	(simple)	(complex)
All	5.5	5.4	6.4	4.6	5.2	5.6
High	6.2	5.7	7.2	5.1	5.4	6.0
Low	5.1	5.1	5.9	4.3	5.0	5.4

Table 4.6 below shows the difference in the mean values for the number of characters per view between the main analysis part 1 (difficulty factor) and the detailed analysis part 2 (complexity factor) over all the participants and across competence levels (low and high). Based on Table 4.5 and in regards to Table 4.6, the following comparisons can be made:

(1) The difference between simple and complex stimuli in the detailed analysis should be greater (and more apparent for basic than advanced) than the difference between basic and advanced in the main analysis. Figure 4.16 to Figure 4.19 show that splitting the stimuli into two levels of complexity (simple and complex) increases the difference in the total number of characters per view between these two complexity levels when compared with the main analysis (part 1). It is clear that the basic category has the greatest improvement in the characters per view mean values, as would be expected (from the stimuli complexity analysis in Table 4.5 above).

(2) The mean number of characters per view should be higher for the high- than the low-competence group. Figure 4.16 to Figure 4.17 and Table 4.6 show that there is generally a positive relationship between the number of characters per view and Java familiarity scores (matching H1). This relationship is significant for basic (simple) stimuli (high 7.2 vs low 5.9, t=-2.02, p=.028, df=49, 1 tail) and not significant for basic (complex) stimuli (high 5.1 vs low 4.3, t=-1.49, p=.075, df=49, 1 tail). For the Advanced category, the relationship is not significant for both simple (high 5.4 vs low 5.9, t=-0.96, p=.173, df=49, 1 tail) and complex (high 6.0 vs low 5.4, t=-1.20, p=.122, df=49, 1 tail).

(3) The mean number of characters per view should be greater for simple than complex stimuli (matching H5). Overall, Figure 4.16 to Figure 4.17 and Table 4.6 show that the basic category is consistent with prediction H5, across all participants; simple stimuli have more characters per view than complex ones (participants performed better on simple than complex stimuli). The performance on simple stimuli is superior to performance on complex stimuli and this is significant for all participants (6.4 vs 4.6 respectively, t=-7.23, p=.000, df=49, 1 tail), and for low- and high-competence groups as well (low: t=-4.73, p=.000, df=29, 1 tail, high: t=4.42, p=.000, df=18, 1 tail). These findings are significantly stronger than the main analysis findings (part 1), and consistent with complexity analysis expectations (Table 4.5 above).

The advanced category contradicts prediction H5. Figure 4.19 shows that all participants performed slightly better on complex stimuli than simple ones; this was expected as, based on the stimuli complexity analysis, the percentage of *PunctuationALL* is close for both simple and complex. The distinction in the performance between simple and complex is significant for all participants (5.2 vs 5.6 respectively, t=-2.94, p=.002, df=49, 1 tail) and almost significant for low- and high-competence groups as well (low: t=-1.88, p=.033, df=29, 1 tail, high: t=-2.17, p=.017, df=18, 1 tail).

Concerning the PPCS test (Figure 4.18 and Figure 4.19), the findings are similar to the main analysis (part 1), which suggests that the effect of transcribing the Java stimuli exists beyond the act of merely transcribing any stimuli.

Charters per view	Main analysis part1		Detailed Analysis part2			
Competence levels	Basic	Advanced	Basic		Advanced	
			(simple)	(complex)	(simple)	(complex)
All	5.5	5.4	6.4	4.6	5.2	5.6
High	6.2	5.7	7.2	5.1	5.4	6.0
Low	5.1	5.1	5.9	4.3	5.0	5.4

Table 4.6 Mean total number of characters per view across low- and high-competence groups and across stimuli difficulty and complexity factor for both (analysis parts 1 & part 2)

4.3.3.2.2 Writing-times

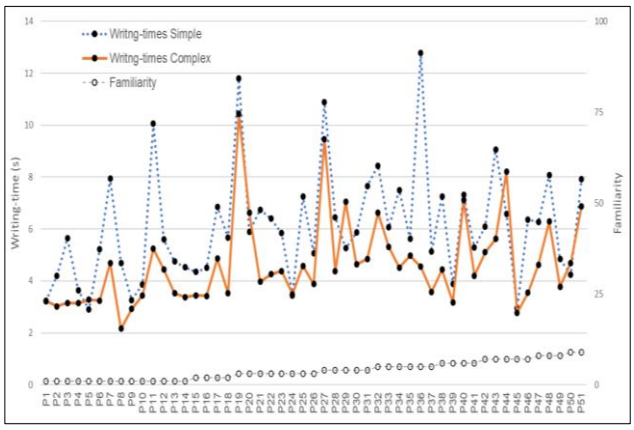

Figure 4.20 (Basic) Median writing-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores

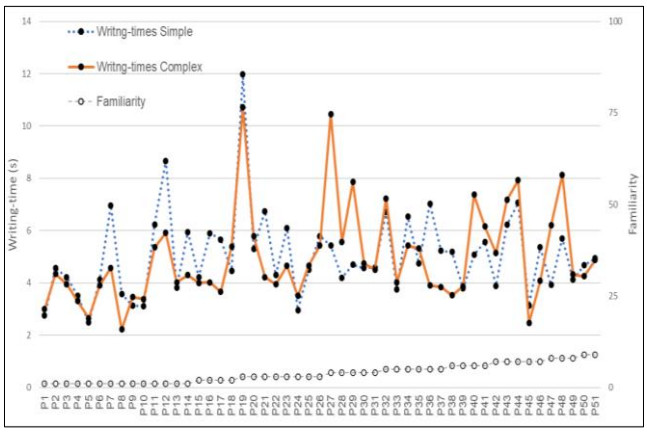

Figure 4.21 (Advanced) Median writing-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores

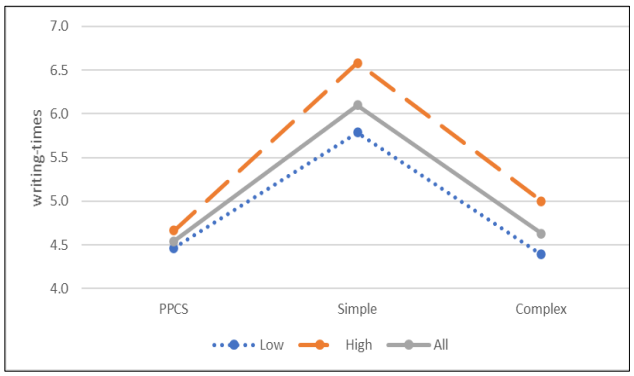

Figure 4.22 (Basic) Mean of median writing-times across stimuli types and competence levels

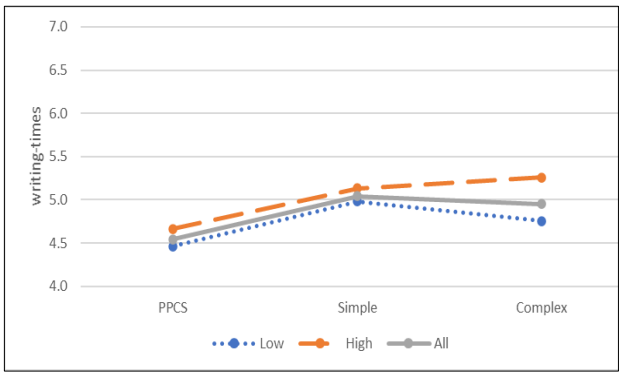

Figure 4.23 (Advanced) Mean of median writing-times across stimuli types and competence levels

Figure 4.20 and Figure 4.21 above show the distribution of writing-times for all participants, ranked in familiarity order, over simple and complex stimuli. Figure 4.22 and Figure 4.23 show mean writing-times values across simple and complex and low- and high-competence groups, for basic and advanced. Table 4.7 below presents the mean writing-times for both part 1 (difficulty factor) and part 2 (complexity factor) over all the participants and low- and high-competence groups. Based on Table 4.5 and in regards to Table 4.7, the following comparisons can be made:

(1) The disparity between the simple and complex analyses (part 2) should be larger (and more noticeable for basic than advanced) than the difference between the basic and advanced analyses (part 1). Figure 4.20 to 4.23 show that, compared to the analysis in part 1, dividing the stimulus into two degrees of complexity increases the disparity in the mean writing-times for these two complexity levels. Similar to the characters per view, the basic (simple, complex) category has the greatest improvement in the mean values of writing-times, as anticipated (from the stimuli complexity analysis, Table 4.5 above). On the whole, the difference between simple and complex is apparent for basic (Figure 4.22 and Figure 4.23) but not for advanced (Figure 4.23), as predicted.

(2) The mean writing-times should be longer for the high-competence group than for the low. The overall pattern (Figure 4.20 to 4.23 above and Table 4.7 below) for both basic and advanced shows a positive relationship between familiarity and writing-times (conforming to H2) but this is not significant for both basic (simple: high 6.6 vs low 5.8, t=-1.31, p=.102, df=49, 1 tail; complex: high 5.0 vs low 4.4, t=-1.29, p=.105, df=49, 1 tail) and advanced (simple: high 5.1 vs low 5.0, t=-0.31, p=.378, df=49, 1 tail; complex: high 5.3 vs low 4.8, t=-0.98, p=.168, df=49, 1 tail).

(3) The mean writing-time could be longer for simple stimuli than for complex stimuli (matching H5). Overall, Figure 4.20 to 4.23 and Table 4.7 reveal that the basic category (Figure 4.22) is aligned with prediction H5, over all participants, and simple stimuli have longer writing-times than complex (participants performed better on simple than on complex stimuli). Superior results on simple compared to complex stimuli were significant to all participants (6.1 vs 4.6 respectively, t=-6.66, p=.000, df=49, 1 tail) and significant for low- and high-competence participants as well (low: t=-5.49, p=.000, df=29, 1 tail; high: t=-3.27, p=.001, df=18, 1 tail). These results are better overall than part 1 of the analysis, and they are compliant with the complexity analysis expectations, as seen in Table 4.5 above. The results show that the disparity in writing-times between simple and complex is better for the basic category.

The advanced category (Figure 4.23) is marginally aligned with prediction H5 for the low-competence group only. All participants showed similar performance on both simple and complex stimuli (5.0 vs 5.0 respectively, t=-0.413, p=.341, df=49, 1 tail). The low group performed slightly better on simple than complex, however the difference is not significant (5.0 vs 4.8 respectively, t=-0.799, p=0.214, df=29, 1 tail). For the high group, writing-times for complex stimuli were slightly longer than for simple (5.1 vs 5.3 respectively, t=-0.402, p=.345, df=18, 1 tail). This is not surprising because, depending on the stimuli complexity analysis (Table 4.5), the ratio of PunctuationALL is similar for both simple and complex stimuli.

Concerning the PPCS test (Figure 4.22 and 4.23), the results are consistent with the main analysis (part 1), implying that the influence of transcribing the Java stimulus extends beyond the act of simply transcribing any stimuli.

Writing-times	Main analysis part1		Detailed Analysis part2			
Competence levels	Basic	Advanced	Basic		Advanced	
			(simple)	(complex)	(simple)	(complex)
All	5.3	4.9	6.1	4.6	5.0	5.0
High	5.7	5.1	6.6	5.0	5.1	5.3
Low	5.0	4.8	5.8	4.4	5.0	4.8

Table 4.7 Mean writing-times across low- and high-competence groups and across stimuli difficulty and complexity factor for both (analysis parts 1 & 2)

4.3.3.2.3 View-times

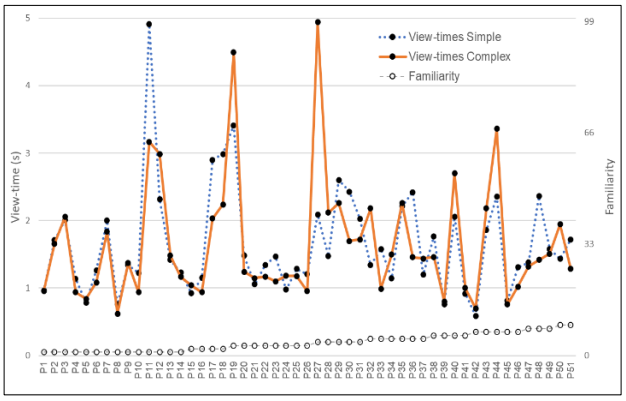

Figure 4.24 (Basic) Median view-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores

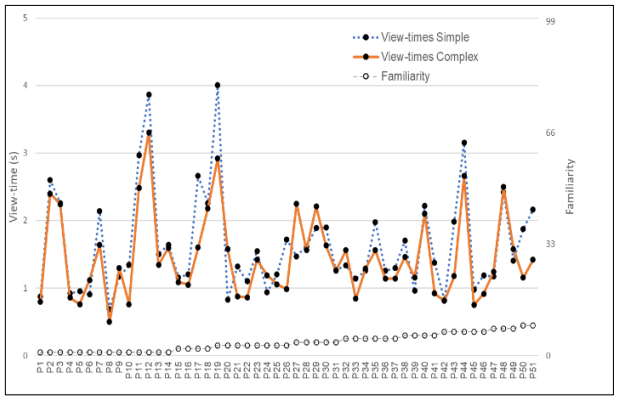

Figure 4.25 (Advanced) Median view-times for participants across simple and complex stimuli; participants are ranked by their familiarity scores

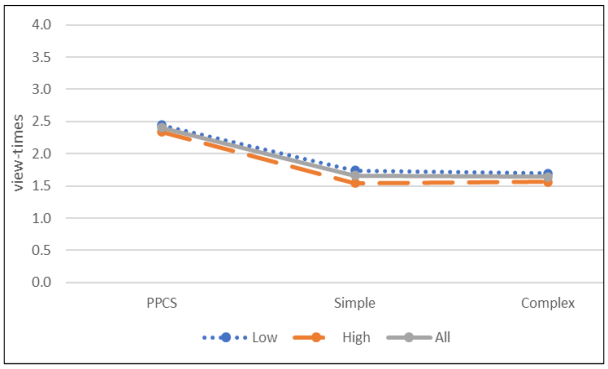

Figure 4.26 (Basic) Mean of median view-times across stimuli types and competence levels

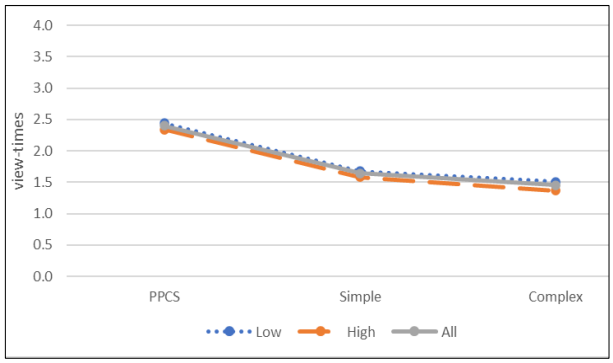

Figure 4.27 (Advanced) Mean of median view-times across stimuli types and competence levels

Figure 4.24 and Figure 4.25 above present the distribution of view-times for all participants, ranked according to their familiarity, across simple and complex stimuli. Figure 4.26 and 4.27 show mean view-times across the two complexity levels and the low- and high-competence groups, for basic and advanced, separately. Table 4.8 below shows the mean view-times for the main analysis and detailed analysis over all the participants and the two levels of competence. The following distinctions should be made based on Table 4.5 above and Table 4.8 below:

(1) The difference in the mean view-times between simple and complex in the detailed analysis should be similar to the difference between basic and advanced in the main analysis. Overall,

Figure 4.24 to 4.27 show that splitting the stimuli into two levels of complexity (simple and complex) produces the same difference in the mean view-times between these two complexity levels when compared with the main analysis (part 1).

(2) The mean view-times values should not be related to Java competence. Figure 4.24 and Figure 4.25 do not show a clear overall downward or upward trend in view-times in relation to Java familiarity scores for both basic and advanced categories, consistent with prediction H3, which concerns the absence of a general effect of view-times in measuring competence. That is in contrast to the characters per view (Figure 4.16) and writing-times (Figure 4.20) trends.

(3) The mean view-times should be similar for simple and complex stimuli, as view-times are not related to competence. Figure 4.24 to 4.27 and Table 4.8 show that the mean view-times are similar for both simple and complex stimuli across the low- and high-competence groups and across the basic and advanced categories.

In terms of the basic category, consistent with the main analysis (part 1), across all participants, there is no significant difference in the mean view-times between simple and complex (1.66 vs 1.65 respectively, t=-0.180, p=.429, df=49, 1 tail). The difference between simple and complex is not significant for both competence groups (low: t=-0.323, p=0.374, df=29, 1 tail; high: t=-0.202, p=.420, df=18, 1 tail).

Regarding the advanced category, for all participants, there is a significant difference in view-times between simple and complex stimuli (1.64 vs 1.46 respectively, t=-3.519, p=.000, df=49, 1 tail). The difference is significant between simple and complex for both competence groups (low: t=-2.183, p=0.017, df=29, 1 tail; high: t=-2.841, p=.003, df=18, 1 tail). Both competence groups viewed the simple stimuli slightly longer than the complex (this applied to the advanced category only).

View-times	Main analysis part1		Detailed Analysis part2			
Competence levels	Basic	Advanced	Basic		Advanced	
			(simple)	(complex)	(simple)	(complex)
All	1.6	1.5	1.7	1.6	1.6	1.5
High	1.5	1.5	1.5	1.6	1.6	1.4
Low	1.7	1.6	1.7	1.7	1.7	1.5

Table 4.8 Mean view-times across low- and high-competence groups and across stimuli difficulty and complexity factor for both (analysis parts 1 & 2)

4.3.3.2.4 $Pause_{Q3}$

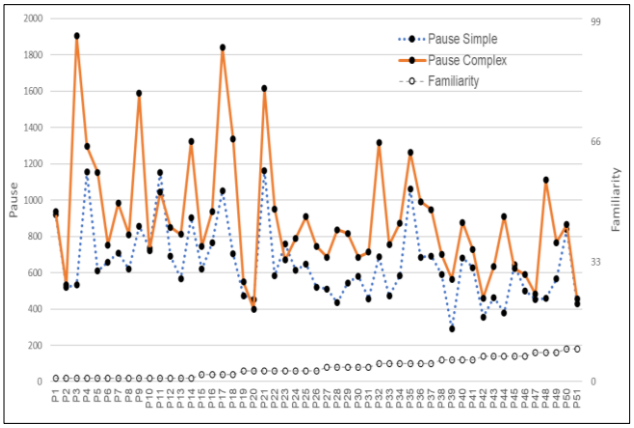

Figure 4.28 (Basic) Q3 pauses for participants across simple and complex stimuli; participants are ranked by their familiarity scores

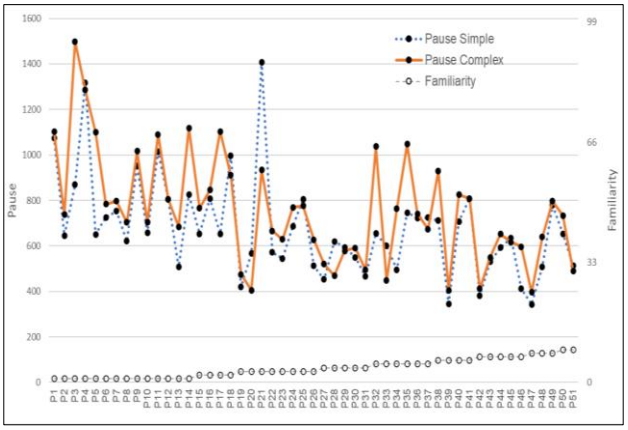

Figure 4.29 (Advanced) Q3 pauses for participants across simple and complex stimuli; participants are ranked by their familiarity scores

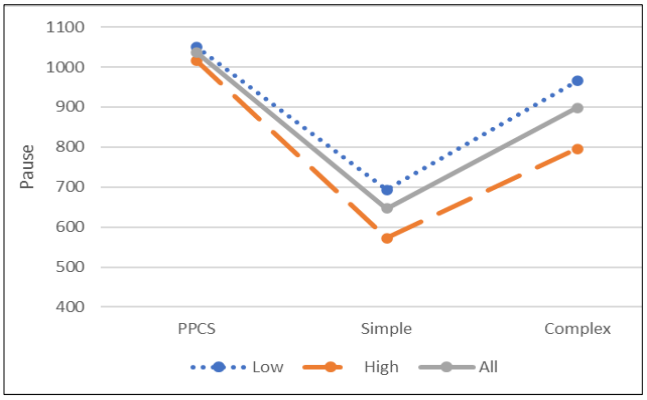

Figure 4.30 (Basic) Mean of Q3 pauses across stimuli types and competence levels

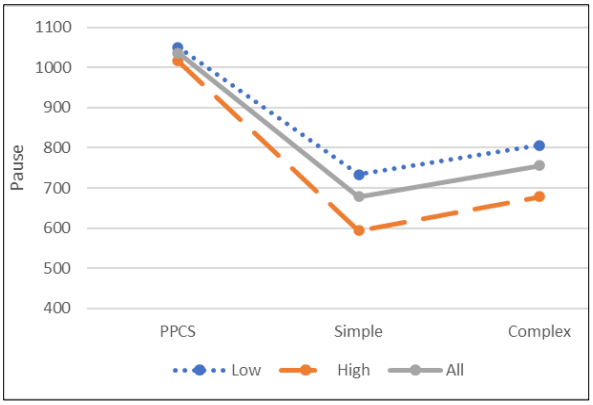

Figure 4.31 (Advanced) Mean of Q3 pauses across stimuli types and competence levels

Figure 4.28 and Figure 4.29 above show the distribution of the Q3 pauses for all participants, ranked according to their familiarity scores, across the simple and complex stimuli complexity

levels. Figure 4.30 and Figure 4.31 show Q3 pause values across simple and complex and low- and high-competence groups, for basic and advanced, respectively. Table 4.9 below shows the disparity in the mean values for the Q3 pauses between the analysis part 1 and analysis part 2 across participants' competence levels (low and high). Based on Table 4.5 above and focusing on Table 4.9 below, the following comparisons can be drawn:

(1) The distinction between simple and complex in the detailed analysis should be stronger (and more visible for basic than for advanced) than the distinction between basic and advanced in the main analysis. Figure 4.28 to Figure 4.31 demonstrate that dividing the stimuli into two degrees of complexity increases the discrepancy in the overall Q3 pauses between these two complexity levels as compared to the main analysis. The basic category (Figure 4.30) clearly has the greatest improvement in the Q3 pause values, as expected (from the stimuli complexity analysis, Table 4.5 above).

(2) The length of the Q3 pauses should be shorter for the high-competence group than the low group. Overall, Figure 4.28 and Figure 4.29 above and Table 4.9 below reveal a clear downward trend for both basic and advanced categories. Both basic and advanced categories show significant negative relations between familiarity scores and the length of pauses, for the basic category, across all participants (simple: high 573 vs low 694, t=-2.18, p=.020, df=49, 1 tail; complex: high 795 vs low 965, t=-1.84, p=.040, df=49, 1 tail) and for the advanced category, across all participants (simple: high 593 vs low 733, t=-2.45, p=.011, df=49, 1 tail; complex: high 678 vs low 806, t=-1.94, p=.033, df=49, 1 tail). Put another way, the more competent participants have shorter pauses (to a great degree, consistent with H4).

(3) The Q3 pauses should be shorter for simple than for complex stimuli (matching H5). Overall, Figure 4.28 to Figure 4.31 and Table 4.9 below demonstrate that, focusing on the basic category, consistent with prediction H5, Figure 4.30 reveals that all participants produce longer pauses while transcribing the complex stimuli than for the simple ones (i.e., participants performed

better on simple than complex stimuli), as shown in Table 4.9 below. Further, the difference between simple and complex is significant (646 vs 899, t=-6.85, p=.000, df=49, 1 tail) and significant for both low- and high-competence groups as well (low: t=-4.67, p=.000, df=29, 1 tail; high: t=-4.42, p=.000, df=18, 1 tail). These results are significantly stronger than the main analysis findings (part 1), and they are compliant with the complexity analysis assumptions, as seen in Table 4.5 above.

The advanced category conforms to prediction H5; Figure 4.31 above shows a similar trend to Figure 4.30 (basic category). All participants performed better on simple than complex stimuli and had significant negative relations between Java familiarity scores and Q3 pauses (678 vs 756, t=-3.24, p=.001, df=49, 1 tail). The difference between simple and complex is significant for low- and high-competence groups as well (low: t=-2.05, p=.023, df=29, 1 tail; high: t=-2.71, p=.005, df=18, 1 tail). In sum, these results are significantly robust when compared with the analysis (part 1).

Q3 pause	Main analysis part1		Detailed Analysis part2			
Competence levels	Basic	Advanced	Basic		Advanced	
			(simple)	(complex)	(simple)	(complex)
All	748	735	646	899	678	756
High	638	652	573	795	593	678
Low	819	789	694	965	733	806

Table 4.9 Q3 Pauses across low- and high-competence groups and across stimuli difficulty and complexity factors for both (analysis parts 1 & 2)

The overall results suggest that the second part of the analysis achieved the required outcome by showing an improvement in the disparity between the level of stimuli difficulty from the main analysis part 1 (basic and advanced) and the detailed analysis part 2 (simple and complex). In other words, when the complexity factor is considered, improvements in the difference between the stimuli difficulty levels occur for all the behavioural measures. In fact, the VD measure (Q3 pauses) is the best in regards to the improvement in differentiating levels of complexity than the HS measures (characters per view and writing-times). Characters per view

has higher disparities than writing-times, whereas view-times remain constant, as expected according to H3. In terms of the low- and high-competence groups, the high group outperformed the low group in terms of characters per view, writing-time, and Q3 pauses. Furthermore, all the behavioural measures in the basic category showed a more significant difference between simple and complex than the advanced category, which matches my predictions based on the stimuli complexity analysis (Table 4.5). Predictions H1, H2, H3, H4, and H5 are all supported for all behavioural measures, with only a few minor differences in details. PPCS mean values are almost the same for low- and high-competence participants, with an obvious difference between PPCS and Java stimuli mean values, which is similar to the results of the main analysis part 1, for all the behavioural measures.

4.3.4 Discussion part (2)

For the detailed analysis (part 2), every stimulus used in the main analysis (part 1) is further broken down (to form two separate degrees of complexity) according to the stimuli complexity analysis (Table 4.5 above). The overall findings of the detailed analysis (part 2) are addressed in this subsection (according to the comparisons made in subsection 4.3.3.2 above).

Generally, when the complexity factor is taken into account, the discrimination between the two complexity levels (simple and complex) is improved for each behavioural measure, as shown in Figure 4.16 Figure 4.17, Figure 4.20 Figure 4.21, and Figure 4.24 Figure 4.25 above. Q3 Pauses (VD) is superior to characters per view and writing-times (HS) in terms of showing improvement in measuring Java competence; this may be because the VD measure is better fitted to the specific nature of the transcription task. Characters per view and Q3 pauses showed better improvement than writing-times. The writing process is variable and more complicated than just looking at the stimulus; as participants write, several variables that can interact with the process that are not explicitly relevant to Java competence may be present. For example, participants will return to the stimulus when writing to double-check that what they wrote is right. Pauses,

meanwhile, simply calculate the time it takes to process and plan to write the next character, thus pauses are more closely linked to what participants have in their memories. Accordingly, $pause_{Q3}$ is believed to be a reliable indicator of Java competence in transcription tasks. View-times showed no differences (improvement between part 1 and part 2) because view-times are not linked to Java competence, as expected in H3.

The figures also show that the basic category had the greatest improvement in characters per view, writing-times, and Q3 pause mean values between simple and complex, which matches my predictions derived from the stimuli complexity analysis (Table 4.5 above). The difference is clearer than it is for results part 1 (Figure 4.1, Figure 4.2, and Figure 4.4 above); this is due to taking the complexity factor into account in addition to the difficulty factor when categorising the stimuli as two complexity levels.

The high-competence group outperformed the low-competence group. For characters per view, writing-times, and Q3 pauses, the high-competence group performed better than the low group for both basic (simple and complex) and advanced (simple and complex) categories. The Q3 pauses measure is superior to characters per view and writing-times in terms of showing better performance for the high over the low group; this may be because the VD measure is better fitted to the specific nature of the transcription task.

Performance on the simple stimuli should be greater than on the complex stimuli. This is reasonable since simple stimuli are intended to be easy, and most participants should be able to transcribe them well. The results significantly support this for characters per view, writing-times, and for $pause_{Q3}$ (superior to the main analysis (part 1)), which is consistent with both the first experiment (Chapter 3) and the stimuli complexity analysis (Table 4.5). However, for characters per view, for the advanced category only, performance on the complex stimuli (A1.2, A2.2) was slightly superior to the simple stimuli (A1.1, A2.1). This may be attributed to differences in the nature of the stimuli for both categories, as the complex part has some lines

with repetitive syntax with slight differences, making the next line easier to understand. For view-times, as well, participants viewed the A1.2, A2.2 stimuli for shorter times than A1.1, A2.1. This may be because of the reason listed above concerning the better performance for characters per view on the complex over the simple. Therefore, it may be said that under certain very particular circumstances, view-times can be affected where there is similarity between parts of the stimuli; if anything is repeated, it will be easier to read the second time.

Concerning the PPCS test, the results are consistent with the main analysis (part 1), implying that the influence of transcribing the Java stimulus extends beyond the act of simply transcribing any stimuli.

In conclusion, this study further demonstrates a noticeable discrimination between the stimuli of two different levels of complexity (simple and complex), with the basic category showing the greatest discrimination and improvement in mean values over all behavioural measures, as anticipated from the stimuli complexity analysis (Table 4.5). It demonstrates that the high-competence group outperforms the low-competence group over the two stimuli complexity levels across all the behavioural measures. Additionally, as predicted from Table 4.5, the performance on simple is higher than on complex, which is more evident in the basic category than the advanced one. According to the stimuli complexity analysis, this is rational, and provides some evidence that the complexity factor has an effect, which suggests that when designing stimuli, it is not enough to consult the Java module (difficulty factor), but it is also necessary to consider the stimuli complexity factor.

Finally, it is worth noting that Q3 pauses showed the greatest improvement in the mean values of all the behavioural measures between simple and complex; This might be due to the VD metric being more suited to the unique nature of the transcribing process.

4.4 Overall discussion

Previous studies have shown the possibility of utilizing temporal chunk indicators acquired in transcription tasks as measurements of participants' competence, across different domains. Generally, both the first experiment (Chapter 3) and this experiment's main and detailed analysis permit me to answer 'Yes' to the thesis's main question, 'Can programming competence be measured by analysing patterns of chunk behaviour in the task of program code transcription?'. For both experiments, characters per view, writing-times, and $pause_{Q3}$ (second experiment) made a significant contribution ($p<0.05$) in predicting familiarity scores. Q3 pauses are obtainable in the view display (VD) mode of transcription, while characters per view, view-times, and writing-times are obtainable in the hide and show (HS) presentation mode.

This experiment examined the possibility of using the above-listed temporal chunk signals to assess Java programming competence in a transcription task through my seven hypotheses. Hypothesis H1 is concerned with the characters per view increase as familiarity scores increase. Hypothesis H2 reflects on the rise of writing-times with a rise in familiarity. Hypothesis H3 focuses on view-times and predicts that there will be no association between them and Java competence. Hypothesis H4 considers the decrease in Q3 pauses as familiarity scores rise. Hypothesis H5 concerns the better performance on basic than advanced stimuli. Hypothesis H6 considers the likelihood of improving characters per view, writing-times, view-times, and Q3 pause values after PPCS normalisation. And finally, hypothesis H7 concerns the existence of a direct significant relationship between writing-times and characters per view. Generally, H1, H2, H3, H4, H5, and H7 are all supported, though the details vary: characters per view (HS), writing-times (HS), and $pause_{Q3}$ (VD) significantly discriminate participants' levels of Java competence. Therefore, these behavioural measures have the potential to predict Java competence in transcription tasks. This experiment proposes that characters per view and Q3 pauses are

reliable predictors of Java competence, although Q3 pauses become a better predictor of Java competence than characters per view.

There was no noticeable difference between basic and advanced stimuli in the participants' temporal chunk signals (difficulty factor) when compared with the previous experiment (Chapter 3). As a result, a detailed analysis was performed, and the stimuli were further subdivided into two complexity levels (simple and complex), according to the stimuli complexity analysis (subsection 4.3.3.1). Generally, the detailed analysis shows more differences in participant behaviour between the two stimuli complexity levels than the main analysis. This means that the goal of performing the second analysis has been achieved. The distinction between simple and complex is visible in the basic (Figure 4.18) more than the advanced (Figure 4.19) categories, as anticipated from Table 4.5. Characters per view and Q3 pauses showed greater improvement in distinguishing simple and complex stimuli than writing-times; these two measures have also been identified as more accurate predictors of Java competence for the main analysis (part 1). View-times remained constant, as expected according to H3. Besides this, the difference in participant performance between simple and complex (complexity factor) is significant, in contrast to the main analysis (part 1) and compatible with both the first experiment (Chapter 3) and the stimuli complexity analysis (Table 4.5). So, one of the aspects that seems to be particularly important to consider when designing stimuli is the stimuli complexity factor. As a consequence, there are some factors that we should pay attention to while designing stimuli: participants' level of competence, stimuli difficulty levels (i.e., consulting the modules), content analysis results (Chapter 3), stimuli complexity analysis (Table 4.5), and avoiding using stimuli with similar syntax (this may be a problem, as discovered in the advanced category, complex part (A1.2 and A2.2)).

In terms of the differences between the first and second experiments, in summary:

- The stimuli were designed and categorized in basic and advanced among the first and second experiment differently:
    - First experiment: by consulting the undergraduate student Java modules (i.e., difficulty factor).
    - Second experiment main analysis part 1: by consulting (1) the undergraduate student Java modules; (2) the content analysis (Chapter 3), which showed that participants choose to see the stimuli more often (i.e., take breaks) before transcribing the following categories: punctuation marks, capitals within variables or reserved words, and spaces. What was learned was that consulting these two aspects is not enough to improve the difference in the behavioural measures' values between the two stimuli difficulty levels.
    - Second experiment detailed analysis part 2: by consulting the stimuli complexity analysis (Table 4.5), which inspired me to split each stimulus into two complexity levels (simple and complex), what was discovered is that when the complexity factor is taken into account, there is an increase in the disparity between the stimuli complexity levels for all behavioural measures. In particular, Q3 pauses outperforms the HS measures in terms of showing improvement in differentiating levels of complexity. In more detail, characters per view have greater differences than writing-times, while view-times are consistent.
- In the first experiment, only HS measurements (view-numbers, writing-times, view-times) were used, while in this experiment, both HS and VD measures (Q3 pauses) were used. What was discovered is that HS and VD measures are both reliable indicators of Java familiarity scores.
- The first experiment used practise items for normalisation, which did not improve the HS measures. In this experiment, the behavioural measures were normalised for a PPCS

(Participants Preferred Cluster Size) test, and this did not increase characters per view, writing-times, or Q3 pause values. This suggests that the behavioural measures accurately predict Java performance.

- In the first experiment, the low-competence group consisted of a wider range of educational levels (undergraduate to faculty members). In this experiment, the low-competence group was a concentrated group (all were first-year undergraduates).
- The number of participants in this experiment (51) was twice the number of the first experiment (24).
- In this experiment, linear regression was used to investigate the relationship between the various variables, whereas the first experiment only used correlations. Both experiments revealed a strong relationship between behavioural measures and familiarity. This implies that the behavioural measures accurately predict Java competence.

4.4.1 Suggestions and future work

It would be possible to repeat the same experiment with the same stimuli and with the same group of participants after a short period of learning. The idea would be to learn if performing a longitudinal study would result in significant changes in students' behavioural temporal chunk signals and Java programming competence.

Why is it now necessary to do a longitudinal study in addition to the cross-sectional research (first and second experiments, Chapters 3 and 4) and other related studies (Cheng, 2014, 2015; Zulkifli, 2013)? This is an important step in trying to apply the approach to more realistic samples with a range of ability similar to that found in real tests. Also important in demonstrating the sensitivity of the whole approach (i.e., measuring competence using temporal chunk signals in a transcription task). It is as well necessary to investigate whether short-term learning gains will influence chunk structures.

The low-competence group were assigned to perform the next experiment, since, according to the results of this experiment, characters per view, writing-times, and Q3 pauses strongly predict familiarity in low-competence participants, more than for those of high competence. Furthermore, since all low group participants are first-year undergraduate students, I want to see if repeating the experiment with only those on the Java module would show an improvement in their behaviours. Will we see the small changes they have experienced after a few months of learning? I tested to see how the behavioural measures will handle small changes over the course of a single module. Participants in the first and second experiments, on the other hand, had a wider range of Java programming expertise.

4.5 Summary

- Pauses, in addition to view-numbers and writing-times, have potential as a measure of programming competence.
- Normalizing for PPCS had no effect on the measures (individual differences do not impact the effectiveness of the behavioural measures).
- Significant relations were found between the dependent measures (Characters per view, writing-times, and pauses) and the independent measure of competence (i.e., familiarity scores).
- Significant relationships between Characters per view and writing-times were again spotted, similarly to the previous experiment.
- In most cases $pause_{Q3}$ becomes a better predictor of Java competence than characters per view.

- After splitting each stimulus into further complexity levels: basic into (simple and complex) and advanced into (simple and complex), the difference in the participants' performance between the two levels of stimuli complexity becomes clearer.

5 Experiment 3: Longitudinal post-test study of the impact of learning on participants' behaviour

Chapter content:

- An introduction that includes the experiment hypothesis and key questions.
- The experiment methodology:
    - Overall experiment design.
    - Participants.
    - Employed materials.
    - Stimuli design.
    - The experiment procedures.
- Results and discussion:
    - Results part 1: comparing pre-test and post-test.
        - The behavioural measures.
    - Discussion part 1.
    - Results part 2: The effectiveness of the behavioural measures at detecting learning gain
        - Independent measures of competence.
        - Regression analysis for the behavioural measures against final exam marks.
        - Regression analysis for the behavioural measures against characters per view.
    - Discussion part 2.
- Overall discussion
- Summary.

5.1 Introduction

The previous experiments (Chapters 3 and 4) demonstrated the potential of my method of evaluating Java competence by assessing chunk structures in a transcription task. It has been shown that the behavioural measures (characters per view, writing-times, $pause_{Q3}$) can substantially predict the competence measure (familiarity). The second experiment (Chapter 4)

showed that $pause_{Q3}$ predicts Java competence better than both characters per view and writing-times. On the other hand, the PPCS (Participants Preferred Cluster Size) normalization did not improve the behavioural measures. And there was no significant correlation found between view-times and Java programming competence.

The purpose of this experiment is to see whether it is possible to measure an actual learning gain by participants over the duration of a course in Java. This is the first time this has been done in any of the experiments, so it is particularly novel. The narrow range of participants (all participants will be treated here as a single group, rather than being divided into low and high) is a consequence of selecting just those participants who previously took part and were at the beginning of their Java course. This presents a challenge to the temporal chunk transcription method, because the range of competence is narrower. I was fortunate to have the students' exam marks as an additional measure, but this was not one of the main purposes of the experiment.

This experiment therefore aimed to answer the question:

Can the method be used to measure learning gains over the course of one term of study?

As the same participants enrolled in both the second (previous) experiment and this one, and all studied the same module over the same period of time, I predict that:

Given that the size (the total number of characters) of each Java stimulus is known:

Ha) The number of characters per view will increase after learning.

As more capable participants' chunks contain more information:

Hb) The duration of written answers following each stimulus view will be longer after learning.

Now, if the time it takes to interpret a chunk is roughly constant (Chase & Simon, 1973), and if the amount maintained per view is independent of competence, I predict:

Hc) The amount of time spent on each individual view of the stimulus will be unrelated to learning gain.

Based on chunking theory, I predict:

Hd) The length of the $pause_{Q3}$ will be shorter after learning.

Since commonly used Java components implemented earlier (in students' programming module) during teaching are assigned to basic stimuli, I predict:

He) Participants' performance will be better on the basic stimuli than on the advanced ones after learning.

To reconfirm the results of the first and second experiments (Chapters 3 and 4) I have the same H1, H2, H3, H4, H5, and H6 hypotheses as the second experiment (Chapter 4), but also consider exam marks as an independent variable in addition to the familiarity measure.

I am therefore also seeking to answer this question:

Will the behavioural measures be effective at detecting the learning that occurred during the Java course?

As the number of characters of each stimulus is constant, I predict:

H1) The number of characters per view of a particular stimulus in a trial will be greater for more competent participants (higher familiarity scores and exam marks).

As the more competent participants' chunks contain more content, I predict:

H2) Writing-times will be longer for more competent participants (higher familiarity scores and exam scores).

So, as the time to comprehend a chunk is roughly stable (Chase & Simon, 1973), and if the number of characters retained per view is independent of competence, then I expect:

H3) View-times will not be directly related to programming competence.

Based on chunking theory and on the specific hierarchical models for different levels of competence, as shown in the pause generation model (Chapter 2, section 2.4.4), I anticipate:

H4) $pause_{Q3}$ will be longer for less competent participants (lower familiarity scores and exam marks).

As basic and advanced stimuli will be used here again, and as the basic Java concepts are applied for the basic stimulus, I predict:

H5) The performance on basic stimuli will be higher than on the advanced stimuli.

Since chunking processes are predicted to affect characters per view and writing-times, there should be a consistent and systematic association between them; therefore, observing the relationship between them provides further proof that chunking occurs (proved in the first and second experiments). Thus, I predict:

H6) Writing-times will be directly related to characters per view.

This chapter contains three key sections: Methodology, Results, and Discussion. It consists of two main 'results and discussion' analyses. The first section contrasts both pre-test and post-test findings. The second is the use of the basic (B1, B2) and advanced (A1, A2) stimuli.

5.2 Method

This experiment was carried out using the same experimental design as the second experiment. However, here the focus is on comparing participants' performance before and after undertaking the Java module. Thus, this section describes only the minor variations between the second and third experiments.

5.2.1 General experiment design

The experiment is a counter-balanced 2x2 design (two VD and HS display factors and two basic and advanced stimuli difficulty factors). The previous experiment was a mixed design with low- and high-competence participants and basic and advanced stimuli, while this experiment is a within-participant factor with each participant transcribing basic and advanced stimuli, since the participants in this experiment are of the same educational level. In the previous experiment, they were from a wider educational context.

5.2.2 Participants

21 adults from the School of Engineering and Informatics at the University of Sussex participated. Participants are first-year undergraduate students who have just finished their first semester and completed their Java module exams. Their ages ranged from 18 to 25 years (mean= 20.5, SD= 3.5); 14 were male and 7 were female. They were given £10 for their involvement.

5.2.3 Materials

Please refer to the Materials section of the second experiment (Chapter 4). The only difference here is that the online questionnaire consisted of one section (not four parts, as in Chapter 4). After the participants had carried out the transcription task, they were only asked to fill in the portion that assessed their familiarity with the Java stimuli provided in the experiment.

5.2.4 Stimuli design

The stimuli in this experiment are the same as the one used in the second experiment, see the section on stimuli design in Chapter 4.

(Additionally, the PPCS (Participants Preferred Cluster Size) test was conducted with quintuple (i.e., S5) and sextuple (i.e., S6) such as ‘9g2b6 d7f5w …’ and ‘z8b3n2 7y3e5n …’ etc stimuli in

order to extend the previous PPCS data. However, the analysis of PPCS is not included in this chapter.)

5.2.5 Procedure

The procedure in this experiment is the same as the one in the second experiment. See the Procedure section in Chapter 4.

5.3 Results and discussion

The experiment's results consist of two parts: part 1 compares pre-test and post-test results; part 2 reports the regression analysis findings across all the measures for all students as a single category, using the basic and advanced stimuli difficulty factor.

5.3.1 Results part 1: Comparing pre-test and post-test results

The primary goal of this experiment was to determine if the behavioural measures are sensitive to learning development over a one-module period. That is, to see if repeating the same experiment with those on the Java module would result in an increase in their performance before and after learning.

This subsection seeks to answer the first question: Does examining the same students after a short period of learning show a significant difference in their performance, and thus their Java programming competence? Do the measures reflect the learning that the students did?

5.3.1.1 *Behavioural measures*

The following subsections present characters per view, writing-times, view-times, and $pause_{Q3}$. Figures Figure 5.1 to Figure 5.8 below show the performance for each participant in both pre-test and post-test, and the difference before and after learning for each measure and for basic and advanced separately. Table 5.1 below shows the mean and SD for the differences between

pre-test and post-test for all behavioural measures across both stimuli difficulty levels, basic and advanced.

Measures	HS mesures						VD measure	
	Char/View Basic Posttest - Pretest	Char/view Advance Posttest - Pretest	WT Basic Posttest - Pretest	WT Advance Posttest - Pretest	VT Basic Posttest - Pretest	VT Advance Posttest - Pretest	pause Basic Posttest - Pretest	Pause Advance Posttest - Pretest
mean	1.8	1.0	0.9	0.4	-0.1	0.0	-105.1	-93.5
SD	1.7	1.0	1.3	0.9	0.3	0.3	151.2	115.0

Table 5.1 Mean and SD for the difference between post-test and pre-test behavioural measure values across all behavioural measures over basic and advanced stimuli, HS=hide and show, VD=view display, WT=writing-times, VT=view-times

5.3.1.1.1 Characters per view

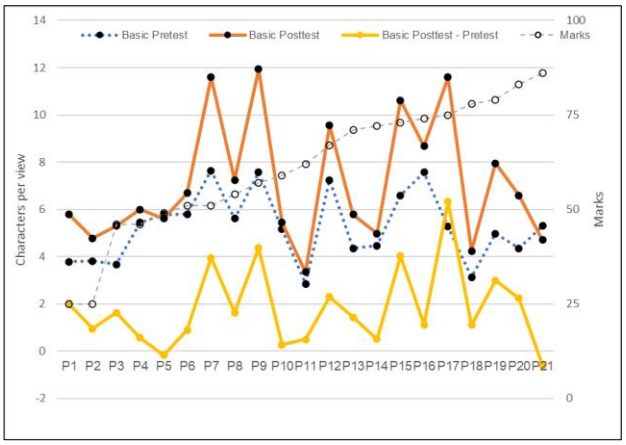

Figure 5.1 Total characters per view for basic stimuli across pre-test and post-test; participants are ranked by their exam marks

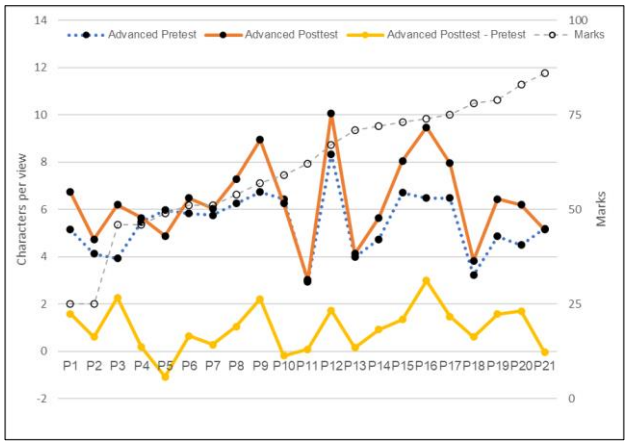

Figure 5.2 Total characters per view for advanced stimuli across pre-test and post-test; participants are ranked by their exam marks

Consistent with prediction Ha, which concerns the increase in the number of characters per view after learning, for all participants, Figure 5.1 and Figure 5.2 show that the number of characters per view increased from pre-test to post-test by about 2 characters per view (about 20%) for the basic stimuli (mean=1.8, SD=1.7) and by 1 character for the advanced stimuli (mean= 1.0, SD= 1.0).

For the basic stimuli, the post-test scores are higher in 19 of the 21 cases than the pre-test scores; 1 case has almost identical scores, while the other is a little lower. For the advanced

stimuli, the post-test scores are greater in 15 of the 21 cases than the pre-test scores, 5 cases have almost the same scores, and 1 case is slightly worse. The increase is significant for both stimuli difficulty levels, basic (5.26 (pre-test) vs 7.07 (post-test), t=-4.93, p=.00, df=19, 1 tail) and advanced (5.38 (pre-test) vs 6.34 (post-test), t=-4.54, p=.00, df=19, 1 tail).

Consistent with prediction He, which predicts that participants' performance will be better on the basic stimuli than on the advanced after learning, across all participants, the basic stimuli (Figure 5.1) show more characters per view after learning than the advanced stimuli (Figure 5.2), i.e., participants' performance on the basic stimuli is higher than on the advanced stimuli, which is significant (basic: 7.1 vs advanced: 6.3, t=-1.96, p=.032, df=19, 1 tail).

5.3.1.1.2 Writing-times

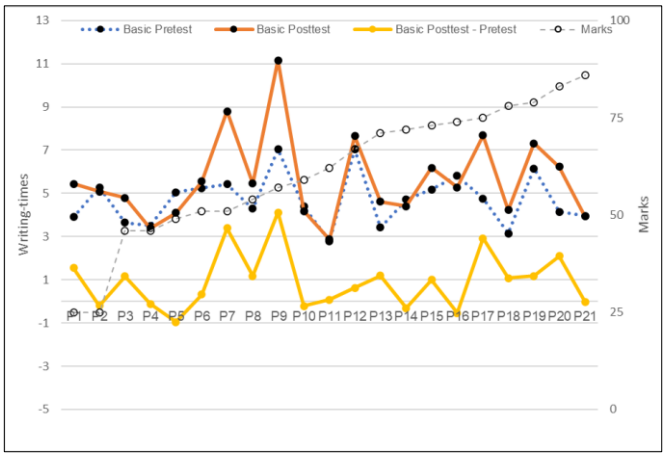

Figure 5.3 Median writing-times for basic stimuli across pre-test and post-test; participants are ranked by their exam marks

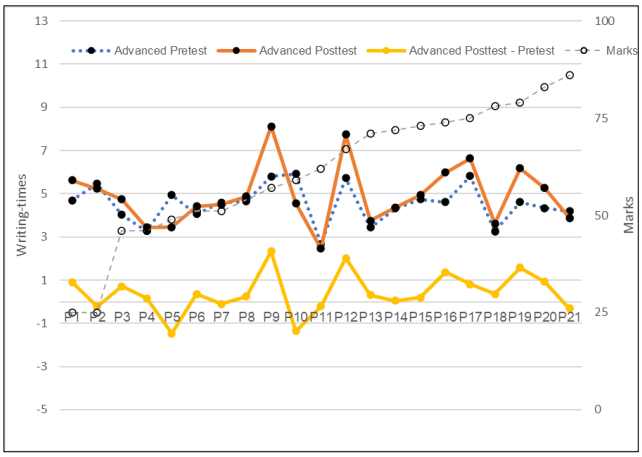

Figure 5.4 Median writing-times for advanced stimuli across pre-test and post-test; participants are ranked ordered by their exam marks

Consistent with prediction Hb, which predicts that the duration of written answers following each stimulus view will be longer after learning, for all participants, Figure 5.3 and Figure 5.4 show that there was a significant improvement in writing-times after learning for both basic and advanced stimuli. Participants took nearly a second longer while writing the basic stimuli than the advanced. For basic stimuli, the post-test scores are greater in 13 of the 21 cases than the

pre-test scores, 2 cases deteriorate marginally, while 6 cases have almost equal scores. For the advanced stimuli, the post-test scores are higher in 9 out of the 21 cases than the pre-test scores, 2 cases deteriorate slightly, and 10 cases have almost identical scores. The difference between the pre-test and post-test means for the basic stimuli is significant (mean=0.9, SD=1.3). So, the writing-times for the basic stimuli show that participants clearly improved (4.71 (pre-test) vs 5.64 (post-test), t=-3.22, p=.00, df=19, 1 tail). But for the advanced stimuli, the difference between the means (mean=-0.4, SD=0.9) is not a lot but still significant (4.53 (pre-test) vs 4.94 (post-test), t=-1.99, p=.03, df=19, 1 tail). This trend is identical for the characters per view above, as well as for pauses (below), but not for view-times.

Consistent with prediction He, across all participants, the basic stimuli (Figure 5.3) saw longer writing-times after learning than the advanced stimuli (the output of the basic stimuli was higher than that of the advanced stimuli) which is significant (basic: 5.6 vs advanced: 4.9, t=-2.78, p=.006, df=19, 1 tail).

5.3.1.1.3 View-times

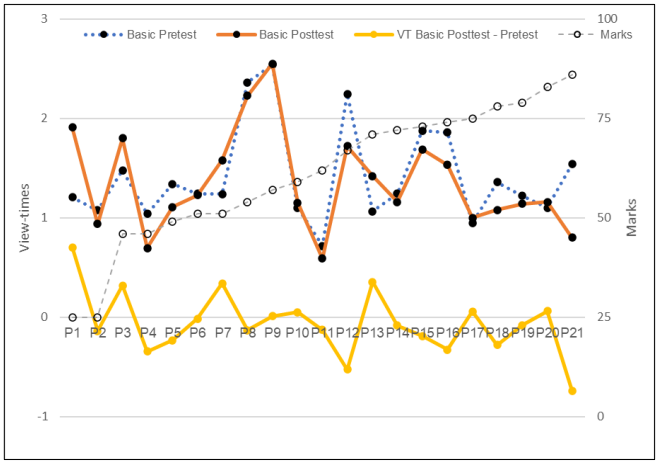

Figure 5.5 Median view-times for basic stimuli across pre-test and post-test; participants are ranked by their exam marks

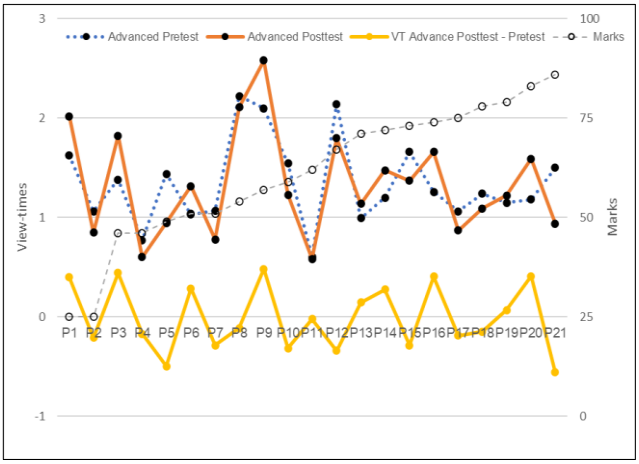

Figure 5.6 Median view-times for advanced stimuli across pre-test and post-test, participants are ranked by their exam marks

Prediction Hc predicts that the amount of time spent on each individual view of the stimulus will be unrelated to learning gain. Figure 5.5 and Figure 5.6 reveal that there was no improvement

in participants' performance before and after learning, despite the fact that there has been improvement since they completed the Java course. The above figures do not show a clear relation between view-times and Java competence, as they do not show general increasing or decreasing trends for either basic or advanced stimuli before and after learning. In general, with both basic and advanced stimuli, most cases have almost equal pre-test and post-test scores because the data points are very similar to each other, with post-test scores being marginally lower in some cases. To sum up, analysis of participants' performance differences before and after learning supports the claim that view-times do not predict Java competence and therefore are not a valid indicator of competence. Further, view-times' trends differ from the trends in Figure 5.1 to Figure 5.4.

Consistent with prediction He, across all participants, there is no significant difference between basic and advanced stimuli, after learning, in terms of view-times (basic: 1.4, advanced: 1.3, t=-0.442, p=.33, df=19, 1 tail).

5.3.1.1.4 $Pause_{Q3}$

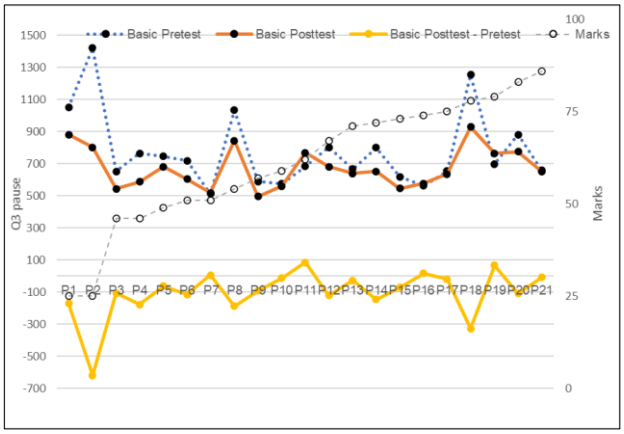

Figure 5.7 Q3 pauses for basic stimuli across pre-test and post-test; participants are ranked by their exam marks

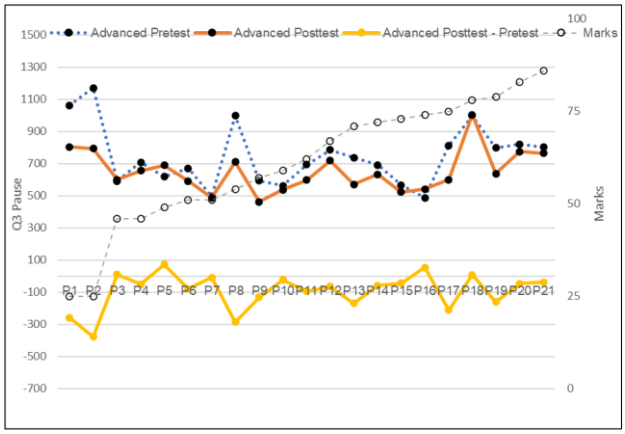

Figure 5.8 Q3 pauses for advanced stimuli across pre-test and post-test; participants are ranked by their exam marks

Consistent with prediction Hd, which predicts a reduction in the duration of $pause_{Q3}$ after learning, for all participants, Figure 5.7 and Figure 5.8 show that the length of Q3 pauses

decreases by about 105 milliseconds for the basic stimuli (mean=-105.1, SD=151.2) and by about 93 milliseconds for the advanced stimuli (mean=-93.5, SD=115.0).

For the basic stimuli, in 13 of the 21 cases, the post-test scores are smaller than the pre-test scores, 6 cases have almost equal scores, and 2 cases have slightly increased scores. When it comes to advanced stimuli, the post-test scores are lower in 12 of the 21 cases than the pre-test scores, 7 cases have almost the same scores, and in 2 cases they are marginally higher. The decrease in the length of Q3 pauses is significant between pre-test and post-test for both basic stimuli (778.15 (pre-test) vs 673.04 (post-test), t=-3.21, p=.00, df=19, 1 tail) and advanced stimuli (745.67 (pre-test) vs 652.21 (post-test), t=-3.76, p=.00, df=19, 1 tail). In other words, participants' pauses after learning were shorter than they were prior to learning.

Contrary to prediction He, which predicted that participants' performance on basic stimuli would be higher than on advanced stimuli once they have learned, across all participants, Figure 5.7 shows that the basic stimuli had slightly longer Q3 pauses than the advanced stimuli (Figure 5.8). Thus, the output of the basic stimuli is not higher than that of the advanced stimuli, though this is not significant (basic: 673.0, advanced: 652.2, t=-1.363, p=.094, df=19, 1 tail).

Concerning all the behavioural measures (HS and VD), table Figure 5.1 above shows that all the behavioural measures reflect the learning gained by the students. The yellow lines in Figure 5.1 to Figure 5.8 above show the disparity between participants' performance before and after learning for both basic and advanced characters per view, writing-times, view-times, and Q3 pauses. In general, the variations in output pre-test and post-test for characters per view, writing-times, and Q3 pauses are present across all participants and follow a similar trend for basic and advanced. When the differences between the two stimuli difficulty levels are compared, the differences are larger for basic than for advanced. View-times, when considered individually, do not show a significant difference in participant performance. When it comes to characters per view and writing-times, when the difference in one measure is high, the other

measure is also large. Furthermore, the Q3 pauses reveal major variations in the performance of the participants. Another point to make is that, in all measures and levels of exam scores, there are improvements in students' performance from pre-test to post-test, not only for students with high marks, but also for students with low marks.

To summarise, characters per view, writing time, and Q3 pauses clearly represent the learning that students gained. The trend (improvement in participants' output after learning) between basic and advanced is consistent for characters per view and writing-times. Participants' progress pre-test to post-test is consistent across the HS measures.

Basic	Characters per view	Writng-times	View-times	Pauses
Characters per view				
Writng-times	0.83			
View-times	0.34	0.47		
Pauses	-0.17	-0.14	-0.05	

Advanced	Characters per view	Writng-times	View-times	Pauses
Characters per view				
Writng-times	0.84			
View-times	0.67	0.56		
Pauses	0.01	0.01	0.05	

Table 5.2 Correlations for post-test - pre-test scores between various behavioural measures (n=21, Pearson correlation, 1 tail, critical value is 0.5034 at p<.01)

Table 5.2 presents the correlations between disparities in student performance before and after learning (post-test and pre-test), demonstrating that differences for characters per view and writing-times are strongly correlated for basic and advanced. In other words, the greater the difference in the number of characters per view, the greater the difference in writing-times for every participant. View-times were significantly correlated with both characters per view and writing-times for advanced stimuli only. For the basic stimuli, the VD measure (Q3 pauses) is negatively correlated (not significant) with the other HS measures for the basic stimuli, whereas for the advanced stimuli, the correlation values are close to zero.

5.3.2 Discussion part 1

In studies of related style, such as Cheng (2014, 2015), van Genuchten and Cheng (2010), and Zulkifli (2013), participants had varying levels of expertise (education), as in my first and second

experiments (Chapters 3 and 4). However, the previously listed studies did not measure improvement over a period of time. Thus, this experiment is the first to utilize participants from the same educational background (expertise level) and to measure their improvement over a period of time. The reason for focusing on a single year group was to try to apply the approach to more realistic samples with a range of ability similar to that found in real tests. As a result, we will be able to answer the main question of this experiment, which is whether examining the same students after a short period of learning time (3 months) will reveal a significant difference in their performance.

Hypothesis Ha predicts that characters per view will increase after learning gain and hypothesis Hb predicts that writing-times will increase after learning gain. The findings are remarkably consistent with these predictions, which gives extra support to my base predictions H1 and H2. Both characters per view and writing-times increased significantly after learning for almost all participants. This is because participants' behaviour (total number of characters per view and their writing durations) were significantly affected by their new chunk structures after learning. Hence, according to chunking theory, their chunk structure hierarchy will have less layers. Hypothesis Hc predicts that there is no relationship between view-times and learning gain. The results are consistent with this prediction and provide extra support to my base prediction H3. This assumes that any chunking effects that could occur during stimulus viewing will be eliminated.

Hypothesis Hd predicts that Q3 pauses will decrease after learning gain. The results are remarkably consistent with this prediction, and this adds weight to my fundamental prediction H4. The students showed significantly shorter pauses after learning, which means that participants' length of pauses was significantly affected by their improved Java competence, and thus their new chunk structures.

Hypothesis He predicts that participants' performance will be better on the basic stimuli than advanced after learning, which would strengthen my base prediction H5. The results are significantly consistent with this prediction for characters per view and writing-times; this is as predicted, as the basic stimuli are supposed to be easier than the advanced stimuli. On the other hand, the prediction is not upheld for Q3 pauses, which is similar to what was found in the previous experiment (because of the details of the design of the stimuli (complexity factor not considered); justification can be found in Chapter 4).

The differences in the participants' performance after learning (Figure 5.1–Figure 5.8) is more consistent for characters per view and writing-times (i.e., significantly correlated: the greater number of characters per view, the longer the writing-times) rather than Q3 pauses. This may be due to the nature of the transcription task; characters per view and writing-times are collected in the HS presentation mode which is different from the VD mode in which Q3 pauses were collected.

Thus, I can answer 'Yes' to my main question: Does examining the same students after a short period of learning time show a significant difference in their performance, thus, their Java programming competence? The results show significant improvement in the mean scores of all the behavioural measures after learning.

To conclude, this experiment shows a disparity in participant performance over a particular time, despite the fact that the extra experience is just three months.

5.3.3 Results part 2: The effectiveness of the behavioural measures at detecting learning gain

The results in this section consider the following: (1) the correlations between the independent variables and between these and the dependent variables, and the justification for the dedicated competence measure (exam marks) for this experiment. (2) The six hypotheses concerning the relationships between the behavioural measures (characters per view, writing-

times, view-times, and pauses) and the competence measure (exam marks). (3) And finally, the six hypotheses related to the relation of the writing-times to characters per view.

5.3.3.1 *Independent measures of competence*

In this section, I will look at the relationship between the independent variables, familiarity and exam marks, across the behavioural measures (characters per view, writing-times, view-times, and $pause_{Q3}$).

	Independent variables	Dependent variables	Basic	Advanced
Correlations	Familiarity	Characters per view	0.157	0.037
	Marks		0.270	0.454
	Familiarity	Writng-times	0.143	0.073
	Marks		0.189	0.380
	Familiarity	View-times	-0.239	-0.126
	Marks		0.293	0.281
	Familiarity	Pause	0.020	0.070
	Marks		-0.143	-0.489

Table 5.3 Correlation between competence measures with both familiarity and exam marks (n=21, Pearson correlation, 1 tail, critical value is 0.503 at p<.01, 0.369 at p<.05)

In this experiment there were two independent variables, familiarity and exam marks. Familiarity was examined via online questionnaire, which is similar to what was implemented in the first and second experiments (Chapters 3 and 4). As the students were from the same year group, their exam marks are available to use. The students' final exam took place before they completed the post-test; students' marks are based on their final exam scores for their first Java module. The exam is a written exam that contains common programming questions such as implementing a piece of Java code according to the question requirements, or providing the students with the program code and asking comprehension questions that assess their understanding of Java concepts.

Table 5.3 above shows the correlations across both dependent and independent variables over basic and advanced stimuli. The table reveals that the behavioural measures correspond with

exam scores rather than familiarity. None of the correlations with familiarity are significant. The correlations are significant, for the exams marks only, at p<.05 for characters per view, writing-times and Q3 pauses for the advanced stimuli only. In other words, the higher the exam marks, the more characters per view, the longer the writing-times, and the shorter the Q3 pauses. For familiarity, correlation values are higher for basic stimuli than advanced stimuli, while the reverse is true for exam marks.

		Familiarity		Marks	
		Basic	Advanced	Basic	Advanced
N		21	21	21	21
df		19	19	19	19
Characters per view	Constant	0.21	0.03	0.04	0.05
	Intercept	6.01	6.16	4.57	3.41
	f	0.48	0.03	1.50	4.93
	significance f	0.496	0.872	0.236	0.039
	$R^{2\cdot}$	0.02	0.00	0.07	0.21
Writing-times	Constant	0.14	0.05	0.02	0.03
	Intercept	4.90	4.67	4.31	3.04
	f	0.40	0.10	0.70	3.20
	significance f	0.536	0.754	0.413	0.090
	$R^{2\cdot}$	0.02	0.01	0.04	0.14
View-times	Constant	-0.06	-0.03	0.01	0.01
	Intercept	1.66	1.50	0.84	0.81
	f	1.15	0.31	1.78	1.63
	significance f	0.297	0.586	0.198	0.217
	$R^{2\cdot}$	0.06	0.02	0.09	0.08
Pauses	Constant	1.23	4.52	-1.03	-3.64
	Intercept	666.76	629.18	735.84	874.37
	f	0.01	0.09	0.40	5.96
	significance f	0.933	0.765	0.536	0.025
	$R^{2\cdot}$	0.00	0.00	0.02	0.24

Table 5.4 Relationship of characters per view, writing-times, view-times, and pauses to familiarity and exam marks

As seen in the pre-test (Chapter 4), familiarity (independent variable) was well correlated with the behavioural measures (dependent variables). However, familiarity stopped serving as a metric in this experiment (the explanation for this will be discussed later), as shown in Table 5.4 above, which shows the linear regression analysis results for the dependent variables across the

independent variables for both basic and advanced stimuli. As a result, it was determined that the students' final exam scores would serve as the independent variable in this experiment. So, are characters per view, writing-times, and $pause_{Q3}$ still good measures to show the differences between the participants within this experiment? The post-test (this experiment) does not demonstrate the variations as clearly as the pre-test (previous experiment). This may be because there was a broader spectrum of experience in the previous experiment amongst the participants, while the participants have the same level of experience here. Also, all of the participants in this experiment have learned, so what was an appropriate level of test stimuli in the pre-test is no longer appropriate in the post-test.

5.3.3.2 *Regression analysis for the behavioural measures against final exam marks*

The following subsections explain each behavioural measure separately. This subsection examines the six hypotheses that connect temporal chunk signals to Java programming competence. A simple linear regression is applied. The findings for each behavioural measure are interpreted in relation to exam marks, as well as for writing-times, view-times, and Q3 pauses in relation to characters per view.

Overall, predictions H1, H3, H4, and H5 (not for view-times and Q3 pauses) are supported, with variations in details. There is weak support for prediction. The relationships' direction is as expected.

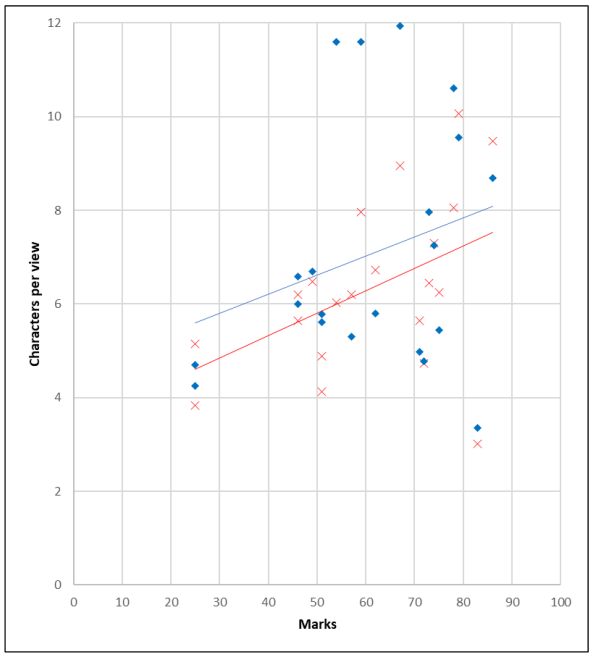

Figure 5.9 Lines for linear best fit of characters per view to exam marks for all participants, basic (blue) and advanced (red) stimuli

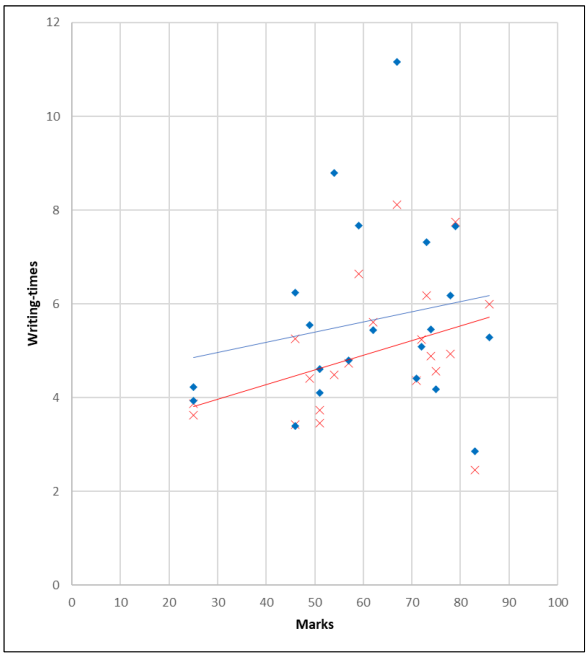

Figure 5.10 Lines for linear best fit of writing-times to exam marks for all participants, basic (blue) and advanced (red) stimuli

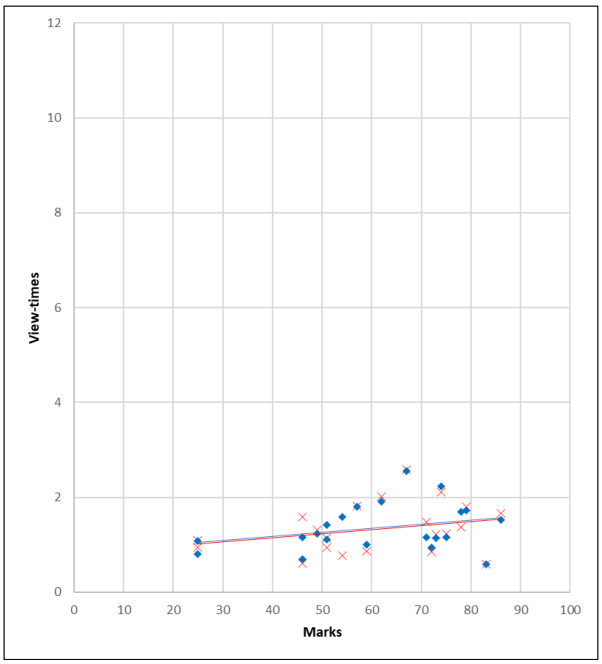

Figure 5.11 Lines for linear best fit of view-times to exam marks for all participants, basic (blue) and advanced (red) stimuli

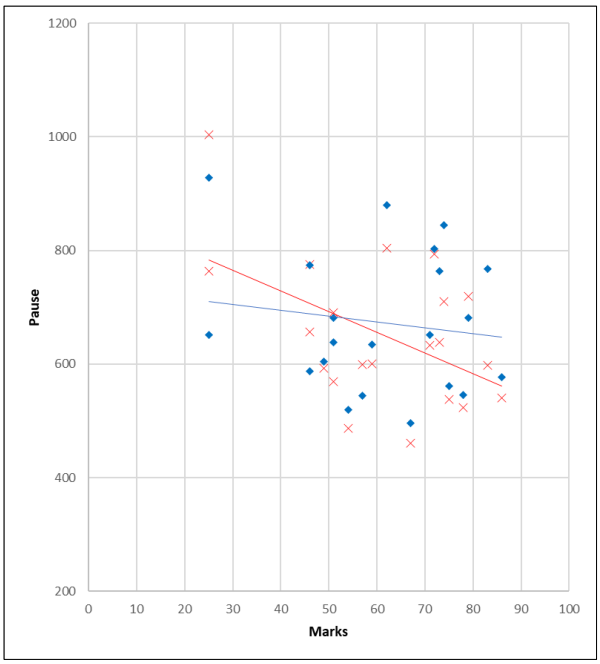

Figure 5.12 Lines for linear best fit of pauses to exam marks for all participants, basic (blue) and advanced (red) stimuli

Figure 5.9, Figure 5.10, Figure 5.11, and Figure 5.12 show the linear regression relations of characters per view, writing-times, view-times, and $pause_{Q3}$ to final exam marks for students,

who were ranked in order of exam marks. They also present the relations among all participants across basic and advanced stimuli.

The above graphs show that characters per view and Q3 pauses can be predictors for Java competence. The lines for characters per view, writing-times, and Q3 pauses are steeper for advanced stimuli than for basic stimuli, meaning that advanced stimuli outperform basic stimuli in distinguishing participant learning gain levels.

		Marks	
		Basic	Advanced
N		21	21
df		19	19
Characters per view	Constant	0.04	0.05
	Intercept	4.57	3.41
	f	1.50	4.93
	significance f	0.236	0.039
	$R^{2\cdot}$	0.07	0.21
Writing-times	Constant	0.02	0.03
	Intercept	4.31	3.04
	f	0.70	3.20
	significance f	0.413	0.090
	$R^{2\cdot}$	0.04	0.14
View-times	Constant	0.01	0.01
	Intercept	0.84	0.81
	f	1.78	1.63
	significance f	0.198	0.217
	$R^{2\cdot}$	0.09	0.08
Pauses	Constant	-1.03	-3.64
	Intercept	735.84	874.37
	f	0.40	5.96
	significance f	0.536	0.025
	$R^{2\cdot}$	0.02	0.24

Table 5.5 Relationship of characters per view, writing-times, view-times, and pauses to exam marks

5.3.3.2.1 Characters per view

Hypothesis H1 focuses on an increase in the total number of characters per view with an increase in the final exam marks. Table 5.5 shows linear regression models of characters per view as

functions of exam marks for all the participants with both stimuli difficulty levels (basic and advanced).

As predicted and shown in Figure 5.9 and Table 5.5, the lines show upward trends for both basic and advanced stimuli. Hence, characters per view increase with an increase in exam marks (the more competent participants are with Java, the more characters per view they have). Thus, characters per view can distinguish Java competence levels among participants. The advanced stimuli have a more stable relationship and a steeper line – greater constant – than the basic stimuli. In general, characters per view can predict Java competence, and the advanced stimuli is more suitable for the participant levels in this experiment. The goodness of fit value for advanced stimuli is greater than for the basic stimuli. Put another way, characters per view may have some potentials to measure Java competence for the advanced stimuli. H1 is partially supported as only the correlation for the advanced stimuli is significant but not for the basic stimuli.

Consistent with the fifth prediction H5, which concerns better performance on basic than advanced stimuli, for most of the participants, basic stimuli have more characters per view than advanced stimuli (i.e., participants performed better on basic stimuli than on the advanced stimuli). This is significant (basic: mean=7.1, SD=2.61 vs advanced: mean=6.3, SD=1.82, t=-1.96, p=.032, df=19, 1 tail). Put another way, the difference in the participants' performance between basic and advanced stimuli is not due to chance.

5.3.3.2.2 Writing-times

Hypothesis H2 predicts a rise in writing-times as final exam marks increase. Table 5.5 displays linear regression models of writing-times as a result of exam marks for all participants using both basic and advanced stimuli.

As expected, and as shown in Figure 5.10 and Table 5.5, the lines show a slight positive trend for both stimuli difficulty levels. Therefore, writing-times slightly rise with a rise in exam marks (the

more participants understand Java, the more time they spend writing). Writing-times thus may be able to distinguish participants' Java competence levels. The advanced stimuli have a steeper line – slightly greater constant – than the basic stimuli, thus, the advance stimuli are more suitable for the participants' levels of competence in this experiment. In general, writing-times in this experiment may have some potential to predict Java competence for the advanced stimuli only, but not as effectively as characters per view. However, H2 cannot be stated to be supported because the regression analysis correlations for both basic and advance stimuli are not significant.

Consistent with the fifth prediction, H5, which concerns better results on basic than advanced stimuli, over all the participants, basic stimuli have longer writing-times than advanced stimuli (i.e., participants' performance on basic stimuli is superior to that of advanced stimuli), which is significant (basic: mean=5.6, SD=1.98 vs advanced: mean=4.9, SD=1.42; t=-2.78, p=.006, df=19, 1 tail).

5.3.3.2.3 View-times

View-times are the subject of hypothesis H3, which predicts no relationship between view-times and Java competence. In the HS presentation condition, view-time is the length of each view of the stimulus. View-times vary from characters per view, writing-times, and Q3 pauses because the chunking hypothesis suggests that view-times have no relationship with competence. Table 5.5 shows linear regression models of view-times as a function of exam marks for all participants using both basic and advanced stimuli.

As presented in Figure 5.11 and Table 5.5, the view-times lines show a very slight upward trend (poor constant values) with exam marks, thus do not discriminate participants' levels of competence well for either basic or advanced. This is counter to the other behavioural measures. Thus, view-times cannot predict Java competence in transcription tasks, which is consistent with previous studies (Chapters 3 and 4).

All of the correlations are most likely due to chance. The R^2 values are close to 0 (very weak) for both basic and advanced stimuli. The scatter of data points along the regression lines is extremely large. As a result, view-times is classified as an inaccurate indicator of Java competence in transcription tasks. To conclude, there is support for prediction H3 and consistency with the previous two experiments (Chapters 3 and 4).

Contrary to the fifth prediction, H5, which states that basic stimuli show better results than advanced stimuli, in terms of view-times there is no noticeable distinction between basic and advanced stimuli (basic: mean=1.4, SD=0.50, advanced: mean=1.3, SD=0.53; t=-0.442, p=.33, df=19, 1 tail).

5.3.3.2.4 $Pause_{Q3}$

Hypothesis H4 focuses on a decrease in the length of Q3 pauses with an increase in final exam marks. Table 5.5 presents linear regression models of Q3 pauses as functions of exam marks for all participants with both basic and advanced stimuli. It should be noted that this measure is collected in VD, which differs from the HS method, which collects characters per view, writing-times, and view-times.

As expected, and as shown in Figure 5.12 and Table 5.5, there are clear descending trend lines for both basic and advanced stimuli. Hence, unlike characters per view and writing-times, Q3 pauses decrease with an increase in exam marks-steep lines- (the more competent participants are, the shorter $pause_{Q3}$ they have before they start writing each stroke). Thus, Q3 pauses can differentiate participants' Java competence. The advanced stimuli have a more consistent relationship with the exam marks and a steeper line – a higher constant – than the basic stimuli. In general, in this experiment, the advanced stimuli are better suited to the participant levels because the R^2 values for advanced stimuli are greater that for the basic stimuli. Hence, Q3 pause may have some potentials to measure Java competence for the advance stimuli only. H4

has some support because the correlations for advanced stimuli is significant but not for basic stimuli. The effect is weaker in this experiment compared with the previous experiment.

Contrary to the fifth prediction, H5, the basic stimuli had slightly longer pauses than the advanced stimuli, which is similar to what was found in the previous experiment (Chapter 4), for the participants with higher marks, as shown in Figure 5.12, but the difference is not significant (basic: mean=673.0, SD=123.90, advanced: mean=652.2, SD=128.25; t=-1.363, p=.094, df=19, 1 tail). In conclusion, H4 is supported.

In summary, the results are similar to the previous experiment's findings (Chapter 4) in terms of two points: (1) while there is some inconsistency in the details, the overall results for characters per view, writing-times, view-times, and Q3 pauses are consistent with H1, H2, H3, H4, and H5. (2) While both characters per view and Q3 pauses may have some potential to predict Java competence, in most situations, Q3 pauses are a slightly better predictor of Java competence than Characters per view.

The regression analysis for writing-times, view-times, and Q3 pauses against characters per view is presented in the following subsection to provide further evidence for the importance of chunking in transcription tasks.

5.3.3.3 *Regression analysis for the behavioural measures against characters per view*

The existence of a direct relationship between writing-times and characters per view is the concern of prediction H6, and as previously mentioned, writing-times increase as the number of characters per view for each participant rises. Prediction H6 was added to this experiment as a consequence of the previous studies' (first and second experiments, Chapters 3 and 4) observations about the relationship between writing-times, view-times, and $pause_{Q3}$ to characters per view. Since chunking processes are predicted to affect characters per view, writing-times, and Q3 pauses, there should be an explicit and systematic relationship between them. Since view-times are not meant to be chunk-based, there will be no actual relationship

between view-times and characters per view. The previous two experiments showed that writing-times significantly predicted characters per view, which provides further evidence of the importance of chunking in freehand transcribing. Thus, with this experiment, a positive relationship between characters per view and writing-times is predicted, as both are expected to improve as Java familiarity scores increase (according to H1 and H2). Whereas a negative relationship between characters per view and Q3 pauses is anticipated, since pauses are expected to decrease as Java familiarity scores increase (according to H4). I anticipate a poorer relationship with view-times, since view-times are not thought to be directly linked to programming competence (according to H3).

In summary, to further investigate whether chunking might explain the differences between writing-times, view-times, and Q3 pauses, scatter plots of these variables versus characters per view were plotted (Figure 5.13,Figure 5.14, and Figure 5.15) for all the participants in all the conditions of the experiment (i.e., basic and advanced).

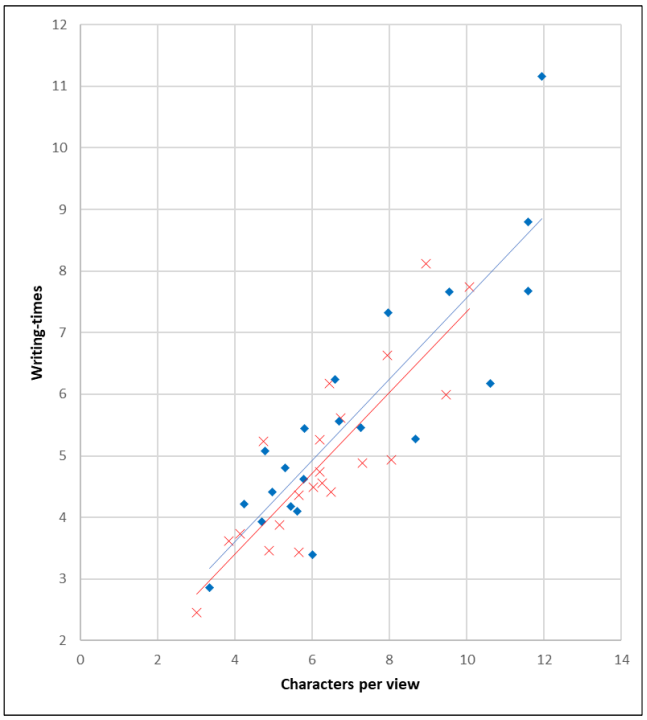

Figure 5.13 Relation of writing-times to characters per view (basic (blue) and advanced (red) stimuli)

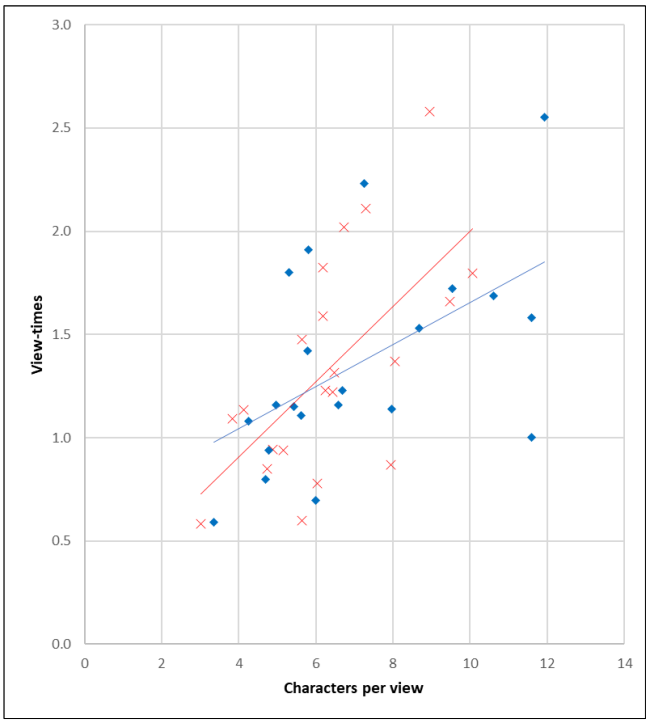

Figure 5.14 Relation of view-times to characters per view (basic (blue) and advanced (red) stimuli)

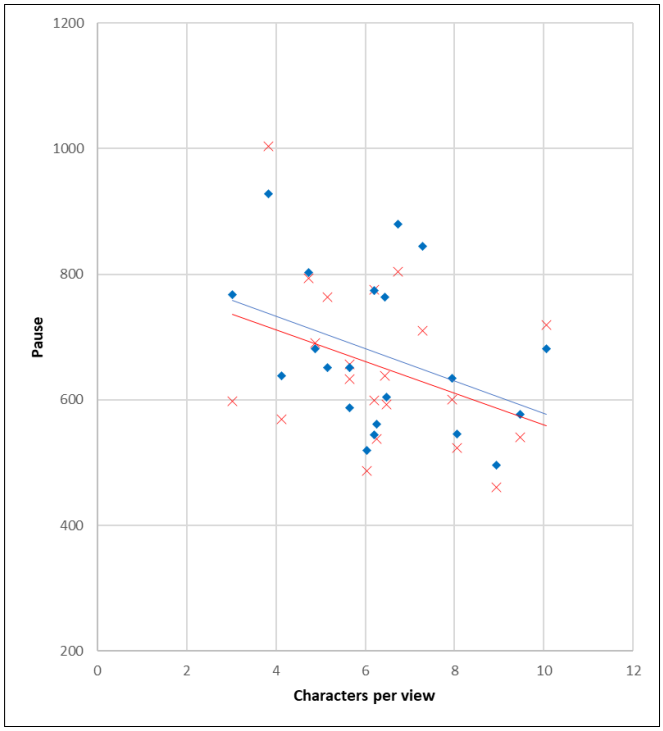

Figure 5.15 Relation of pauses to characters per view (basic (blue) and advanced (red) stimuli)

stimuli name		Basic	Advanced
N		21	21
df		19	19
Writing-times	Constant	0.66	0.66
	Intercept	0.96	0.77
	f	61.33	48.39
	significance f	0.000	0.000
	$R^{2\cdot}$	0.76	0.70
View-times	Constant	0.10	0.18
	Intercept	0.64	0.18
	f	7.46	12.59
	significance f	0.013	0.002
	$R^{2\cdot}$	0.28	0.40
Pauses	Constant	-23.68	-25.11
	Intercept	840.44	811.42
	f	6.32	2.78
	significance f	0.021	0.112
	$R^{2\cdot}$	0.25	0.13

Table 5.6 Relation of writing-times, view-times, and pauses to characters per view

Figure 5.13, Figure 5.14, and Figure 5.15 above demonstrate the linear regression relationship between writing-times, view-times, and Q3 pauses against characters per view for the participants across the basic and advanced stimuli. A simple linear regression was used to see whether writing-times, view-times, and Q3 pauses could estimate characters per view. In Table 5.6 above, the linear regression relationships of writing-times, view-times, and Q3 pauses as functions of characters per view are presented for both stimuli difficulty levels.

In Figure 5.13, Figure 5.14, and Figure 5.15, the blue and red lines for basic and advanced stimuli, respectively, are steeper (greater constant values) for writing-times and $pause_{Q3}$ than view-times, which is consistent with the first and second experiments' findings (Chapters 3 and 4) and supports H6. None of the correlations can be attributed to chance excluding that for the Q3 pauses advanced stimuli. The R^2 values for writing-times are higher – indicating a closer relationship – than for view-times and Q3 pauses. Writing-times are thus considered to be a strong predictor of characters per view in transcription tasks.

Another point to note is that the writing-times for both basic and advanced stimuli predict characters per view at equal strength in the relationship, whereas for view-times, advanced stimuli have a much closer relationship with characters per view than basic stimuli, and vice versa for Q3 pauses.

In summary, writing-times has notably the better R^2 constant and intercept values than view-times and $pause_{Q3}$ against characters per view, as revealed in Table 5.6. Thus, prediction H6 is supported. The relationships' direction is as expected, the absolute magnitudes of the gradients are significant, and the data points follow the linear regression line very well. All of this means that writing-times versus characters per view are the most reliable indicator of the chunking process, as shown in Figure 5.13. Since chunking is the whole principle behind my method of assessing Java competence in freehand transcription, additional proof that chunking is occurring is reassuring.

The subsection that follows goes into further detail about the findings for the analyses (part 2) using my six hypotheses.

5.3.4 Discussion part 2

The results of part 2 are discussed in this subsection, as well as the six hypotheses, H1 to H6, that were discussed in the introduction section of this chapter.

The previous experiment (Chapter 4) revealed that both the view display (VD) measure Q3 pauses and the hide and show (HS) measures characters per view and writing-times can be used to assess Java programming competence in a transcription task. Earlier studies have shown that measurements of pause distribution, in similar types of tasks, seem to reflect chunk structures of participants and hence could be applied to evaluate learners' competence (Cheng, 2014, 2015; van Genuchten & Cheng, 2010; Zulkifli, 2013). This experiment, like the first and second experiments (Chapters 3 and 4), builds on those findings and is similar to the previous experiment (Chapter 4) in terms of experiment design, method, and stimuli. The distinction is

that in this experiment all of the participants were from the same year group/education level, while in previous experiments the participants were from a wider range.

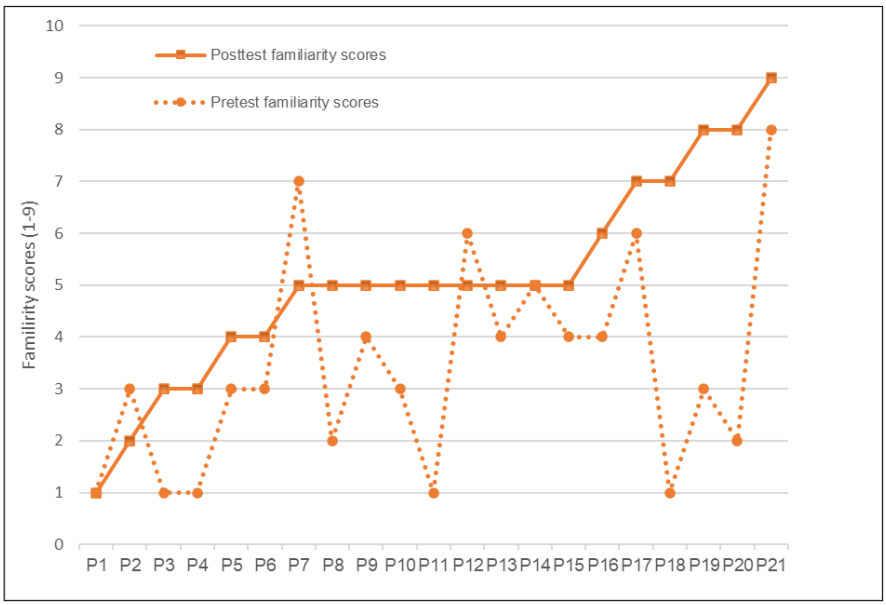

Figure 5.16 Pre-test and post-test familiarity scores for participants, ranked according to post-test familiarity scores

Figure 5.16 above shows the pre-test and post-test familiarity scores for all the students, who are ranked (P1 to P21) according to their post-test familiarity scores. Overall, participants' familiarity scores improved from the pre-test to the post-test, implying that they were more familiar after studying. Only three of the 21 cases have slightly higher familiarity scores from the pre-test to the post-test. However, when the participants performed the experiment for the first time and before studying (pre-test), there is proof that familiarity scores were accurate because they matched up with their Java ratings in the pre-test, as seen in Table 4.2 (Chapter 4).

On the other hand, after studying the Java module (post-test) and repeating the experiment, familiarity is no longer reliable, as shown in Table 5.4. When participants were asked to re-evaluate their familiarity with the same stimulus for the post-test, they were very confused about what precisely to apply their familiarity scores to. They considered various factors, such as: had they come across these code segments of the stimulus before studying the module? Or

did they come across them while working on the Java module? Or maybe both? Participants were confused and unable to provide accurate and precise familiarity scores at this point of the experiment. As a result, familiarity becomes even more unclear, and it no longer serves as an independent measure of this experiment. Furthermore, we might conclude that familiarity as an independent variable only works in a limited number of situations. Luckily, I had a good measure of the participants' Java competence in the form of their final exam marks.

Generally, the linear regression analysis showed that there is a relationship between the independent variable (exam marks) and the behavioural measures, which is stronger for characters per view and $pause_{Q3}$ than it is for writing-times. This experiment confirmed the results of the previous experiment (Chapter 4) in terms of the significant variance in temporal chunk signal HS measure (characters per view) and the VD measure (Q3 pauses) as a function of participants' Java competence. For both experiments, the direction of the relationship of characters per view and Q3 pauses to Java competence, are similar, which matches H1 and H4.

The simple linear regression outcomes shown in Table 5.5 allow me to answer 'Yes' to the question: Will the behavioural measures be effective at detecting the learning that occurred during the Java course?

Hypothesis H1 predicts a positive relationship between characters per view and final exam marks, and hypothesis H2 also predicts a positive relationship between writing-times and final exam marks. Hypothesis H3 predicts that there will be no relationship between view-times and programming competence. The results are consistent with H1 and H3, with minor differences in specifics, and expand the observations of the first and second experiments (Chapters 3 and 4), as well as the work of Albehaijan and Cheng (2019). H2 is weakly supported. Additionally, the effect is weaker in this experiment compared with the second experiment, this could be because the stimuli used for this experiment (after learning) are the same as those used in the second experiment (before learning). But the students have had a full term of learning Java, so the

advanced stimulus is not difficult enough to successfully discriminate them (i.e., the advanced stimuli become much easier). In accordance with chunking theory, the overall findings reveal that participants' temporal chunk signals are determined by their Java chunk structures, as reflected in their final grades. View-times, on the other hand, were not considered to be a sufficient predictor of competence in this experiment. This assumes that any chunking consequences that can occur when viewing a stimulus will be minimised. Characters per view can be regarded, in this experiment, as a measure of Java competence in transcription tasks.

Hypothesis H4 predicts a negative relation between $pause_{Q3}$ and final exam marks. For each participant, Q3 pauses may affected by the participant's Java comprehension level, and thus the participant's chunk structure. The simple linear regression analysis, as shown in Table 5.5, demonstrates that Q3 pauses can distinguish participants' Java competence. As a result, Q3 pauses may have some potentials to be an indicator of Java competence in transcription tasks.

Hypothesis H5 predicts that participants will perform better on basic stimuli than on advanced stimuli, and is supported for characters per view and writing-times and partially supported for Q3 pauses, which is compatible with the previous experiment (Chapter 4). This is to be anticipated because the basic stimulus is intended to be easy, and most participants should be able to transcribe it appropriately. The disparity in participant results between the HS and VD tests may be due to a variety of factors, as mentioned before; this might be because VD (where $pause_{Q3}$ were collected) and HS (where characters per view and writing-times were collected) are two distinct display methods. Also, $pause_{Q3}$ is a finer-grained measure that picks up on every particular piece of punctuation, while characters per view and writing-times are made up of groups of characters. (More justification can be found in Discussion part 1, Chapter 4).

Lastly, hypothesis H6, which focuses on the existence of a clear and significant relationship between writing-times and characters per view (the more time participants spend transcribing,

the more characters per view they will develop), is supported by this experiment as well as the first and second experiments (Chapters 3 and 4). The linear regression analysis for writing-times, view-times, and $pause_{Q3}$ against the characters per view measure proved the following: Figure 5.13 and Table 5.6 show that there is an evident relation between writing-times and characters per view, which supports my base predictions, H1 and H2. On the other hand, as shown in Figure 5.14 and Table 5.6, the relationship between view-times and characters per view is not evident, as it is with the writing-times measure, which supports my third prediction, H3. As presented in Figure 5.15 and Table 5.6, there is a negative relation between $pause_{Q3}$ and characters per view, which supports my fourth prediction H4.

To sum up, predictions H1, H4, and H5, related to characters per view and $pause_{Q3}$, are supported by the independent measure (exam marks). H2 is weakly supported. Additionally, I did not spot any indication that view-times are a suitable measure of competence, which supports my third prediction, H3. Prediction H6 was prompted because, as shown in the results earlier, there is a significant relationship between writing-times and characters per view, which provides further evidence about chunking, thus, support Albehaijan and Cheng (2019) findings.

5.4 Overall discussion

The findings of all three experiments confirm that I am able to answer 'Yes' to the main research question: Can programming competence be measured by analysing patterns of chunk behaviour in a program code transcription task? The outcomes assure that I am also able to answer 'Yes' to this chapter's questions: 1) Will examining the same students after a short period of learning time show a significant difference in their performance, and thus their Java programming competence? The results (part 1) show significant development in the mean scores of all the behavioural measures after learning, thus this experiment reveals a difference in participant performance over a particular time, despite the fact that the extra experience is just three months. 2) Are the behavioural measures still able to demonstrate the differences between the

participants within the experiment? Hypotheses Ha to He and hypotheses H1 to H6 (except H2) are supported, with some divergence in the details.

This experiment's outcomes, generally, supports the findings of both the first and second experiments (Chapters 3 and 4). Table 5.5 above shows that, in this experiment, characters per view and $pause_{Q3}$ may have some potentials as measures of competence, and there is a marginal effect for the writing-times measure. No evidence of a relationship between view-times and programming competence was found in this experiment. When it comes to the stimuli difficulty levels in this experiment, the advanced stimuli differentiate participants' levels better than the basic stimuli. The correlation (Table 5.4) and the regression analysis (Table 5.5) values are not significant for basic but are for advanced, and this may be because all the students improved. Thus, the stimuli were no longer sufficiently difficult on average across all the students, meaning there was no visible difference between basic and advanced as everything had now become basic for all the students.

Both the second (pre-test) and third (post-test) experiments are similar, except for the following differences:

- The aim of the third experiment was to conduct a post-test analysis in order to compare participants' performance after studying the Java module over a three-month period to their performance in the pre-test (previous experiment, Chapter 4). It was discovered that the temporal chunk signals (behavioural measures) would detect substantial changes in student success before and after learning.
- PPCS stimuli contain randomly ordered letters and numbers. These were used in both the second and third experiments, but the difference was in the PPCS stimuli content design. In this experiment, PPCS were written in two ways (quintuple (i.e., S5) and sextuple (i.e., S6)). However, the PPCS in the previous experiment were written in four different forms (single (i.e., S1), double (i.e., S2), triple (i.e., S3), and quadruple (i.e.,

S4)), such as ‘k 6 m 4 I 7 h 2 z …’ and ‘f6 3c 1t w9 4e x5 …’ etc. Notwithstanding, the analysis of PPCS is not included in this chapter. It was found that characters per view, writing-times, view-times, and Q3 pauses are not improved by normalizing using PPCS. Normalizing the PPCS test has very little impact on participants’ temporal chunk signals, which contradicts prediction H6.

- In the previous experiment (Chapter 4), participants had a broader range of educational levels (undergraduate to faculty members). The participants in this experiment were all first-year undergraduate students who had participated in the previous experiment (second experiment, Chapter 4).
- There were 51 participants in the previous experiment, and 21 participants in this experiment.
- The previous experiment showed the differences between the participants’ performance quite well. But when focusing on only the 21 participants from both experiments, the results were not as strong for the relations between the independent variables (either familiarity or final exam marks) and the dependent variables (characters per view, writing-times, view-times, $pause_{Q3}$).
- For the previous experiment, the dependent variables significantly predicted familiarity, whereas familiarity no longer works as an independent variable for this experiment. Thus, we could infer that familiarity as an independent variable is only effective in a narrow range of cases, and we have to be careful when using it as a competence measure.
- In the previous experiment, the independent variable was familiarity scores, and in this experiment, it was final exam scores.

5.4.1 Suggestions and future work

In terms of recommendations for future experiments, if the debate is to be moved forward, a better understanding of familiarity scores needs to be developed and we should be more precise about what we mean by familiarity. Another suggestion is the implementation of Java or programming self-rating scores, such as using a proper Java test before running the experiment e.g., 10 multiple-choice Java questions instead of the familiarity scores. In this experiment I did not have materials to challenge the students at the right level of familiarity; future work could have more sophisticated materials.

5.5 Summary

- It is possible to measure actual learning gains over a short period of time using this thesis methodology; this is especially novel because it is the first time this has been done in such studies.
- Almost all students showed significant differences in mean values before (Pretest) and after (Posttest) learning.
- Familiarity scores may be effective as an independent measure of competence only when the experiment is carried out for the first time.
- Significant relations were found between the dependent measures (Characters per view, writing-times, and pauses) and the independent measure of competence (i.e., final exam marks). However, these relationships are weaker in this experiment, which is to be expected given that all of the students have learned and the same stimuli can no longer differentiate their levels.
- Significant relationships between Characters per view and writing-times were once more spotted, likewise the previous experiments.

6 Discussion and conclusion

Chapter content:

- Introduction which presents the main thesis questions.
- Recap of the chunking theory.
- How this research relates to previous work.
- Main research findings:
    - Does the longitudinal post-test study show learning gain over a short period of learning time?
    - Does the complexity analysis improve the discrimination of the stimuli?
- Discussion of findings, limitations, and implications for future work:
    - Why a freehand transcription (i.e., copying) task?
    - Transcription modes.
    - What kind of stimuli were used?
    - Can the components of the chunks be revealed?
    - What kind of measure is suitable?
        - Extra evidence of chunking
    - Does normalization improve the measures?
    - Novel thesis contributions.
    - Limitations and suggestions for future research.
- Real-world application of the approach and final conclusion
- Summary

6.1 Introduction

The overarching aim of this study was to evaluate whether signals of cognitive chunk structures could be used to reveal cognitive processes occurring in a freehand writing programming transcription task, to serve as measures of Java competence. Put another way:

To develop a novel method that quickly and efficiently assesses programming comprehension by analysing the cognitive chunk structures and behavioural measures

(characters per view, writing-times, view-times, and Q3 pauses) that occur during the activity of Java code transcription.

Throughout this thesis, a number of questions were raised:

1) Can programming competence be measured by analysing patterns of chunk behaviour in the task of program code transcription? This extends chunk-based measures of comprehension to domains beyond mathematics and natural language learning.
2) Is it possible that handwriting a program code could provide powerful and stable temporal chunk signals that can be used to assess programming comprehension?
3) Can programming comprehension be accurately measured using view-numbers/or characters per view, writing-times, view-times, and $pause_{Q3}$, in a freehand writing transcription task?
4) Can characters per view, writing-times, view-times, and Q3 pauses be improved by normalizing using participants' preferred cluster size?
5) Will the behavioural measures be effective at detecting the learning that occurred during a Java course?

These questions were constructed around the goals of this research, which were to explore: (1) an efficient transcription technique, (2) the type of stimuli appropriate to a transcription task, and (3) the probable programming comprehension measures, characters per view, writing-times, view-times, and Q3 pauses.

This chapter begins with a reminder of how the approach used in this thesis taps into chunking theory and how it relates to comparable prior work, followed by a quick summary of the primary results for this thesis. Different aspects of all the experiments are then discussed. The chapter then discusses limitations, implications, and future research contributions. It concludes with a general statement on the discoveries of this thesis.

6.2 Recap of chunking theory

A chunk is a memory unit whose components are highly related to one another but only weakly related to components in other chunks. In general, the chunking process occurs during learning, and it begins to build hierarchies of structures of chunks in memory through learning. These chunk structures in memory can be used for a variety of tasks, including transcription. When participants are about to transcribe a piece of code, they use the chunk hierarchy to recognise what is in the stimulus. And because participants are using chunks, this affects their behaviour, which can be assessed in terms of how often they look at the stimuli (view-numbers), how many characters they have been transcribing in one view (characters per view), and how long they have been transcribing (writing-times), all of which are obtained in the hide and show (HS) presentation mode, or in terms of how long they pause before transcribing any stroke (Q3 pauses), obtained in the view display (VD) mode. These measures were then used to assess Java programming competence, as how high the competence is equivalent to how good the chunk hierarchy is.

The three experiments in this thesis support the underlying idea of chunking as a cognitive process that is used to differentiate high-competence from low-competence participants in various domains, and confirms that high-competence participants create bigger chunks with functional components, with a smaller number of views. This thesis has demonstrated that the chunk structures of each participant offered indications of their programming comprehension level. And the findings confirm the findings of prior research (mentioned in Chapter 2, section 2.2.2) that demonstrated the potential of utilising chunking in evaluating individuals' programming comprehension.

6.3 How this thesis relates to previous work

The work of Cheng (2014, 2015) in the maths field and Zulkifli (2013) in regards to second language learners served as the foundation for this thesis in terms of using pause analysis to measure competence and to distinguish participants' competence levels in various domains. This thesis validates the use of this method in measuring programming comprehension by differentiating participants' Java programming comprehension levels.

This research method, which relies on analysis of chunk signals in freehand transcribed Java code, surpasses other conventional methods of measuring programming comprehension, which use comprehension tasks such as composing, debugging, or modifying program code, by producing more accurate and precise outcomes in a short period of time, saving the student and the instructors (teacher, trainer, marker) time and effort. Previous studies, such as Adelson (1984), Miara et al. (1983), Rambally (1986), and Sarkar (2015), measured programming competence using computer-based techniques and common tasks such as modification, debugging, composition, and so on, whether in educational assessments or empirical studies. Studies such as Soloway and Ehrlich (1984), McKeithen et al. (1981), Barfield (1986), and Shneiderman (1976) were able to identify distinctions between specialists and beginners in programming via recall tasks and programming knowledge structures, whereas this thesis aimed to differentiate programmers' competence levels via a freehand transcription (copying) task and determining chunk structure signals. Despite the fact that time was utilised as a metric in that research, their time frame is relatively long (i.e., minutes). As a result, while their notion of utilising time to assess programming comprehension is beneficial, it is not so effective for the purposes of this thesis, since the pauses are considerably shorter. Pauses in research more similar to this, such as Cheng (2015) and Zulkifli (2013), range from 100 milliseconds to a few seconds.

6.4 Main findings

This thesis primarily demonstrated that individuals' chunk structure signals have potential as a method of assessing Java programming comprehension in a freehand transcription task. In particular, view-numbers or characters per view, writing-times, and pauses have some potential in measuring competence. However, view-times do not (as expected).

Dependant V	Transcription mode	1st Experiment					2nd Experiment					3rd Experiment				
		N	Familirity (Independent V)				N	Familirity (Independent V)				N	Exam marks (Independent V)			
			Basic		Advance			Basic		Advance			Basic		Advance	
			R^2	*P*	R^2	*P*		R^2	*P*	R^2	*P*		R^2	*P*	R^2	*P*
View-numbers	HS	24	0.60	0.000	0.46	0.000	51					21				
Char per View								0.21	0.001	0.13	0.009		0.07	0.236	0.21	0.039
Writng-times			0.47	0.001	0.32	0.231		0.12	0.015	0.05	0.137		0.04	0.413	0.14	0.090
Q3 pause	VD							0.23	0.000	0.25	0.000		0.02	0.536	0.24	0.025
View-times	HS		0.21	0.000	0.08	0.000		0.00	0.762	0.00	0.762		0.09	0.198	0.08	0.217

Table 6.1 Relationship of view-numbers, characters per view, writing-times, pauses, and view-times to familiarity and exam marks

Table 6.1 above summarises the relationships between the dependent variables and the independent variables, across all the three experiments and across the two levels of stimuli difficulty, by presenting the R^2 and significance values. For the first experiment, only the HS transcription task (which provides the view-numbers or characters per view, writing-times, and view-times measures) was addressed, but for the second and third experiments, both HS and VD (Q3 pauses measure) tasks were considered. Concerning the independent variables, familiarity scores were calculated for all three experiments, whilst final exam marks were also used for the third experiment. In terms of the stimuli, the same stimuli were applied for both the second and third experiments, as the main purpose of the third experiment was to assess whether students' learning over a short period of time (three months) can be measured.

Overall, the R^2 of view-numbers, characters per view, writing-times, and pauses were significant in the first and second experiments. On the other hand, despite the fact that the students' familiarity scores increased after learning (third experiment), the dependent variables for the third experiment were more weakly related to the independent measure of competence,

although, as in Table 6.1 above, characters per view and Q3 pauses significantly predict final exam marks for the advanced stimuli.

Cohen (1992) pointed out that R^2 values (goodness of fit values) are considered low when they are 0.12 or lower, medium when they are between 0.13 to 0.25, and high when they are 0.26 or higher. For view-numbers (or characters per view), writing-times, and Q3 pauses results, the R^2 values are stronger for the first experiment than for both the second and third experiments. To illustrate, the R^2 values for the second experiment decreased, which is not surprising given that the second experiment had a larger number of participants, the majority of whom had a narrower range of Java experience. As a result, the amount of variation among all participants was reduced. Why did I choose to administer the second and third experiments to students with a narrower range of abilities? The first experiment demonstrated the applicability of the approach to the programming domain, while the second and third experiments applied the approach to more realistic samples with a range of ability similar to that which would be found in real tests.

The R^2 values decreased for the third experiment basic stimuli (as expected) but increased for the advanced, for various reasons; all the students were from the same year group, and all of them had learned for the same period of time (details about the improved results are discussed below), thus, the basic stimuli became easy for everyone after learning. As a result, distinguishing between student levels became more difficult. For the advanced stimuli, the R^2 values increased, as expected, indicating that the students had learned. As a result, the advanced stimuli became more effective in differentiating their levels of competence after learning than when they were used before learning (second experiment).

Across the experiments, for the first experiment, view-number (or characters per view) and writing-times have the stronger R^2 value. For the second and third experiments, characters per view and Q3 pauses have stronger R^2 values than writing-times. View-times have low values (as

expected), as the purpose of implementing view-times was to give more support to the chunking explanation. It is apparent from all three experiments that view-numbers (characters per view), writing-times, and pauses have the potential to predict Java programming competence. These findings are consistent across all the behavioural measures, as they point in the same directions across the three experiments (i.e., they consistently distinguish participants' levels of Java competence).

The results overall support the main thesis hypotheses: (1) More competent participants have a smaller number of views of the stimulus because their chunk size is bigger. (1a) The number of characters per view increases after learning. (2) For more competent participants, the duration of written answers after each stimulus view is longer because their chunks are bigger and each character takes the same time to write. (2a) The duration of written answers following each stimulus view is longer after learning. (3) The amount of time spent on each individual view of the stimulus is not directly related to competence. (3a) The amount of time spent on each individual view of the stimulus is unrelated to learning gain. (4) Pause duration before beginning writing each stroke is shorter for more competent participants. (4a) Pause duration is shorter after learning. (5) Performance on simple stimuli is superior to performance on advanced stimuli, with fewer view-numbers, longer writing-times, and shorter pauses, but there is no effect on view-time. (5a) Participants' performance is better on the basic stimuli than on the advanced after learning. (6) Writing-times are directly related to characters per view; the longer participants spent transcribing, the more characters per view they produce. On the other hand, the results did not support one of my hypotheses: that behavioural measures would improve after normalizing them for the PPCS measures (justification is presented below).

To sum up, the results indicate that view-numbers, characters per view, writing-times, and pauses have potential as indicators of Java competence, extending the findings of previous work in maths (Cheng 2014, 2015) and second language learning (Zulkifli, 2013). Further details

regarding the independent and dependent variables, as well as the best-suited measure, are given below.

6.4.1 Does the longitudinal post-test study show learning gain over a short period of learning time?

Temporal chunk signals have been used to measure competence in previous studies (such as Cheng, 2014, 2015; van Genuchten & Cheng, 2010; Zulkifli, 2013), and this thesis's first and second experiments. Static levels of competence were used and participants came from a diverse range of backgrounds. However, the third experiment in this thesis (Chapter 5) aimed to assess learning gains in the same students who took part in the previous test (second experiment, Chapter 4). This thesis's third experiment is the first to show changes in chunk structures over a short period of learning time (i.e., from the start to the end of a single Java module), which equates to only three months of additional experience.

Measures	HS mesures												VD measure			
	Basic		Advanced		Basic		Advanced		Basic		Advanced		Basic		Advanced	
	Char per view Posttest	Char per view Pretest	Char per view Posttest	Char per view Pretest	Writng-times Posttest	Writng-times Pretest	Writng-times Posttest	Writng-times Pretest	View-times Posttest	View-times Pretest	View-times Posttest	View-times Pretest	Pauses Posttest	Pauses Pretest	Pauses Posttest	Pauses Pretest
Mean	7.07	5.26	6.34	5.38	5.64	4.71	4.94	4.53	1.36	1.42	1.33	1.34	673.04	778.15	652.21	745.67
Change percentage	34.4%		17.8%		19.7%		9.1%		-4.2%		-0.7%		-13.5%		-12.5%	

Table 6.2 Mean post-test and pre-test values and the percentage of change in the values of behavioural measures across all the behavioural measures over basic and advanced stimuli

Table 6.2 above presents the mean values for all of the HS and VD behavioural measures for both basic and advanced stimuli before (pre-test, in the second experiment) and after (post-test, in the third experiment) learning on the Java course. The table shows the percentage change in behavioural measure values after completing the Java course. The study showed significant differences in participants' behaviour due to learning for a short period of time, which is clear from the percentage of change in the behavioural measures. Put another way, the experiment outcomes show significant progress in the mean scores of characters per view, writing-times, and $pause_{Q3}$ for all the students after learning, with stronger values for characters per view.

Again, these findings reassuringly support the previous experiment's results (before learning) because the behavioural measures show consistent patterns across both experiments. Characters per view and writing-times increased with an increase of competence, while pause duration decreased. The results also reassure that view-times do not suggest a difference in participant outcomes before and after learning.

6.4.2 Does the complexity analysis improve the discrimination of the stimuli?

After analysing the second experiment's data, a complexity analysis was conducted (section 4.3.3.). The investigation was carried out to determine whether the complexity of the stimulus is an extra factor that needs to be addressed when designing the stimuli. Might it enhance the disparity between the levels of stimuli complexity? This analysis concentrated on analysing the experimental stimuli based on the complexity factor by evaluating the proportion of various stimulus components.

Behavioural measures	Main analysis		Complexity analysis			
			Basic		Advanced	
	Basic	Advanced	(simple)	(complex)	(simple)	(complex)
Charters per view	5.5	5.4	6.4	4.6	5.2	5.6
Writing-times	5.3	4.9	6.1	4.6	5.0	5.0
View-times	1.6	1.5	1.7	1.6	1.6	1.5
Q3 pause	748	735	646	899	678	756

Table 6.3 Mean values of all the behavioural measures across all participants, over different stimuli difficulty (basic, advanced) and complexity (simple, complex) levels

Table 6.3 above compares the mean values of the number of characters per view, the writing durations, the viewing stimulus durations, and the pause durations between different levels of stimuli difficulty (basic and advanced) and complexity (basic: simple and complex; advanced: simple and complex). I observed that when the complexity factor is considered, the difference between the stimulus complexity levels increases for all behavioural measurements. Q3 pauses, in particular, show a marked increase in differentiation. Characters per view change more than

writing-times, while view-times are constant. This analysis shows a distinct discrimination between the two complexity stimuli (simple and complex), with the basic category exhibiting the highest discrimination and improvements in mean values across all behavioural measures, as predicted by the stimuli complexity analysis (Table 4.5 in Chapter 4).

The results show that the complexity factor has an influence, implying that when developing stimuli, it is important to take into account both the stimuli difficulty level (according to the Java curriculum) and the stimuli complexity level. Finally, the Q3 pauses show the highest enhancement in mean values of all the behavioural measures, which may be due to the VD measure being better suited to the unique characteristics of the transcription task.

6.5 Discussion of findings, limitations, and implications for future work

6.5.1 Why a freehand transcription (i.e., copying) task?

The reason for implementing the freehand transcription task for this thesis was that transcription simplifies and clarifies cognitive processes, making participants' knowledge visible and able to be measured, whereas other programming tasks, such as composing program code, may involve editing the code (i.e., rewrite, delete, debug, etc.), which will undoubtedly generate noise that will negatively impact the programming comprehension measure. As a result, the participants' programming knowledge may be hidden, making it difficult to precisely identify chunk structures.

Freehand transcription was chosen over typing in order to get a better method of measuring competence. The focus in this thesis was on using a more precise time scale (milliseconds), which is only achievable with freehand writing, and which offers richer and more distinguishable chunk signal data than keyboard typing (Cheng & Rojas-Anaya, 2005; Zulkifli, 2013). And as this thesis targeted participants who were educated adults, freehand writing (i.e., transcribing) is a

fundamental skill for them. On the other hand, the results obtained through typing would be influenced by the different participants' typing abilities and thus would be imprecise.

To summarise, freehand transcription was considered an adequate, effective, and simple writing assignment that could be used to evaluate cognitive processes due to the consistency of data it provides. In addition, the transcription method used in this thesis is a novel method that has not previously been used in the programming domain.

6.5.2 Transcription modes

The research demonstrates that in order to obtain the primary measures of this thesis, both display techniques (i.e., transcription modes), view display (VD), and hide and show (HS), can be used. The HS transcription mode was used to obtain view-numbers, characters per view, writing-times, and view-times, where the stimulus is only visible when a special button is pressed. To obtain the Q3 pauses measure values, the VD transcription mode was used, in which the stimulus is always visible. Thus, throughout this series of experiments, I was able to successfully assess participants' Java programming comprehension by using both HS and VD measures.

In both transcription modes, participants transcribed each character (letter, number, or punctuation) in a separate box within the specially designed response grid. The study confirms that this method of transcription is beneficial, as strong correlations were found across the experiments. This method was chosen because it produced successful outcomes for Cheng and Rojas-Anaya (2004, 2005, 2006, 2007), who established the method first, and Zulkifli (2013), who successfully followed their approach. While putting each character in a distinct box may seem uncontrollable, the impact is constant for each character, so it should have no effect on the overall data. Besides, participants quickly adjust to this method of transcribing, and it does not appear to have a negative impact on their performance (Cheng, 2014; Zulkifli, 2013).

6.5.3 What kind of stimuli were used?

As the target population was from the Department of Informatics at the University of Sussex, and Java is the base programming language for almost all undergraduate modules, undergraduate Java modules played a significant role in deciding on the stimuli content for all of the experiments. Two pairs of basic and advanced versions of the stimuli were used. The syntax in the basic stimuli was a critical component of their first year of Java instruction. The advanced stimuli syntax was more specialised and comes from advanced Java modules. Following the analysis of the first experiment's data, a content analysis was performed, which revealed that, in general, participants choose to return to the stimulus (i.e., break) more frequently before transcribing three elements: punctuation marks, capitals within variables or reserved words, and spaces. This was taken into account while developing the second experiment's stimuli, which involved increasing the density of the syntax (i.e., punctuation, variables with capital letters in the middle, and spaces). After analysing the second experiment's data, it was discovered that not only is the stimuli difficulty factor important, but also the complexity factor. *PunctuationALL* (i.e., summation of brackets and punctuation marks) and *TextAll* (summation of reserved words and variable names) were compared, then the stimuli (both basic and advanced) were further broken into simple and complex. When the complexity factor was considered, it distinguished the students' behaviour between simple and complex stimuli much more clearly.

Various aspects were considered when designing the stimuli (as discussed in Chapter 2), such as the use of indentation and modularity (where each module has a clear function), which aided in differentiating programming comprehension levels. In order to achieve a high level of experimental control, I decided to use small code fragments. Each fragment consisted of a specific number of code lines. Because the response sheet is made up of distinct small boxes, needless spaces between code phrases were eliminated to minimise uncertainty when transcribing these unnecessary spaces. After obtaining the content analysis findings (Chapter 3),

further stimulus design decisions were made for the subsequent experiments, such as focusing on increasing the syntactic intensity of the code fragments and shortening the variable names. Finally, after conducting three experiments for this thesis, it was revealed that adopting the previously mentioned aspects aided in meeting the thesis's major aim of differentiating high-competence participants from low-competence participants. Focusing on the program code complexity level (by increasing the quantity of punctuation such as symbols and brackets) aided in creating two clearly separate stimuli difficulty levels. Via all the above, participants' competence levels were clearly differentiated, which is concluded via obtaining clear and precise chunk structure signals.

Prior research indicates that when utilising scrambled versions, beginners and professionals behave similarly; Zulkifli (2013), who utilised the same approach (i.e., pause analysis during copying) to assess second language learners' competence, and Shneiderman (1976) discovered no benefit to utilising random/scrambled versions. As the primary goal of this thesis was to discover a strategy that clearly distinguishes professionals from beginners, prior work suggested that employing scrambled program code versions in this thesis would have been pointless.

6.5.4 Can the components of the chunks be revealed?

At the end of the first experiment (Chapter 3), a content analysis was performed to provide additional evidence that chunking had occurred, and to state that chunking does not happen at arbitrary positions, but rather at the boundaries between things that are meaningful. Thus, I could determine the exact content of each chunk.

This analysis revealed a set of results, mainly that there is a negative relation between participants' programming familiarity and the number of breaks they take (i.e., number of views of the stimulus). The results imply that, firstly, the breaks are more associated with punctuation marks than reserved and variable words. Thus, the greater number of breaks that accompany punctuation, the less competent participants are. Secondly, the more breaks associated with

capital letters within reserved and variable words, the lower participants' competence. For example, if a capital appears within a variable or reserved word, low-competence participants write it as two words because they recognise it as two distinct words, whereas high-competence participants identify it as one word. Thirdly, low-competence participants produced more space-related breaks than highly competent participants. For example, a high-competence participant can get three words as one chunk rather than three separate chunks, while a low-competence participant gets them as three sperate words. On the other hand, for all the participants, there is a weak relation between familiarity and the number of breaks in reserved and variable words, and in the strokes within each character, as shown in Table 3.6. All of this implied that when designing the upcoming experiment stimuli, the emphasis should be on syntax density by increasing the number of punctuation marks and capitals inside variable names, and on attempting to minimise the number of variable names. As a result, the levels of programming comprehension will be better distinguished between low- and high-competence groups.

To summarise, the chunks effect was clearly visible in the punctuation marks, capital letters within variable and reserved words, and allocated spaces in the program code. Views (i.e., breaks), on the other hand, were not displayed within reserved and variable words or in strokes within each character. Consequently, it might be possible to use this kind of analysis to supplement the overall measures of competence. For example, knowing what characters have been written allows you to select pauses that are likely to be associated with the beginning of chunks (i.e., punctuation marks or the first letter of specific reserved words). In other words, knowing what is likely to be at the start of each new chunk, and thus knowing where the pauses are likely to be longer. So, rather than using the pauses' third quartile, an interesting line for future research could be to use this type of data.

6.5.5 What kind of measure is suitable?

Throughout the experiments, various independent and dependent variables were utilised.

Firstly, familiarity scores and final exam scores (third experiment) were used as independent variables. The behavioural measures significantly predicted familiarity scores in both the first and second experiments but not in the third. To demonstrate, familiarity as an independent measure performed well when the participants performed the experiment for the first time and before taking the Java module (first and second experiments). However, for the third experiment (after taking the Java module) the participants' experience had generally increased (i.e., participants' competence levels were more similar), so there was much less to differentiate their levels of competence. To illustrate, in the third experiment, when participants were asked to re-evaluate their experience with the identical stimuli, I suspect they became quite confused about what to attribute to familiarity to. Thus, it may be concluded that familiarity is only effective in a limited number of situations. However, the Java module results show a difference between student levels of competence; as a result, student exam scores were used as an independent variable in the third experiment.

Secondly, characters per view (view-numbers in first experiment), writing-times, view-times, and $pause_{Q3}$ were used as dependent variables calculated from each participant's logs. These variables were evaluated for all experiments with the exception of $pause_{Q3}$ in the first experiment, which was not recorded due to a technical error. Across all three experiments, view-numbers and characters per view had a stronger relation with the independent variables and better closeness of fit of participants to the independent variables than writing-times. The overall results, on the other hand, show that there is no relationship between the view-times measure and the independent variables, and that the participants showed a poor fit to the independent measure regression lines. In most cases, the Q3 pauses measure had a stronger relationship with the independent measures and a better fit of participants to the independent variable regression lines than the HS measures, writing-times and characters per view.

In summary, the overall results for characters per view, writing-times, view-times, and $pause_{Q3}$ are consistent with the predictions, but with some divergence in details. Characters per view and Q3 pauses are recognized as more reliable predictors of Java competence, but in most cases, Q3 pauses is a slightly better predictor of Java competence than characters per view.

6.5.5.1 *Extra evidence of chunking*

The significant negative relationship between writing-times and view-numbers in the first experiment, and the significant positive relationship between writing-times and characters per view in the second and third experiments, confirm the chunk-based assumptions underpinning the thesis predictions, which involve a decrease in view-numbers and an increase in characters per view and writing-times as Java familiarity scores grow. Thus, these strong relationships, across the three experiments, serve as extra evidence of chunking.

View-numbers (or characters per view) and writing-times were both anticipated to be dependent on chunking procedures; hence, they should have some type of stable and formal relation. And as the whole theory behind my method of measuring competence is 'chunking', so having some extra evidence that chunking is happening (writing-times vs view-numbers and writing-times vs characters per view) is reassuring.

6.5.6 Does normalization improve the measures?

Concerns may arise regarding whether the effect of individuals' working memory size may hide any competence participants have got. Thus, throughout this research, I sought to improve the behavioural measures values via normalizing them using the practise stimuli (first experiment) and the PPCS stimuli (second experiment). The PPCS test can be thought of as a working memory size (chunk size) test, as it was used here to determine the size of the chunks that the participants used for transcribing the stimuli. However, the normalization process did not enhance the behavioural measures' values. Furthermore, not only did the PPCS normalization (second experiment) not improve the values of the behavioural measures, the practice items

(first experiment) did not either. This may be due to the fact that transcribing the PPCS stimuli is not very important when it comes to the transcription task, and/or the individuals' variations in characters per view, writing-times, and $pause_{Q3}$ that arose using the practise and PPCS items may not be very large in comparison to the differences when actually transcribing the Java stimulus. Normalization improved the behavioural measures in the similar work of Cheng (2014, 2015), measuring maths competence using algebra equations. Cheng normalizes using very simple arithmetic equation materials. But in this thesis, PPCS (arbitrary letters and numbers) was implemented, which did not improve the behavioural measures of Java competence. This is not surprising, because copying simple equations may be affected by individual experience, while copying PPCS is not affected by programming experience.

In conclusion, the normalisation performed in this thesis provides additional evidence that it is worthwhile to normalise materials that include the fundamental aspects of the topic being measured. As a result, this is yet another piece of indirect evidence that it is not worthwhile to try to normalise for factors such as working memory size, writing speed, or writing ability. Another important conclusion is that the lack of improvement in the behavioural measure values after normalisation confirms that my method measures individuals' programming comprehension and learning gained rather than individuals' working memory capacity.

6.5.7 Novel contributions

This thesis's novel contributions compared to previous work on micro-behavioural temporal chunk signals are:

1. The strategy can be applied to programming discipline in addition to maths and language learning.
2. In contrast to the single-line stimuli used in previous related micro-behavioural temporal chunk signals studies, this thesis's stimuli were larger (nine lines). Because there is more data per trial, single trials can provide significant useable correlations

with competence without the theoretical difficulties of choosing how to aggregate data from numerous trials or the practical difficulties of switching between multiple trials.

3. This is the first application of the content analysis, where the identity of each stoke was discovered, in the use of temporal chunk signals and chunk structure to measure competence in freehand transcription task.
4. There is another complementary method (rather than using pauses) of obtaining a chunk-based measure of competence, which is to use the characters per view (or view-numbers) and writing-times (HS) measures. The correlations reported for the HS measures are at the same level as the correlations identified for pauses in prior studies (Cheng, 2014; van Genuchten & Cheng, 2010; Zulkifli, 2013).
5. The longitudinal study (i.e., the third experiment) to demonstrate learning gain over a short period of time is thought to be the first to be used in this type of study (compared to e.g., Cheng, 2014, 2015; van Genuchten & Cheng, 2010; Zulkifli, 2013, where participants came from a wide range of backgrounds, and only static levels of competence were evaluated).

Novel contributions compared to various types of programming assessments studies are:

1. This is the first known study to use transcription (i.e., copying) as a task to measure individuals' levels of programming competence.
2. Some studies have used response times to study programming comprehension in whole tasks, such as sets of multiple-choice questions, lasting minutes (e.g., Adelson, 1981, 1984; Ye & Salvendy, 1996). For this thesis, the focus was on the time required for component activities within a task, rather than overall task time, and the examination of process durations that may be directly related to the chunks possessed by participants.

3. The pause analysis technique is used for the first time as a measure to assess individuals' programming comprehension.

6.5.8 Limitations and suggestions for future research.

The experiment series reported in this thesis focused on examining view-numbers or characters per view, writing-times, view-times, and $pause_{Q3}$ to cognitively evaluate programming comprehension.

The research constraints and suggestions for future works are:

1) The use of a graphical tablet (i.e., non-standard equipment) in this thesis is regarded as a practical limitation that can be overcome in future work by employing standard equipment such as computer desktop interfaces, as in the work of Ismail and Cheng (2021), which uses a standard computer interface with a mouse to measure English language competence, attempting to use the same chunking theory and utilizing temporal chunk signals.
2) Using a transcribing task may be considered a limitation because it is an artificial task to be used in such a rich domain as programming, which involves problem solving, debugging, and designing. This method of competency assessment does not, however, focus on this skills-oriented aspect of programming. It focuses on the knowledge structure of program code in particular.
3) This thesis utilized a specially designed response grid and an unnatural non-cursive mode of transcribing each character in a sperate box, in order to prevent cursive transcribing and to enable me to easily differentiate words, letters, and punctuation pauses. Future work could include experimenting with cursive writing, which will only allow for pauses between words.

4) My sample categorisation is limited to high- and low-competent groups, whereas in future work, other detailed categorisations, such as naive, novice, intermediate, and expert, can be considered.

5) The programming language used in this thesis was Java. Future research could look into other programming languages where the structure of chunks can be reflected in the structure of written programming notations, implying that my micro-behaviour transcription task method could work for such languages. Will programming languages with more complex syntax be more amenable to this approach?

6) Only a limited range of Java concepts were included here (as this thesis's stimuli design relied on the undergraduate programming curriculum). It would be interesting to see whether other concepts in the language might reveal chunk structures more strongly in future work.

7) For this thesis, I used content analysis after the first experiment to determine where each participant in each stimulus decided to view the stimulus again. This additional analysis revealed that the views were more closely associated with punctuation, capitalization within words, and spaces. Additional research may be conducted to match participants' pauses to specific characters. This could potentially enhance the precision of the measures, as they would be associated with specific bits of code that could be related to familiarity.

8) In terms of the stimuli design space, stimuli manipulations that might impact the chunking process for participants with different levels of competence were applied. For instance, the use of indentation and modularity provides an advantage to those participants who can use their chunks, and this could be manipulated in future research. The literature, on the other hand, revealed that the use of code scrambling, flowcharts, and similar syntax within the same code fragment demonstrates that participants with varying levels of competence act in the same way, hence, it is recommended that they be avoided. A key

point for future research would be to explore stimuli manipulations that might benefit high-competence participants *differentially* compared with low-competence participants.

9) This study was limited to the independent post-tests (online questionnaires) that participants completed after running the experiments, which concern education levels, general programming, Java, and familiarity scores. The fifth variable is the final exam score, which was used in the third experiment. It is recommended that in future research, other tests, such as a pre-test for Java or programming knowledge in general before launching the experiment, be performed to point out other types of programming components that are not involved in this thesis. Furthermore, it is recommended to implement more than one independent test (pre-programming test and post-programming test), as in the third experiment.
10) Only basic and advanced programming knowledge categories were used in this thesis. But there are different kinds of knowledge associated with programming, such as text-structure and plan knowledge, explained in Chapter 2, and future work should look at these various other types of programming knowledge.
11) Because my method of measuring programming competence is based primarily on reading and writing, it is considered limited if applied to people with learning disabilities such as dyslexia (reading and language disorder) and Dysgraphia (writing difficulties). If they will be considered, this will necessitate additional research and consideration of three major aspects: (1) how the visual processing of the stimulus affects the transcriptional encoding (i.e., viewing and reading the stimuli). (2) How any disabilities may affect memory use (e.g., memory retention, mental processing (chunk building)). (3) How it might affect the actual writing process (i.e., motor processing).

The aggregate findings of this thesis show that the implementation of temporal chunk structure signals has potential in measuring the cognitive processes of Java programming.

6.6 Real-world application of the approach and final conclusion

The applications of this thesis's methodology in education are easily anticipated, especially given the success of the longitudinal post-test research (third experiment), which makes use of relative simplicity, short trial times, and the possibility of fully automated scoring. Simply put, such transcription assignments might be given as part of summative end-of-course assessments or as independent screening assessments at the beginning of a course. More interestingly, with appropriately designed test items, the approach might be used as a form of formative assessment to provide tutors with information about individuals' growing understanding of targeted programming concepts.

Another interesting broader application of this thesis methodology (i.e., use the analysis of temporal chunk signals and chunk structure to measure competence via transcription task) is to use it as a possible diagnostic tool for some categories of learning disabilities/disorders such as dyslexia and Dysgraphia. It will be interesting to see how the writing process differs between people with and without learning disabilities. For example, the participants could be asked to transcribe the PPCS stimuli. This could provide a more detailed understanding of the process that distinguishes people with and without learning disabilities.

This thesis sought to build a novel technique for measuring programming comprehension via assessing participants' chunk structures. It manipulated the appropriate transcription methods, suitable program code fragments, and three appropriate measures for assessing participants' cognitive chunk structures. It has demonstrated that there is authentic potential for measuring programming competence using view-numbers, writing-times, and pause analysis.

The thesis finishes with a number of unresolved concerns that require additional examination in order to fully assess the efficiency of the methodology. It creates opportunities for future study to build on the findings and develop a more conclusive technique, utilising a larger group of participants or a broader variety of stimuli, programming languages, and concepts.

This thesis has demonstrated the utility of my novel method for assessing programming competence. In order to improve the technique, further study will undoubtedly be required.

6.7 Summary

To summarise my findings from the three experiments:

- In a free handwriting transcription task, temporal chunk signals have the potential to predict Java programming competence.
- The results are consistent across all the behavioural measures.
- The findings, with minor differences in details, support the overall hypothesis.

Appendices

Stimuli for practice items in Chapters 3, 4, and 5

No.	Stimulus content
1	# Sussex University United Kingdom Informatics Department Engineering Department
2	# Computer Science Programming Course Java Programming Language Pascal Programming Language

Stimuli for PPCS in Chapter 4

Stimulus name	Stimulus content
S1 (HS)	# 9 g 2 b 6 d 7 f 5 w 1 c 8 x 3 h 6 n 4 e 8 m 1 d 5 w 9 j 3 b 8 z 2 h 7 i 4 m 6 k
S1 (VD)	# k 6 m 4 i 7 h 2 z 8 b 3 j 9 w 5 d 1 m 8 e 4 n 6 h 3 x 8 c 1 w 5 f 7 d 6 b 2 g 9

Questionnaire for Chapter 3

Participant No.: Participant code: Date:

Introduction and biographical information (text questions):

1. How old are you?
   ……
2. What is your gender? (Male – Female – other ……)
3. In which group of the following you are? (1st year – 2nd year – final year – MSc – PhD/member of faculty)
4. If you are a student, do you do programming activities which are not part of your study? If yes, for how many years?
   ……

Education (yes/no responses):

1. Did you learn more than 3 programming languages?
   ……
2. Did you take more than 3 programming courses in which you were required to implement source code?
   ……

General programming background (yes/no responses):

1- I am familiar with all the following programming concepts: declaring variables, conditions and iterations, and printing out a sentence on the screen
   ……
2- I can develop programs using procedural languages only and not object-oriented
   ……
3- I can develop programs using at least one object-oriented programming languages
   ……
4- I can develop programs using more than one object-oriented programming language
   ……

5- I develop long programs (100+s lines of code) using object-oriented programming languages such as Java, python, Ruby, and C++ away from my study or work

……

Java (yes/no questions)

1- I am familiar with the 'main' method in Java

……

2- I am familiar with both objects and classes in Java

……

3- I can implement all the following Java object-oriented concepts: encapsulation, inheritance, and polymorphism

……

4- I am familiar with using GUI in Java

……

5- I develop programs utilizing all I/O (streams, readers, writers, files) in Java

……

6- I am familiar with both Java's API (Application Programming Interface) and Annotations

……

7- I have used both data structures and recursion in Java

……

8- I am familiar with database programming in Java

……

(Text questions)

9- Approximately, how many lines of code have you implemented using Java?
Less than 100
Less than 1000
Less than 5000
More than 10000

10- For how many years have you been programming using Java?

Participant No.: Participant code: Date:

Before you started the experiment, how familiar would think you were with these statements.

Stimuli	Stimuli content	Very unfamiliar	Unfamiliar	Neutral	Familiar	Very familiar
S1 A	`public class Person{ private String name; private int age;}`					
S2 A	`public class Person{ public String name; public int age;}`					
S1 B	`public int Balance(){ int amountRefund; return amountRefund;}`					
S2 B	`public void Balance(){ System.out.println("#"); Total += balance;}`					
S1 C	`for(int h=0;h<hCount.length;h++) { System.out.println(h+hCount[h]);}`					
S2 C	`int h=0; while(h<hCount.length){ System.out.println(h+hCount[h]);h++;}`					
D1 A	`<body> Hello! The time is now <%=new java.util.Date()%> </body>`					
D2 A	`<body> <%! private static boolean visited = false; %> </body>`					
D1 B	`int[] numbers= {1,1,3,5,8,13}; for(int item:numbers)`					
D2 B	`int summation=0; for(int counter:arrayTall) summation+=;`					
D1 C	`cont=f.getContentPane(); button=new JButton("Yes"); cont.add(button);`					
D2 C	`Font f=new Font("S",Font.PLAIN,6); g.setFont(f); g.drawString("T",1,9);`					

Questionnaire for Chapter 4

Participant No.: Participant code: Date:

Introduction and biographical information (text questions):

1. How old are you?
   ……
2. What is your gender? (Male – Female – other ……)
3. In which group of the following you are? (1st year – 2nd year – final year – MSc – PhD/member of faculty)
4. If you are a student, do you do programming activities which are not part of your study? If yes, for how many years?
   ……

Education (yes/no questions):

1. Did you learn more than 3 programming languages?
   ……
2. Did you take more than 3 programming courses in which you were required to implement source code?
   ……

General programming background (yes/no responses):

1. I am familiar with all the following programming concepts: declaring variables, conditions and iterations and printing out a sentence on the screen
   ……
2. I can develop programs using procedural languages only and not object-oriented
   ……
3. I can develop programs using more than one object-oriented programming language
   ……
4. I develop long programs (100+ lines of code) using object-oriented programming languages such as Java, python, Ruby, and C++ away from my study or work
   ……

Java programming language (yes/no responses):

1. I am familiar with the ‘main’ method in Java

……

2. I am familiar with both objects and classes in Java

……

3. I can implement all the following Java object-oriented concepts: encapsulation, inheritance, and polymorphism

……

4. I am familiar with using GUI in Java

……

5. I develop programs utilizing all I/O (streams, readers, writers, files) in Java

……

6. I am familiar with both Java’s API (Application Programming Interface) and Annotations

……

7. I have used both data structures and recursion in Java

……

8. I am familiar with database programming in Java

……

Participant No.: Participant code: Date:

Before you started the experiment, how familiar would you say you were with these statements?

Stimuli	Stimuli content	Very unfamiliar	Unfamiliar	Neutral	Familiar	Very familiar

S1 A	`import java.util.Scanner;`
	`class A{`
	`public static void main(String[]args){`
	`Scanner s=new Scanner(System.in);`
	`double l=s.nextDouble();}}`					

S2 A	`import java.util.Scanner;`
	`class S{`
	`public static void main(String[]args){`
	`Scanner b=new Scanner(System.in);`
	`int n=b.nextInt();}}`					

S1 B	`int n,i,j;`
	`int a[]=new int[n];`
	`for(i=0;i<(n-1);i++){`
	`for(j=0;j<n-i-1;j++)}}`					

S2 B	`int num,i,j,tem;`
	`if(ar[j]>ar[j+1]){`
	`tem=ar[j];`
	`ar[j]=ar[j+1];}`					

D1 A	`InputStream is=null;`
	`Try{`
	`is=new FileInputStream(f);`
	`byte c[]=new byte[2*1024];`					

D2 A	`try{`
	`Files.createFile(f);}`
	`catch(FileAlreadyExistsException x){`
	`System.err.format("n",f);}`					

D1 B	`Node<T>n=new Node<T>();`
	`n.setValue(i);`
	`n.setNext(f);`
	`if(f!=null)f.setPrev(n);`
	`if(f==null)r=n;`					

D2 B	`Node<T>d=new Node<T>();`
	`d.setValue(i);`
	`d.setPrev(r);`
	`if(r!=null)r.setNext(d);`
	`if(r==null)f=nd;`					

Questionnaire for Chapter 5

Date: Participant code:

Before you started the experiment, how familiar would you say you were with these statements?

Stimuli	Stimuli content	Very unfamiliar	Unfamiliar	Neutral	Familiar	Very familiar

S1 A	`import java.util.Scanner;`
	`class A{`
	`  public static void main(String[]args){`
	`  Scanner s=new Scanner(System.in);`
	`  double l=s.nextDouble();}}`					

S2 A	`import java.util.Scanner;`
	`class S{`
	`  public static void main(String[]args){`
	`  Scanner b=new Scanner(System.in);`
	`  int n=b.nextInt();}}`					

S1 B	`int n,i,j;`
	`  int a[]=new int[n];`
	`  for(i=0;i<(n-1);i++){`
	`  for(j=0;j<n-i-1;j++)}}`					

S2 B	`int num,i,j,tem;`
	`if(ar[j]>ar[j+1]){`
	`  tem=ar[j];`
	`  ar[j]=ar[j+1];}`					

D1 A	`InputStream is=null;`
	`  Try{`
	`  is=new FileInputStream(f);`
	`  byte c[]=new byte[2*1024];`					

D2 A	`try{`
	`  Files.createFile(f);}`
	`  catch(FileAlreadyExistsException x){`
	`  System.err.format("n",f);}`					

D1 B	`Node<T>n=new Node<T>();`
	`  n.setValue(i);`
	`  n.setNext(f);`
	`  if(f!=null)f.setPrev(n);`
	`  if(f==null)r=n;`					

D2 B	`Node<T>d=new Node<T>();`
	`  d.setValue(i);`
	`  d.setPrev(r);`
	`  if(r!=null)r.setNext(d);`
	`  if(r==null)f=nd;`					

References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, *9*(4), 422–433. https://doi.org/10.3758/BF03197568

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *10*(3), 483–495. https://doi.org/10.1037/0278-7393.10.3.483

Albehaijan, N., & Cheng, P. C. (2019). Measuring Programming Competence by Assessing Chunk Structures in a Code Transcription Task. In C. Goel, Ashok, Seifert, Colleen and Freksa (Ed.), In *Proceedings of the 41st Annual Meeting of the Cognitive Science Society.* (Vol. 2, pp. 76–82). Montreal, Canada, 24 - 27 July, 2019.

Anderson, J. R. (2000). *Learning and memory: An Integrated approach.* (second ed.). John Wiley & Sons, Inc.

Atwood, M. E., & Ramsey, H. R. (1978). Cognitive structures in the comprehension and memory of computer programs: An investigation of computer program debugging. *Research institute for the behavioral and social science, Alexandria, Virginia*. USA.

Atwood, M. E., Turner, A. A., Ramsey, H. R., & Hooper, J. N. (1978). An Exploratory Study of the Cognitive Structures Underlying the Comprehension of Software Design Problems. *Research institute for the behavioral and social science, Alexandria, Virginia*. USA.

Barfield, W. (1986). Expert-novice differences for software: Implications for problem-solving and knowledge acquisition. *Behaviour and Information Technology*, *5*(1), 15–29. https://doi.org/10.1080/01449298608914495

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, *18*(6), 543–554. https://doi.org/10.1016/S0020-7373(83)80031-5

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, *4*(1), 55–81. https://doi.org/10.1016/0010-0285(73)90004-2

Cheng, P. (2014). Copying equations to assess mathematical competence: An evaluation of pause measures using graphical protocol analysis. In *Proceedings of the 36th Annual Meeting of the Cognitive Science Society* (pp. 319–324).

Cheng, P. (2015). Analyzing chunk pauses to measure mathematical competence : Copying equations using ‘ centre - click ’ interaction . In: Noelle, D C, Dale, R, Warlaumont, A S, Yoshimi, J and Matlock, J (eds.). In *Proceedings of the 37th Annual Conference of the Cognitive Science Society. Cognitive Science Society., Austin, TX* (pp. 345–350).

Cheng, P. C., & Albehaijan, N. (2020). Some Determinants of Chunk Size in Sequential Behavior : Individual Differences in the Transcription of Alphanumeric Strings. In D. Dension, M. Mack, Y. Xu, & B. C. Armstrong (Eds.), In *Proceedings of the 42nd Annual Conference of the Cognitive Science Society* (Vol. 1, pp. 1164–1170). Austin: TX.

Cheng, P., McFadzean, J., & Copeland, L. (2001). Drawing out the temporal signature of induced perceptual chunks. In *Proceedings of the Twenty-Third Annual Conference of the Cognitive Science Society*, (pp. 200–205).

Cheng, P., & Obaidellah, U. (2009). Graphical Production of Complex Abstract Diagrams:Drawing Out Chunks and Schemas. In *Proceedings of the Annual Meeting of the Cognitive Science Society, 31(31)* (pp. 2843–2848).

Cheng, P., & Rojas-Anaya, H. (2005). Writing out a temporal signal of chunks: patterns of pauses reflect the induced structure of written number sequences. In *Proceedings of the 27th Annual Conference of the Cognitive Science Society* (pp. 424–429).

Cheng, P., & Rojas-Anaya, H. (2006). A Temporal Signal Reveals Chunk Structure in the Writing of Word Phrases. *In Proceedings of the Twenty Eighth Annual Conference of the Cognitive Science Society. Mahwah, NJ: Lawrence Erlbaum*.

Cheng, P., & Rojas-Anaya, H. (2007). Measuring Mathematic Formula Writing Competence : An Application of Graphical Protocol Analysis. In *The Annual Meeting of the Cognitive Science Society,* (Vol. 29).

Cheng, P., & Rojas-Anaya, H. (2008). A Graphical Chunk Production Model : Evaluation Using Graphical Protocol Analysis With Artificial Sentences. In B. C. Love, K. McRae & V. M. Sloutsky (Eds.)*, In Proceedings of the Thirtieth Annual Conference of the Cognitive Science Society.* (pp. 1972-1977). *Austin, TX.*

Cohen, J. (1992). *Statistical Power Analysis for the Behavioral Science*. New York: John Wiley.

Cowan, N. (2001). The magical number 4 in short-term memory : A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, *24*(1), 87–185. https://doi.org/https://doi.org/10.1017/S0140525X01003922

Curtis, B., Forman, I., Brooks, R., Soloway, E., & Ehrlich, K. (1984). Psychological perspectives for software science. *Information Processing and Management*, *20*(12), 81–96. https://doi.org/10.1016/0306-4573(84)90041-4

Davis, J. S. (1984). Chunks: A basis for complexity measurement. *Information Processing and Management*, *20*(12), 119–127. https://doi.org/10.1016/0306-4573(84)90043-8

Deitel, P., & Deitel, H. (2017). *Java How to program* (11th ed.). Pearson, 2017.

Dimitri, G. M. (2015). The impact of Syntax Highlighting in Sonic Pi. In *Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group* (pp. 59–68). PPIG 2015.

Education department (2016). Official Statistics Graduate outcomes, by degree subject and university. Retrieved February 5, 2018, from https://www.gov.uk/government/statistics/graduate-outcomes-by-degree-subject-and-university

Egan, D. E., & Schwartz, B. J. (1979). Chunking in recall of symbolic drawings. *Memory & Cognition*, *7*(2), 149–158. https://doi.org/10.3758/BF03197595

Gobet, F., Lane, P. C. R., Croker, S., Cheng, P. C., Jones, G., Oliver, I., & Pine, J. M. (2001). Chunking mechanisms in human learning. *Trends in Cognitive Science*, *5*(6), 236–243. https://doi.org/https://doi.org/10.1016/S1364-6613(00)01662-4

Gobet, F., & Simon, H. A. (1996). Templates in Chess Memory: A Mechanism for Recalling Several Boards. Cognitive Psychology, 31(1), 1–40. https://doi.org/10.1006/cogp.1996.0011

Gonzalez, C., Anderson, J., Culmer, P., Burke, M. R., Mon-Williams, M., & Wilkie, R. M. (2011). Is tracing or copying better when learning to reproduce a pattern? *Experimental Brain Research*, *208*(3), 459–465. https://doi.org/10.1007/s00221-010-2482-1

Grocevs, A., & Prokofjeva, N. (2016). The Capabilities of Automated Functional Testing of Programming Assignments. In *2nd International Conference on Higher Education Advancesrocedia - Social and Behavioral Sciences* (Vol. 228, pp. 457–461). https://doi.org/10.1016/j.sbspro.2016.07.070

Hemmendinger, D. (2017). Computer programming language. Retrieved March 18, 2017, from https://www.britannica.com/technology/computer-programming-language

Higgins, C. a., Gray, G., Symeonidis, P., & Tsintsifas, A. (2005). Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing*, *5*(3), Article 5. https://doi.org/10.1145/1163405.1163410

Ismail, H. B., & Cheng, P. (2021). Competence assessment by stimulus matching: an application of GOMS to assess chunks in memory. In *Proceedings of 19th Internation Conference on Cognitive Modeling. Society for Mathematical Psychology*.

Jones, G. (2012). Why Chunking Should be Considered as an Explanation for Developmental Change before Short-Term Memory Capacity and Processing Speed. *Frontiers in Psychology*, *3*, 167. https://doi.org/10.3389/fpsyg.2012.00167

Kalogeropoulos, N., Tzigounakis, I., Pavlatou, E. A., & Boudouvis, A. G. (2013). Computer-based assessment of student performance in programing courses. *Computer Applications in Engineering Education, 21*(4), 671-683. https://doi.org/10.1002/cae.20512

Lane, P. C. R., Cheng, P., & Gobet, F. (2000). CHREST+ : Investigating How Humans Learn to Solve Problems Using Diagrams. *Arti. Intell. Simul. Behav. Quart.*, *103*, 24–30.

Leijten, M., & Van Waes, L. (2005). Writing with speech recognition: The adaptation process of professional writers with and without dictating experience. *Interacting with Computers*, *17*(6), 736–772. https://doi.org/10.1016/j.intcom.2005.01.005

Lister, R., Seppälä, O., Simon, B., Thomas, L., Adams, E. S., Fitzgerald, S., … Sanders, K. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36*(4), 119–150. https://doi.org/10.1145/1041624.1041673

Liu, D., Xu, S., & Cui, Z. (2012). Programmer's performance with the keystroke as an indicator: A

further study. In *Proceedings of the 11th International Conference on Computer and Information Science* (pp. 577–583). *IEEE/ACIS 2012*. https://doi.org/10.1109/ICIS.2012.88

Martinez, M. E., & Mead, N. A. (1988). *Computer Competence, The First National Assessment*. *National Assessment of Educational Progress, Educational Testing Service, Educational Resources Information Center (ERIC)* (Vol. 17-CC–01). Princeton, New Jersey.

Matsuhashi, A. (1981). Pausing and Planning: The Tempo of Written Discourse Production. *Research in the Teaching of English*, *15*(2), 113–134.

Matsuhashi, A. (1987). Revising the plan and altering the text. In *A. Matsuhashi (Ed.), Writing in real time-modelling production processes* (pp. 197–223). Norwood, NJ: Ablex Publishing Company.

McKay, B., & Shneiderman, D. (1976). Experimental Investigations of Computer Program Debugging and Modification. In *Proceedings of the Human Factors Society Annual Meeting* (Vol. 20, pp. 557–563). https://doi.org/10.1177/154193127602002401

McKeithen, K. B., Reitman, J. S., Rueter, H. H., & Hirtle, S. C. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, *13*(3), 307–325. https://doi.org/10.1016/0010-0285(81)90012-8

Miara, R. J., Musselman, J. A., Navarro, J. A., & Shneiderman, B. (1983). Program Indentation and Comprehensibility. *Communications of the ACM*, *26*(11), 861–867. https://doi.org/10.1145/182.358437

Miller, G. A. (1956). The magical number seven , plus or minus two : some limits on our capacity for processing information. *Psychological Review*, *63*(2), 81–97. https://doi.org/10.1037/h0043158

Obaidellah, U. (2016). Comprehension and Composition of Flowcharts. In *the 27th Annual Conferecne of the Psychology of Programming Interest Group, PPIG*.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137–168. https://doi.org/10.1016/0732-118X(84)90018-7

Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, *19*(3), 295–341. https://doi.org/10.1016/0010-0285(87)90007-7

Rambally, G. K. (1986). The Influence of Color on Program Readability and Comprehensibility. In *Proceedings of the 17th ACM SIGCSE symposium on Computer science education* (Vol. 18, pp. 173–181). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/953055.5702

Rawson, K., & Stahovich, T. (2013). Predicting course performance from homework habits. In *Proceedings of the 120th Annual ASEE Conference and Exposition* (pp. 23–974). American Society for Engineering Education.

Rawson, K., Stahovich, T. F., & Mayer, R. E. (2017). Homework and achievement: Using smartpen

technology to find the connection. *Journal of Educational Psychology*, *109*(2), 208–219. https://doi.org/10.1037/edu0000130

Roller, R., & Cheng, P. (2014). Observed strategies in the freehand drawing of complex hierarchical diagrams. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, (Vol. 36).

Romli, R., Sulaiman, S., & Zamli, K. Z. (2015). Improving Automated Programming Assessments: User Experience Evaluation Using FaSt-generator. *Procedia Computer Science*, *72*, 186–193. https://doi.org/10.1016/j.procs.2015.12.120

Sarkar, A. (2015). The Impact of Syntax Colouring on Program Comprehension. In *Preceedings of the 26th Annual Conference of the Psychology of Programming Interest Group, PPIG* (pp. 49–58).

Schilperoord, J. (1996). *It's about time. Temporal aspects of cognitive processes in text production.* (Rodopi B.). USL&C.

Schilperoord, J. (2001). On the cognitive status of pauses in discourse production. In *G. Rijlaarsdam (Series ed.) & T. Olive & C.M. Levy (Vol. eds.), Studies in Writing: Volume 10: Contemporary Tools and Techniques for Studying Writing* (pp. 61–87). Kluwer Academic Publishers: Netherlands.

Shneiderman, B. (1976). Exploratory experiments in programmer behavior. *International Journal of Computer & Information Sciences*, *5*(2), 123–143. https://doi.org/10.1007/BF00975629

Shneiderman, B. (1977). Measuring computer program quality and comprehension. *International Journal of Man-Machine Studies*, *9*(4), 465–478. https://doi.org/10.1016/S0020-7373(77)80014-X

Shneiderman, B., & Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, *8*(3), 219–238. https://doi.org/10.1007/BF00977789

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, *20*, 373–381. https://doi.org/10.1145/359605.359610

Sitthiworachart, J., & Joy, M. (2004). Effective Peer Assessment for Learning Computer Programming. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Vol. 36, pp. 122–126). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1007996.1008030

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge.pdf. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595–609. https://doi.org/10.1109/TSE.1984.5010283

Spelman Miller, K. & Sullivan, K. P. H. (2006). Keystroke logging: an Introduction. In *G. Rijlaarsdam (Series Ed.) and K.P.H. Sullivan, & E. Lindgren. (Vol. Eds.), Studies in Writing, Vol.18, Computer Keystroke Logging: Methods and Applications.* (pp. 1–9). Oxford; Elsevier.

Spelman Miller, K. (2000). Academic writers on-line: Investigating pausing in the production of

text. *Language Teaching Research*, *4*(2), 123–148. https://doi.org/10.1177/136216880000400203

Spelman Miller, K. (2006). The pausological study of written language production. In *G. Rijlaarsdam (Series Ed.) and K.P.H. Sullivan, & E. Lindgren. (Vol. Eds.), Studies in Writing, Vol.18, Computer Keystroke Logging: Methods and Applications.* (pp. 11–30). Oxford; Elsevier.

Stahovich, T. F., & Lin, H. (2016). Enabling data mining of handwritten coursework. *Computers and Graphics (Pergamon)*, *57*, 31–45. https://doi.org/10.1016/j.cag.2016.01.002

Tremblay, G., Gu´erin, F., Pons, A., & Salah, A. (2008). Oto, a generic and extensible tool for marking programming assignments. *Software - Practice and Experience*, *38*(3), 307–333. https://doi.org/10.1002/spe.839

Truong, N., Bancroft, P., & Roe, P. (2005). Learning to program through the web. *ACM SIGCSE Bulletin*, *37*(3), 9–13. https://doi.org/10.1145/1151954.1067452

University Education in Saudi Arabia. (2017). Retrieved February 5, 2018, from https://www.moe.gov.sa/ar/HighEducation/Pages/default.aspx

Van Genuchten, E., & Cheng, P. .-H. (2010). Temporal Chunk Signal Reflecting Five Hierarchical Levels in Writing Sentences. In S. Ohlsson & R. Catrambone (Ed.), *Proceedings of the 32nd Annual Conference of the Cognitive Science Society.* (pp. 922–1927). Austin, TX: Cognitive Science Society.

van Genuchten, E., Cheng, P. C.-H., Leseman, P. P. M., & Messer, M. H. (2009). Missing working memory deficit in dyslexia: Children writing from memory. In N. A. Taatgen, & H. van Rijn (Eds.), *Proceedings of the 31st Annual Cnference of the Cognitive Science Society. (pp. 1674-1679). Austin, TX: Cognitive Science Society.*

Wengelin, Å. (2006). Examining pauses in writing: theory, methods and empirical data. In *Computer Keystroke Logging and writing: Methods and Applications* (pp. 107–130). Brill; Elsevier.

Wiedenbeck, S. (1991). The initial stage of program comprehension. *International Journal of Man-Machine Studies*, *35*(4), 517–540. https://doi.org/10.1016/S0020-7373(05)80090-2

Ye, N., & Salvendy, G. (1996). Expert-novice knowledge of computer programming at different levels of abstraction. *Ergonomics*, *39*(3), 461–481. https://doi.org/10.1080/00140139608964475

Zulkifli, M. (2013). *Applying Pause Analysis to Explore Cognitive Processes in the Copying of Sentences by Second Language Users.* Unpublished PhD Thesis, Department of Informatics, University of SUSSEX.