INVESTIGATING NOVICE PROGRAMMERS' MENTAL MODELS

by

Syeda Fatema Mazumder

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2024

Approved by:

Dr. Manuel A. Pérez-Quiñones

Dr. Celine Latulipe

Dr. Heather Lipford

Dr. Debarati Basu

Dr. Alexia Galati

Dr. Erik Saule

ABSTRACT

SYEDA FATEMA MAZUMDER. Investigating Novice Programmers' Mental Models. (Under the direction of DR. MANUEL A. PÉREZ-QUIÑONES)

Novice programmers are known for holding incomplete and inconsistent mental models. A mental model stores knowledge that reflects a person's belief system, helps determine actions, and facilitates learning. Mental model *correctness* and *consistency* are two criteria that make a mental model useful. Though the literature on mental models is rich with more than two decades of research, novice programmers' mental model is understudied in the CS education research community. Guided by the mental model theories from psychology and cognitive science, I investigated novice programmers' mental models of arrays before and after CS1 course instruction. Furthermore, I explored the gap that might exist between students with varying levels of prior programming experience. To that end, by following the theories of mental models, I defined the mental models for Java arrays, including assertions of the array's *parts* and *state changes*. I further decomposed the array's *parts* and *state changes* into four sub-components each (parts: *name, index, type, element*; state changes: *declaration, instantiation, assigning literals, assignment*). To elicit the mental model assertions of novice programmers from large CS1 classrooms, I adopted a multiple choice-based questionnaire approach (the Mental Model Test of Arrays) covering each array's component. I collected responses from novice programmers as they entered a CS1 course and transitioned into a CS2 course. I analyzed participants' mental model assertions based on their correctness and consistency. The results show that participants' mental model correctness and consistency improved after formal classroom instruction. Moreover, even though improved, I found evidence that the mental model components of *state changes* were less accurate and consistent to novice programmers than the *parts*. In addition, participants with prior programming experience had significantly

lower mental model correctness and consistency than those with prior programming experience before classroom instruction on arrays. The mental model test of arrays highlighted several novice programmers' misconceptions. Over half of the participants held at least one misconception before and after learning arrays in classrooms. Novice programmers mostly held misconceptions about the array's *declaration* (*state change*) as incoming CS1 students and when transitioning into CS2. After classroom instruction, the number of students holding misconceptions about the *parts* components decreased. However, for the *state changes* components, in most cases, the number of students holding misconceptions remained almost the same even after classroom instruction. I close my dissertation by summarizing the overall findings while investigating novice programmers' mental models in their different learning trajectories. Lastly, I discuss the implications of my research in designing instructional materials for CS educators on possible solutions to mitigate the mental model gap of novice programmers.

DEDICATION

I dedicate this work to my dear husband, Saquib Sarwar, for his relentless support, sacrifices, energy, and time. I would also like to dedicate it to my mother, Farida Yesmin, and two dear elder sisters in the Charlotte community, Hasina and Priyanka, who stood by my family when I needed help the most.

ACKNOWLEDGEMENTS

I begin with the utmost gratitude and praise for our Creator, the Al-Mighty, the most merciful and compassionate. I am thankful to our beloved Prophet Muhammad (peace be upon him) for showing the way of kindness and mercy that shaped my morality.

My heartfelt gratitude to my dearest advisor, Dr. Manuel A. Pérez-Quiñones, whose constructive feedback, insights, guidance, and support have made my doctoral journey delightful and positive. He is an excellent human being who treats his students no less than colleagues. His constant feedback and discussion drew many amazing research ideas. Many doctoral students do not get the opportunity of one helpful advisor; I experienced two. I consider myself very fortunate that I could start my doctoral journey with Dr. Celine Latulipe as my advisor. I am grateful for her compassion towards me and my family. I could always reach out to her whenever I needed mental support. She trained me very well to begin my journey in academia. I am thankful to Dr. Mary Lou Maher and Dr. Celine Latulipe, who considered me an international graduate student, regardless of knowing I was three and half months pregnant then. I am thankful for Dr. Maher's support in giving me research and teaching opportunities and guiding me in these opportunities.

I am thankful to my committee members for their insights and constructive feedback. I am deeply grateful to the faculty members who let me collect data in their classrooms. I am thankful to all my mentors at UNC Charlotte for helping me strive. My special thanks to Dr. Richard Lambert for helping me numerous times with the statistical analysis and to Dr. Lisa Russell-Pinson for her support and guidance.

I express my gratitude to my undergraduate thesis advisor, Dr. Moinul Islam Zaber. He introduced me to HCI and research. I am grateful to all my mentors from my birth till now, as each one of them shaped my thinking and contributed to my growth.

I am thankful to my department and the graduate school at UNC Charlotte for supporting me financially by offering assistantships, fellowships, and, most importantly, GASP. I am thankful to my HCI labmates, especially Madiha, Johanna, Jeba, and Stephen.

My beloved husband Saquib, I want you to know that I saw your sacrifices and your hard work to support me. You left your family and job and came here with me to support me and our kids. I could never have achieved this dream without your all forms of support. Sarina, my dearest daughter, you have been so supportive of my study and work. I am thankful to you. Sabira, the little one, thank you for strengthening me at the end of my doctoral journey. I am thankful to my parents especially to my mother, Farida Yesmin, for helping me whenever I needed help. I am grateful for her nourishment and the relentless time and energy she spent on my growth. I am grateful to my parents-in-law for always encouraging and supporting me. I am thankful to my maternal aunts, especially Aunt Taslima, for her emotional coaching. I drew my strength to carry on my doctoral journey despite experiencing the challenges of one pandemic, two pregnancies, and a genocide (Gaza) from my late maternal grandmother, Amirun Nesa.

I am deeply grateful to the members of the Bangladeshi Student Organization for their support, including car rides when we did not own a car, meal trains when my kids were born, and constant guidance and support. You are our family in Charlotte. I am thankful to our host family in Charlotte, the Dalton family, for helping us understand the foreign culture and norms. I am grateful to the many Bangladeshi families in Charlotte who supported me. My deepest gratitude towards our neighbor, the Karim family. They treated us as their family and supported us with food, guidance, and care when we needed them most. My family can never return your debt. You are in our hearts and prayers.

TABLE OF CONTENTS

LIST OF TABLES xvi

LIST OF FIGURES xxi

LIST OF ABBREVIATIONS xxviii

CHAPTER 1: INTRODUCTION 1

1.1. Problem Scope and Motivation 2

1.2. My approach 4

1.3. Thesis Objectives 6

1.4. Definitions 7

1.5. Contributions 8

1.6. Overview of the Dissertation 8

CHAPTER 2: LITERATURE REVIEW 10

2.1. Mental Model 10

2.1.1. Mental Model of Arrays 11

2.1.2. Mental Model Consistency 12

2.1.3. Mental Model Correctness 13

2.1.4. Mental Model and Constructivism 13

2.1.5. Impact of Diagrams in Mental Models 17

2.1.6. Novice programmers' mental model elicitation 18

2.2. Knowledge Assessment Instruments 21

2.2.1. Origin and Develoment 21

2.2.2. Validation 22

2.3. Misconceptions 23

2.3.1. Studies of Misconceptions in Other Domains 24

2.3.2. Studies of Misconceptions in Introductory Programming 26

2.3.3. Common misconceptions of CS1 students 28

CHAPTER 3: TEXTBOOK REPRESENTATION OF ARRAY'S PARTS AND STATE CHANGES 30

3.1. Introduction 30

3.2. Mayer's Decomposition of *Parts* and *State Changes* 31

3.3. Methodology 32

3.3.1. Defining Programming Concept's *Parts* and *State Changes* 32

3.3.2. Textbook Selection 34

3.3.3. Data Collection 35

3.4. Results 37

3.4.1. Variables 37

3.4.2. Arrays 38

3.4.3. Objects 40

3.5. Discussion 41

3.6. Limitations 43

3.7. Conclusion 44

CHAPTER 4: THE MENTAL MODEL TEST: AN INSTRUMENT TO ELICIT MENTAL MODELS 45

4.1. Introduction 45

4.2. Definitions and Examples 46

4.3. Development of MMT-A 48

4.4. Collection of Mental Model Assertions 50

4.5. Measurement of Consistency 51

4.6. Measurement of Correctness 53

4.7. Classification of Mental Models 53

4.8. Measurement of a Mental Model 55

4.9. Identification of Misconceptions 55

4.10. Example: A Case Study 56

4.11. MMT-A is not a Concept Inventory 58

4.12. Deployment of MMT-A 59

4.13. Conclusion 61

CHAPTER 5: INCOMING NOVICE PROGRAMMERS' MENTAL MODELS 63

5.1. Introduction 63

5.2. Data Collection 64

5.3. Participants 64

5.4. Results 65

5.4.1. Mental Model Assertions 65

5.4.2. Mental Model Correctness 75

5.4.3. Mental Model Consistency 76

5.4.4. Mental Model Score: Combining Correctness & Consistency 77

5.4.5. Mental Model Score and Demographics 77

5.4.6. Mental Model Classification Frequency Distribution 79

5.5. Discussion 82

5.5.1. Incoming Novice Programmers' Mental Models 82

5.5.2. Parts vs. State Changes 86

5.5.3. Students' Demographics and Mental Models 87

5.6. Summary 89

CHAPTER 6: NOVICE PROGRAMMERS' MENTAL MODELS AFTER INSTRUCTION 90

6.1. Introduction 90

6.2. Data Collection 91

6.3. Participants 91

6.4. Results 92

6.4.1. Mental Model Assertions 92

6.4.2. Mental Model Correctness 101

6.4.3. Mental Model Consistency 102

6.4.4. Mental Model Score: Combining Correctness & Consistency 103

6.4.5. Mental Model Score and Demographics 104

6.4.6. Mental Model Classification Frequency Distribution 106

6.5. Discussion 109

6.5.1. Novice Programmers' Mental Models after Learning Arrays 109

6.5.2. Parts vs. State Changes 111

6.5.3. Students' Demographics and Mental Models 111

6.6. Summary 112

CHAPTER 7: NOVICE PROGRAMMERS' MENTAL MODEL SHIFTS FROM PRE- TO POST-INSTRUCTION 114

7.1. Introduction 114

7.2. Data Collection 114

7.3. Participants 115

7.4. Results 115

7.4.1. Mental Model Correctness 115

7.4.2. Mental Model Consistency 117

7.4.3. Mental Model Classification 119

7.4.4. Mental Model Score 124

7.4.5. Previous Programming Experience and Mental Models 125

7.5. Discussion 127

7.5.1. Mental model correctness and consistency change at the end of the course 127

7.5.2. Parts vs. State Changes 131

7.5.3. Impact of Prior Programming Experience 132

7.6. Summary 132

CHAPTER 8: MISCONCEPTIONS IN NOVICE PROGRAMMERS' MENTAL MODELS 134

8.1. Introduction 134

8.2. Methodology 135

8.3. Results 136

8.3.1. Misconceptions Before and After Classroom Insruction 136

8.3.2. Change in Misconception from Pre-test to Post-test 143

8.4. Discussion 146

8.4.1. Misconceptions Identified 146

8.4.2. Misconceptions: Parts and State Changes 148

8.5. Summary and Limitations 149

CHAPTER 9: EXPLORING VALIDITY AND RELIABILITY OF THE MENTAL MODEL TEST 151

9.1. Introduction 151

9.2. Method 152

9.3. Results 154

9.3.1. Correlation with Course Scores 154

9.3.2. Spring 2021-Post-test Data-set 157

9.3.3. Rasch Analysis with Initial Dataset 158

9.3.4. Analysis with Full Dataset 164

9.4. Discussion 171

9.5. Conclusion 173

CHAPTER 10: EXPLORING MENTAL MODELS WITH THINK-ALOUD 174

10.1.Introduction 174

10.2.Methodology 175

10.2.1. Participants 175

10.2.2. Task 175

10.2.3. Interview Protocol 176

10.2.4. Analysis 176

10.3.Results 176

10.3.1. *Parts* Components 178

10.3.2. *State changes* Components 180

10.3.3. Incorrect Assertions 192

10.3.4. Miscellaneous Findings 195

10.4.Discussion 196

10.4.1. Parts vs. State changes 196

10.4.2. Incorrect Assertions 197

10.4.3. Exploration of mental model consistency 199

10.4.4. Impact of Programming Exposure 199

10.4.5. Implications to the MMT-A 200

10.5.Summary 201

CHAPTER 11: DISCUSSION AND CONCLUSION 202

11.1.Overall Summary of Findings 202

11.2.Implications 204

11.3.Summary of Contributions 206

11.4.Limitations 207

11.5.Future Directions 208

11.6.Conclusion 210

REFERENCES 211

APPENDIX A: THE MENTAL MODEL TEST OF ARRAYS (MMT-A) 235

APPENDIX B: EXPLANATIVE DIAGRAMS OF ARRAYS: A MODEL AND A NOTIONAL MACHINE 236

APPENDIX C: MARKING SHEET FOR CONTRADICTORY ASSERTIONS 278

APPENDIX D: ITEM RESPONSE THEORY ANALYSIS 281

LIST OF TABLES

TABLE 3.1: *Parts* & *State changes* components of Primitive Variables. 36

TABLE 3.2: *Parts* & *State changes* components of Arrays. 36

TABLE 3.3: Parts & State changes of Objects. 39

TABLE 4.1: List of mental model concepts, concept types and the number of questions in which they appear. 50

TABLE 4.2: A classification (ranking) of mental models based on correctness and consistency. 55

TABLE 4.3: Participant X's obtained (partial) mental model assertions grouped with the corresponding concept, marked contradiction, consistency status, and percentage of selecting the corresponding assertion. 58

TABLE 4.4: Participant X's (partial correctness and consistency status along with mental model classification and ranks. 58

TABLE 4.5: Data Collection Timeline 59

TABLE 5.1: List of assertions for the *part* **name**. 66

TABLE 5.2: Frequency distribution of the selection of assertions for the *part* **name**. 66

TABLE 5.3: List of assertions for the *part* **index**. 67

TABLE 5.4: Frequency distribution of the selection of assertions for the part **index**. 68

TABLE 5.5: List of assertions for the *part* **type**. 68

TABLE 5.6: Frequency distribution of the selection of assertions for the *part* **type**. 69

TABLE 5.7: List of assertions of the part **element**. 70

TABLE 5.8: Frequency distribution of the selection of assertions for the part **element**. 70

TABLE 5.9: List of assertions of the *state change* **declaration**. 71

TABLE 5.10: Frequency distribution of the selection of assertions for the *state change* **declaration**. 72

TABLE 5.11: List of assertions for the state change **instantiation**. 72

TABLE 5.12: Frequency distribution of the selection of assertions for the state change **instantiation** 72

TABLE 5.13: List of assertions for the *state change* **assigning elements**. 73

TABLE 5.14: Frequency distribution of the selection of assertions for the *state change* **assigning elements**. 74

TABLE 5.15: List of assertions for the *state change* array **assignment**. 74

TABLE 5.16: Frequency distribution of the selection of assertions for the state change **array assignment** 75

TABLE 5.17: Participant's overall correctness and mental model score with total scores for *parts* and *state changes*. The statistically significant difference between the correctness score and mental model score of arrays *parts* and *state changes* is shown in the last row. 75

TABLE 5.18: Participants' correctness, consistency, and mental model classification for each *part* and *state changes*. 76

TABLE 5.19: Participants' correctness score and mental model score by demographics. 78

TABLE 6.1: List of assertions for the *part* **name**. 92

TABLE 6.2: Frequency distribution of the selection of assertions for the *part* **name**. 93

TABLE 6.3: List of assertions for the *part* **index**. 94

TABLE 6.4: Frequency distribution of the selection of assertions for the part **index**. 94

TABLE 6.5: List of assertions for the *part* **type**. 95

TABLE 6.6: Frequency distribution of the selection of assertions for the *part* **type**. 95

TABLE 6.7: List of assertions of the part **element**. 96

TABLE 6.8: Frequency distribution of the selection of assertions for the part **element**. 96

TABLE 6.9: List of assertions of the *state change* **declaration**. 98

TABLE 6.10: Frequency distribution of the selection of assertions for the *state change* **declaration**. 98

TABLE 6.11: List of assertions for the state change **instantiation**. 99

TABLE 6.12: Frequency distribution of the selection of assertions for the state change **instantiation** 99

TABLE 6.13: List of assertions for the *state change* **assigning elements**. 99

TABLE 6.14: Frequency distribution of the selection of assertions for the *state change* **assigning elements**. 100

TABLE 6.15: List of assertions for the *state change* array **assignment**. 100

TABLE 6.16: Frequency distribution of the selection of assertions for the *state change* **array assignment**. 101

TABLE 6.17: Participant's overall correctness and mental model score with individual scores for *parts* and *state changes*. Participants' correctness score and mental model score for the *state changes* is statistically significantly lower than the *parts* (shown in the last row). 102

TABLE 6.18: Participants' correctness, consistency, and mental model classification for each *part* and *state changes* (N = 144). Here, II: Inconsistent and Incorrect mental model, CI: Consistently Incorrect mental model, IMI: Inconsistent and Mostly Incorrect mental model, CMI: Consistent and Mostly Incorrect mental model, IMC: Inconsistent and Mostly Correct mental model, CMC: Consistent and Mostly Correct Mental Model, C: Correct mental model. 104

TABLE 6.19: Participants' correctness score and mental model score by demographics. Participants' correctness and mental model scores were statistically significantly higher for those who are CS majors (last two rows). 105

TABLE 7.1: Pre-test and Post-test Mean Correctness Score of the Components of Arrays. (N = 66) 116

TABLE 7.2: Frequency distribution of participants from the pre-test and the post-test across different correctness levels. 118

TABLE 7.3: Mental Model Consistency Scores per Component: Pre-test and Post-test (N = 66). 118

TABLE 7.4: Crosstabulation showing the changes in participants' frequency of mental model consistency shift from pre-test to post-test for each component. 120

TABLE 7.5: Frequencies of participants across positive, negative, neutral mental model classification shift with the p value of Wilcoxon Signed-rank test. 124

TABLE 7.6: Pre-test and Post-test Frequency Distribution across the Mental Model Categories. (N = 66) 125

TABLE 7.7: Mental model correctness score, correctness score for *parts*, *state changes*, mental model score, mental model score for *parts*, *state changes* across different programming background from the pre-test and the post-test. 128

TABLE 8.1: List of Misconceptions found in the study. Each misconception is labeled with a unique identifier used as reference throughout the dissertation. The last column shows the percentage of participants holding the misconception. 136

TABLE 9.1: Factor Laodings and Item-Total Correlations for each item of the *parts* components. 165

TABLE 9.2: Factor Laodings and Item-Total Correlations for each item for the *state changes* components. 166

TABLE 9.3: Results of IRT analysis (N = 282) and Point Beserial Correlation for each item in the MMT-A. The third column (% correct) denotes the frequency of participants in the percentage who answered the item correctly. Point Biserial coefficients marked with an asterisk (*) denote statistically insignificant. 169

TABLE 10.1: Details of each participant's (n = 10) programming exposure, previous programming experience, and source where they learned Java. 177

178table.10.2

TABLE 10.3: Themes Emerging from the question 'what is `null`?' 185

TABLE 10.4: Incorrect assertions found in the think-aloud semi-structured interview data with participants (n = 10) separated by the highest course completion. 194

TABLE C.1: Contradictory Assertions for *name*. Assertions belonging to the same cell are contradictory. 278

TABLE C.2: Contradictory Assertions for *index*. Assertions belonging to the same cell are contradictory. 278

TABLE C.3: Contradictory Assertions for *type*. Assertions belonging to the same cell are contradictory. 279

TABLE C.4: Contradictory Assertions for *element*. Assertions belonging to the same cell are contradictory. 279

TABLE C.5: Contradictory Assertions for *declaration*. Assertions belonging to the same cell are contradictory. 279

TABLE C.6: Contradictory Assertions for *instantiation*. Assertions belonging to the same cell are contradictory. 279

TABLE C.7: Contradictory Assertions for *assigning elements*. Assertions belonging to the same cell are contradictory. 280

TABLE C.8: Contradictory Assertions for *assignment*. Assertions belonging to the same cell are contradictory. 280

LIST OF FIGURES

FIGURE 2.1: Many purpose of the mental models (diagram adapted and revised from [1]). 16

FIGURE 3.1: Representative example of a diagram for arrays found in some textbooks. 38

FIGURE 3.2: Representative example of a diagram for objects found in some textbooks. 40

FIGURE 4.1: An example showing the mapping between each choice and the corresponding assertion. 50

FIGURE 5.1: Frequency distribution (in percentage) of the categories of the mental models. Here, II: Inconsistent and Incorrect mental model, CI: Consistently Incorrect mental model, IMI: Inconsistent and Mostly Incorrect mental model, CMI: Consistent and Mostly Incorrect mental model, IMC: Inconsistent and Mostly Correct mental model, CMC: Consistent and Mostly Correct Mental Model, C: Correct mental model. 80

FIGURE 6.1: Frequency distribution (in percentage) of the categories of the mental models. Here, II: Inconsistent and Incorrect mental model, CI: Consistently Incorrect mental model, IMI: Inconsistent and Mostly Incorrect mental model, CMI: Consistent and Mostly Incorrect mental model, IMC: Inconsistent and Mostly Correct mental model, CMC: Consistent and Mostly Correct Mental Model, C: Correct mental model. 106

FIGURE 7.1: Mental model classification shift for the four *part* components. Here, we have portrayed positive rank shifts (color blue), negative rank shifts (color red), and neutral (color green). As an example, 54 participants' mental models were correct in the pre-test (Pre_C), and 59 participants' mental models were correct in the post-test (Post_C) for *name*. 121

FIGURE 7.2: Mental model classification shift for the four *state change* components. Here, we have portrayed positive rank shifts (color blue), negative rank shifts (color red), and neutral (color green). 122

FIGURE 8.1: Frequency distribution of the participants holding at least one misconception across the *parts* and *state changes* components before and after classroom instruction. 140

FIGURE 8.2: Venn diagram showing 13 participants (14%) had misconceptions on only *parts* components, 25 (26.9%) on only *state changes* components, and 15 (16.1%) on both the components. 40 (43%) participants did not have any misconceptions. 141

FIGURE 8.3: Venn diagram showing 9 participants (6.25%) had misconceptions on only *parts* components, 55 (38.19%) on only *state changes* components, and 9 (6.25%) on both the components. 71 (49.31%) participants did not have any misconceptions. 141

FIGURE 8.4: Frequency distribution of the participants across the number of misconceptions (zero, one, two, or three). 142

FIGURE 8.5: Participants' change in misconception from the pre-test to the post-test. The changes are labeled as remained (misconceptions present in the pre-test and remained in the post-test), gone (misconception present in the pre-test but diminished in the post-test), and arose (misconception was not present in the pre-test but arose in the post-test) (N = 66). 145

FIGURE 9.1: Scatterplot showing the correlation between participants' Exam 3 score and the correctness score $r = .475, N = 91, p <= 0.001$. 155

FIGURE 9.2: Scatterplot showing the correlation between participants' Exam 3 score and the mental model score ($r = .480, N = 91, p <= 0.001$). 155

FIGURE 9.3: Scatterplot showing the correlation between participants' total CS1 score and the correctness score ($r = .427, N = 91, p <= 0.001$). 156

FIGURE 9.4: Scatterplot showing the correlation between participants' total CS1 score and the mental model score ($r = .481, N = 91, p <= .001$). 156

FIGURE 9.5: Scatterplot showing the correlation between participants' Exam 3 score and the correctness score $r = .422, N = 101, p < 0.001$. 157

FIGURE 9.6: Scatterplot showing the correlation between participants' Exam 3 score and the mental model score ($r = .405, N = 101, p < 0.001$). 158

FIGURE 9.7: Scatterplot showing the correlation between participants' total CS1 score and the correctness score ($r = .468, N = 101, p < 0.001$). 159

FIGURE 9.8: Scatterplot showing the correlation between participants' total CS1 score and the mental model score ($r = .446, N = 101, p < .001$). 160

FIGURE 9.9: Item-person map (Wright map) portraying person ability distribution with the item difficulty distribution. The letters prefixed to the item numbers denote: N - *part: name*, T - *part: type*, I - *part: index*, E - *part: element*, D - *state change: declaration*, IN - *state change: instantiation*, AE - *state change: assigning elements*, A - state change: assignment. 162

FIGURE 9.10: (a) Question T2 and (b) its Item Characteristic Curve. 170

FIGURE 9.11: (a) Question A1 and (b) its Item Characteristic Curve. 170

FIGURE 9.12: (a) Question A5 and (b) Question A6 used in the Mental Model Test appeared to be problematic in the item analysis. 171

FIGURE 10.1: Think-aloud question probe for *part: name*. The correct answer is option (b). 179

FIGURE 10.2: Think-aloud question probe for *part: index*. The correct answer is option (c). 179

FIGURE 10.3: Think-aloud question probe for *part: elements*. The correct answer is option (a). 180

FIGURE 10.4: Think-aloud question probes for *state change: declaration*. The correct answer is option (d). 181

FIGURE 10.5: Think-aloud question probes for *state change: declaration*. The correct answer is option (a). 182

FIGURE 10.6: Think-aloud question probes for *state change: instantiation*. The correct answer is option (b). 187

FIGURE 10.7: Think-aloud question probes for *state change: instantiation*. The correct answer is option (a). 187

FIGURE 10.8: Think-aloud question probes for *state change: assigning elements*. The correct answer is option (b). 190

FIGURE 10.9: Think-aloud question probes for *state change: assignment*. The correct answer is option (d). 191

FIGURE 10.10: Think-aloud question probes for *state change: assignment*. The correct answer is option (b). 191

FIGURE 10.11: Think-aloud question probes for *state change: assignment*. The correct answer is option (b). 192

FIGURE B.1: The scope of the notional machine defined by ITiCSE working group [2]. 238

FIGURE B.2: As the layer of abstraction gets thinner, the NM resembles more of the conceptual model. 240

FIGURE B.3: Interaction between a notional machine, mental model, and the programming behavior presented in [3]. 242

FIGURE B.4: The interplay between the NM and the mental model from [2]. 242

FIGURE B.5: How human mind processes pictures, printed words, and spoken words from Mayer's CTML [4] 247

FIGURE B.6: From left to right: the system topology of a car's brake system and a bicycle pump from [5]. 257

FIGURE B.7: Component behavior of a car's brake system from [5]. 258

FIGURE B.8: Component behavior of a bicycle pump from [5] 258

FIGURE B.9: Four conditions of effective diagrams by Mayer et al. [5]. 260

FIGURE B.10: The system topology diagram of an array. 268

FIGURE B.11: The explanative diagram illustrating an array's state after declaration. 269

FIGURE B.12: Explanative diagrams showing the each state change after instantiation. 271

FIGURE B.13: The explanative diagrams showing the before and after state change of an array after assignment. 272

FIGURE B.14: The explanative diagrams of assigning a value to an element. 272

FIGURE B.15: The dynamics of array assignment are portrayed with this explanative diagram. 273

FIGURE D.1: Item Characteristic Curve for the item N1 included in the array's *parts* component- *name*. 281

FIGURE D.2: Item Characteristic Curve for the item N2 included in the array's *parts* component- *name*. 282

FIGURE D.3: Item Characteristic Curve for the item I1 included in the array's *parts* component- *index*. 282

FIGURE D.4: Item Characteristic Curve for the item I2 included in the array's *parts* component- *index*. 283

FIGURE D.5: Item Characteristic Curve for the item I3 included in the array's *parts* component- *index*. 283

FIGURE D.6: Item Characteristic Curve for the item I4 included in the array's *parts* component- *index*. 284

FIGURE D.7: Item Characteristic Curve for the item I5 included in the array's *parts* component- *index*. 284

FIGURE D.8: Item Characteristic Curve for the item T1 included in the array's *parts* component- *type*. 285

FIGURE D.9: Item Characteristic Curve for the item T2 included in the array's *parts* component- *type*. 285

FIGURE D.10: Item Characteristic Curve for the item E1 included in the array's *parts* component- *elements*. 286

FIGURE D.11: Item Characteristic Curve for the item E2 included in the array's *parts* component- *elements*. 286

FIGURE D.12: Item Characteristic Curve for the item E3 included in the array's *parts* component- *elements*. 287

FIGURE D.13: Item Characteristic Curve for the item E4 included in the array's *parts* component- *elements*. 287

FIGURE D.14: Item Characteristic Curve for the item E5 included in the array's *parts* component- *elements*. 288

FIGURE D.15: Item Characteristic Curve for the item D1 included in the array's *state changes* component- *declaration*. 288

FIGURE D.16: Item Characteristic Curve for the item D2 included in the array's *state changes* component- *declaration*. 289

FIGURE D.17: Item Characteristic Curve for the item In1 included in the array's *state changes* component- *instantiation*. 289

FIGURE D.18: Item Characteristic Curve for the item In2 included in the array's *state changes* component- *instantiation*. 290

FIGURE D.19: Item Characteristic Curve for the item In3 included in the array's *state changes* component- *instantiation*. 290

FIGURE D.20: Item Characteristic Curve for the item In4 included in the array's *state changes* component- *instantiation*. 291

FIGURE D.21: Item Characteristic Curve for the item AE1 included in the array's *state changes* component- *assigning elements*. 291

FIGURE D.22: Item Characteristic Curve for the item AE2 included in the array's *state changes* component- *assigning elements*. 292

FIGURE D.23: Item Characteristic Curve for the item AE3 included in the array's *state changes* component- *assigning elements*. 292

FIGURE D.24: Item Characteristic Curve for the item AE4 included in the array's *state changes* component- *assigning elements*. 293

FIGURE D.25: Item Characteristic Curve for the item AE5 included in the array's *state changes* component- *assigning elements*. 293

FIGURE D.26: Item Characteristic Curve for the item AE6 included in the array's *state changes* component- *assigning elements*. 294

FIGURE D.27: Item Characteristic Curve for the item A1 included in the array's *state changes* component- *assignment*. 294

FIGURE D.28: Item Characteristic Curve for the item A2 included in the array's *state changes* component- *assignment*. 295

FIGURE D.29: Item Characteristic Curve for the item A3 included in the array's *state changes* component- *assignment*. 295

FIGURE D.30: Item Characteristic Curve for the item A4 included in the array's *state changes* component- *assignment*. 296

FIGURE D.31: Item Characteristic Curve for the item A5 included in the array's *state changes* component- *assignment*. 296

FIGURE D.32: Item Characteristic Curve for the item A6 included in the array's *state changes* component- *assignment*. 297

LIST OF ABBREVIATIONS

CS1 Introductory Programming Course.

CS2 Basic Data Structure Course.

ITSC: 1212 Introduction to Computer Science I (CS1) course at College of Computing and Informatics

ITSC: 1213 Introduction to Computer Science II (CS1) course at College of Computing and Informatics

ITSC: 2214 Data Structure and Algorithm (CS2) course at College of Computing and Informatics

MMT-A Mental Model Test of Arrays.

CI Concept Inventory.

CS Computer Science.

SD Standard Deviation.

CHAPTER 1: INTRODUCTION

Among the many functions of mental models, some key functionalities include storing knowledge, governing actions, predicting outcomes, mirroring belief systems, and performing troubleshooting. Hence, mental models are utilized in every step towards processing new information, interacting with a system, and making a decision. Learning occurs when we can shift our mental model towards being more accurate and consistent in alignment with reality. The literature claims novice learners' mental models hold inaccurate and inconsistent beliefs [6]. Novice programmers tend to bring their own assumptions to problems [7, 8]. According to the mental model theories [9, 10] and constructivism [11, 12], learners utilize their existing knowledge to make sense of the new information. Students enter into CS1 courses with a wide variety of backgrounds, making it hard to assess how students are processing the new information and thus revising their mental models. As learners' mental models mirror their learning, it is crucial to understand them in order to maneuver them. One thing many researchers agree on is that once misinformation becomes fixed in a mental model, new information contradicting the mental model loses its acceptance [13–16]. As a consequence, instructors trying to convey the correct information may suffer one of the educationally nonproductive fates- "ignored, rejected, disbelieved, deemed irrelevant to the current issue, held for consideration at a later time, reinterpreted in light of the student's current theories, or accepted with only minor changes in the student's concept" [14, p.45]. Consequently, a novice programmer trying to debug an error in the program does not realize the bug (i.e., error) is actually in their mental models. The importance of mental models raises the need to investigate novice programmers' mental models.

1.1 Problem Scope and Motivation

Investigating novice programmers' mental models needs a lens from the theory of the mental model's origin- psychology and cognitive science. In between the eighty years since the inception of mental models, researchers used *ad hoc* definitions of mental models [10]. Consequently, most Computer Science (CS) education researchers used their own vague, often intuitive definitions and approaches to investigate mental models. Even so, there have been very few works investigating novice programmers' mental models [17,18], and none investigating novice programmers' mental models of *arrays*. Most common mental model elicitation approaches include time-consuming qualitative techniques such as cognitive interview, observation of task, and drawing [10]. CS1 classrooms being large in size makes the applicability of these research techniques unfeasible. My research aims to address these challenges.

Moreover, the research findings of novice programmers' misconceptions assert that in CS1 courses, students struggle most in understanding reference variables. The programming concept- *arrays* in Java is manipulated with reference variables and is often taught at an interesting point in the intro curriculum. It often follows the intro to control structures and logic but comes before more advanced topics. Arrays are essential for studying the implementation of linear data structures, sorting algorithms, basic linear search, binary search, and hashing functions. All of these topics are typically beyond CS1 (or at least near the end) and part of the follow-up course- CS2. Mastery of arrays, both *parts* and *state changes*, is required to transition between these two sets of topics successfully. *Arrays*- inherently a data structure consists of its structure (*parts*) and behavior (*state changes*). Moreover, arrays contain enough dynamic behavior (e.g., assignments, memory allocation) that understanding arrays involves understanding some of these more difficult aspects typical of dynamic *state changes*- a threshold concept in CS education.

In CS education, a program's dynamic *state changes* is considered a threshold

concept [19–22]. State changes are also referred to in the literature as program dynamics [19], states [20], program behavior [23], or program-memory interaction [24]. Threshold concepts are characterized as the most troublesome and transformative [21]. They are transformative as comprehending a threshold concept enables a learner to view and describe a concept in a new way and may alter their perception, which cannot be unlearnt.

State changes are dynamic and invisible although crucial to learn [25, 26]. State changes are difficult to understand as the mechanism is hidden from the perceptual view [19, 20]. The textual representation of a program has little or no connection to the state change it provokes [25, 27]. Mapping syntax properties to concrete state changes creates obstacles not only for novices but also for advanced programmers [25].

We have seen a plethora of misconceptions related to programs *state changes*, including simple assignments [17, 28], object declaration, and instantiation [7]. Being a troublesome concept, novice programmers are known for holding incorrect or incomplete mental models of *state changes* [17, 29]. Emphasizing the need to study programming concepts' *state changes*, Krishnamurthi et. al [30, p.384] stated, "comparative studies between stateful and non-stateful features are one of the most significant understudied topics in computing education".

In the end, understanding novice programmers' mental model of arrays is critical. It is more critical to understand their mental model of arrays *state changes* in comparison to the *parts* so that we can identify their point of struggle and address them.

In this dissertation, I aim to elicit novice programmers' mental models of *arrays* with the lens of mental model theories. In addition, I aim to decompose novice programmer's mental model of arrays into its *parts* and *state changes* and provide the result of a comparative study between novice programmers' mental models of *parts* and *state changes*.

1.2 My approach

To elicit novice programmers' mental models of arrays, we need a definition of mental model which must be grounded in the theories of mental models and applicable to a programming concept. From the theories and many definitions of mental models, I found two directions of definition suitable for my study.

From a device understanding perspective, many scholars proposed that the mental model is a cognitive representation of a device's components: its *parts* and behavior (*state changes*) [31–33]. This translates into programming by Sorva's following statement, "it is widely accepted that programming requires having access to some sort of 'mental model' of the system" [34, p. 8-9].

From a language acquisition and inference generation perspective, psychologist Johnson-Laird [35] stated that a mental model is a model in which learners validate a set of assertions.

From the above definitions of mental models, I conclude that mental models of a programming concept have a system like *parts* and *state changes* components. Each component has a set of assertions. In this context, an assertion is a single idea or proposition ingrained in a person's mind. To evaluate mental models, I utilize two criteria of mental models- *correctness and consistency*. De Kleer and Brown [36] in their book 'Mental Models' claimed that for a mental model to be useful, it must be: 1) consistent, 2) correct[1], and 3) robust. A mental model is *consistent* when it is free of internal contradictions. A mental model is *correct* when it is faithful to the device it is modeling. Finally, a mental model is *robust* when it can be used to troubleshoot problems (debugging in computing). Since evaluating novice programmers' mental model robustness requires a different approach, my work considered the evaluation of the mental models based on *correctness* and *consistency*.

[1]De Kleer and Brown [37] used the term correspondence instead of correct. In this dissertation, I use the term 'correctness' to imply 'correspondence.'

By analyzing the contents of 15 commonly used CS1 textbooks (mentioned in Chapter 3), I decomposed the programming concept *arrays* into its *parts* and *state changes* components. I identified the *parts* of an array as its *name, type, index*, and *elements*. I finalized the *state changes* components to be *declaration, instantiation, assigning elements*, and *assignment*. I listed and obtained the sets of factual assertions for the components from Java's syntax and semantics. For instance, a factual assertion regarding array's *part* component *index* is *indexing begins with 0*; a factual assertion regarding array's *state changes* component *instantiation* is *after instantiation, memory is allocated to store elements in the array.*

To manifest the definition into a research method to extract novice programmers' mental models and evaluate their mental models based on *correctness* and *consistency*, I needed an approach applicable to the large CS1 classrooms. Carefully constructed multiple-choice-based questionnaires can extract the same information about conceptual knowledge as short answers or open-response questions, with significant advantages in test administration and scoring [38]. Hence, I adopted a multiple-choice-based questionnaire (MCQ) approach and developed the *Mental Modet Test of Arrays (MMT-A)*. I generated questions for each component of *parts* and *state changes*. The correct choice is mapped to the factual assertion. I derived incorrect assertions from the literature on misconceptions and teaching practices. I crafted the distractors of MCQ, which are mapped to the incorrect assertions. To analyze the consistency of novice programmers' mental models, I placed the corresponding options for the assertions multiple times in the . To understand the incoming novice programmers' mental models and their mental model transition after the CS1 course, I elicited novice programmers' mental model assertions before and after classroom instruction on arrays. Based on the correctness and consistency of novice programmers' mental model assertion, I scored their mental model and presented comparative results based on *part* and *state changes* components (details are in Chapter 4). More-

over, their mental model assertion revealed their misconceptions. In this dissertation, I recorded their misconceptions before and after classroom instruction.

1.3 Thesis Objectives

Grounded on the theories of mental models, with the instrument the *Mental Model Test (MMT-A)*, my thesis aims to investigate novice programmers' mental models. The thesis aims to present novice programmers' mental model assertions and evaluation of these assertions based on *correctness* and *consistency*. I elicited their mental model assertions before and after the classroom instruction on arrays to gain an in-depth description of their mental models. Therefore, the remainder of this dissertation attends to the following research questions.

The research questions tied to the incoming CS1 students' mental models are:

RQ1. What are the characteristics of incoming novice programmers' mental models?

RQ2. How are the incoming novice programmers' initial mental models of the array's *parts* components in comparison with the *state changes* components?

RQ3. What impact do prior programming experience and demographics have on the mental model of the participants in our study before classroom instruction?

The research questions tied to the CS1 students' mental models as they transition into CS2 are:

RQ4. What are the characteristics of novice programmers' mental models after they have learned arrays?

RQ5. How are the novice programmers' mental models of the array's *parts* components in comparison with the *state changes* components after they have learned arrays?

RQ6. What impact do prior programming experience and demographics have on the mental model of the participants after they have learned arrays?

Moreover, a Pretest-Posttest study design allowed me to understand how novice programmers mental models change after formal classroom instruction of arrays. The corresponding research question is as follows:

RQ7. How do the correctness and consistency of novice programmers' mental models of arrays change after (pre-test vs. post-test) classroom instruction?

Next, I describe the several definitions that are tightly coupled in this dissertation and then summarize the research contributions and key ideas presented in each chapter.

1.4 Definitions

Several terms in this dissertation need to be highlighted and organized. I am reiterating the definitions here to define their scope and make their use consistent. Some terms have multiple definitions and are inconsistently used in the literature. Therefore, the precise definitions that I will follow throughout this dissertation are mentioned here.

Novice Programmers: In this dissertation, *novice programmers* refer to the students who are enrolled in a Java-based CS1 course and have little knowledge of Java's programming syntax and semantics.

Mental Model: I perceive the mental models of a programming concept as they contain a set of assertions for each *part* (of the structure) and *state changes*. Here, an *assertion* is a single belief or notion in a human's mind.

Mental Model Correctness: I refer to correctness as an attribute of a mental model that shows how closely the mental model aligns with the real world.

Mental Model Consistency: I refer to consistency as an attribute of mental models that indicates if internal contradiction exists among the assertions. A consistent mental model contains no contradictory assertion.

1.5 Contributions

With this dissertation, I make the following contributions to the CS education domain:

- Integration of proper definitions to elicit, evaluate, and categorize novice programmers' mental models based on the mental model theories.
- Decomposition of programming concept *arrays* into *parts* and *state changes*.
- Multiple-choice-questionnaire-based instrument to capture novice programmers' mental models of arrays.
- Portrayal of novice programmers' mental models of arrays before and after classroom instruction based on mental model assertions.
- Comparative study between novice programmers' mental models of *parts* and *state changes*.
- Inventory of novice programmers' misconceptions related to arrays.

1.6 Overview of the Dissertation

Below, I outline a summary of each chapter.

Chapter 2 : I discuss the relevant literature on mental models from the survey of interdisciplinary domains- psychology, cognitive science, and CS education. I also discuss the literature adopting MCQ-based instruments to assess knowledge and the difference between their work and mine. Lastly, I describe related research on learners' misconceptions, including CS1 education and other disciplines.

Chapter 3 : This chapter presents the content analysis of commonly used CS1 textbooks where I decomposed the concept of arrays in *parts* and *state changes*.

Chapter 4 : In this chapter, I elaborate on the details of the methodological approach to design and develop MMT-A. I describe my approach to classify mental models based on *correctness* and *consistency*. I also describe how I measured and scored mental models.

Chapter 5 : I present the study to elicit incoming novice programmers' mental models of arrays. This chapter discusses the findings based on research questions RQ1, RQ2, and RQ3.

Chapter 6 : I present the study that elicits and analyzes novice programmers' mental models of arrays after classroom instruction when they transition into a CS2 course. In this chapter, I answer the research questions RQ4, RQ5, and RQ6.

Chapter 7 : I present a quasi-experimental design study to perceive novice programmers' mental model shift after classroom instruction. In this chapter, I present the result of the paired pre-test and post-test, which answers RQ7.

Chapter 8 : In this chapter, I present the misconceptions found in novice programmers' mental models before and after classroom instruction.

Chapter 9 : In this chapter, I explored the validity and reliability arguments with the data collected from the MMT-A with several statistical procedures.

Chapter 10 : I present the details and findings of a semi-structured think-aloud interview study with a subset questionnaire from the MMT-A to support the quantitative data recorded with the MMT-A.

Chapter 11 : I summarize the findings from each study. Based on these findings, I discuss their implications and conclude with future research directions.

CHAPTER 2: LITERATURE REVIEW

This thesis builds on three areas of research: (1) the literature of **mental models** to learn their application in the domain of a programming concept, (2) multiple-choice based **knowledge assessment instruments** to understand and leverage their development and validation process, and (3) existing studies on learners **misconceptions** to learn common programming misconceptions and to understand the techniques to identify them. In the following, I review the related work in each area.

2.1 Mental Model

Around the early 1940s, Craik [39] termed mental models as the small-scale internal models of the external world. From a cognitive scientist's point of view, Norman [6] posited that people form internal models while interacting with their surroundings. The internal model empowers them with the predictive and explanatory power to understand the interactions. From the system dynamics domain, psychologists Doyale and Ford [40] defined a mental model as an enduring and accessible yet limited internal conceptual representation whose structure is similar to the perceived structure of the system. Jones et al. [41] followed the same view in their synthesis of the interdisciplinary literature on mental models. From a device understanding perspective, many researchers agree that a mental model is a cognitive representation of a system's internal representation, i.e., its component parts and their behaviors [31–33].

From a language acquisition and inference generation perspective, psychologist Johnson-Laird [35] stated that a mental model is a model in which learners validate a set of assertions. In this context, an assertion is a single idea or proposition ingrained in a person's mind.

De Kleer and Brown [36] in their book 'Mental Models' claimed that for a mental model to be useful, it must be: 1) *consistent*, 2) *correct*[1], and 3) *robust*. *Consistency* means the model is free of internal contradictions. A mental model is considered *corresponding* or correct when it is faithful to the device it is modeling. Lastly, a learner's mental model is *robust* when it can be used to successfully troubleshoot problems. In the domain of programming, troubleshooting a program is referred to as debugging. My work explores the first two criteria: *consistency* and *correctness* in the mental model of novice programmers. Since investigating a programmer's debugging behavior requires a different approach than investigating consistency and correctness, I leave this as a future work.

2.1.1 Mental Model of Arrays

Lonati et al. [27] envision a program as a coexisting concrete physical object and an abstract entity. A program is a concrete physical object when we code and run it in a computing device. Also, a program is an abstract entity when we want to make sense of its behavior by envisaging a world of abstractions that somehow come alive in our minds. Arrays in programming have a system-like *parts* and *state changes*. When I am considering mental models of arrays, similar to a concrete physical system-like entity, the mental model of arrays carries the representation of its internal representation (i.e., *parts* and *state changes*). Also, as the *parts* and *state changes* of an array are abstract entities in a programmer's mind, a mental model of arrays can also be seen from Johnson-Laird's [35] definition of a mental model from an abstract language acquisition perspective.

By utilizing the definitions of mental models from two perspectives (concrete: system-like and abstract: language and inference generation), I conclude that mental models of a programming concept have a system like *parts* and *state changes* components where each component has a set of assertions.

[1]De Keer and Brown [36] used the term correspondence to imply correctness.

2.1.2 Mental Model Consistency

As stated by De Kleer and Brown [36], a mental model is *consistent* when it is free of internal contradiction. Furthermore, they implied that there should be one model for each component. Mental model consistency is also referred to as cognitive consistency [42]. One of the most important aspects of mental model inconsistency is that it serves as a cue for potential errors in one's belief system [43]. Identifying errors in a belief system is important as they can undermine context-appropriate behavior. Furthermore, an inconsistent belief system signals that the current belief system needs to be revised for context-appropriate action [42]. De Kleer and Brown [36] restricted themselves to define consistency rigorously. Later, Johnson-Laird [35] provided a definition of cognitive consistency with examples so that collectively, it made it clear how to measure it. According to him, mental model inconsistency is found when one assertion contradicts another. To exemplify this claim, Johnson-Laird [35] laid out three assertions in the context of Chernobyl nuclear power plant:

- The reactor is not dangerous if and only if it is intact.
- If it is intact then all its graphite is inside it.
- The reactor is not dangerous and some of its graphite is not inside it.

Based on the first two assertions, the last assertion is inconsistent since they are contradictory. These three assertions cannot be true at the same time. Based on this argument, a mental model is *consistent* once it is determined that there are no two assertions that contradict each other. The mental model is deemed as *inconsistent* if such proof cannot be found. Thus, to prove the contradiction between assertions, Johnson-Laird [35] proposes to choose any assertions in the set and prove its negation from the remaining assertions. If a negation can be found, then the assertions are inconsistent. In Section 4.5, I describe how I measured the consistency of novice

programmers' mental models of arrays based on Johnson Laird's [35] theory and approach.

2.1.3 Mental Model Correctness

While consistency is based on the analysis of contradictions among assertions, it fails to capture the accuracy of the mental model. A *consistent* mental model can be totally inaccurate, having assertions that do not contradict each other (i.e., they are consistent) but are wrong or inaccurate.

This leads us to correctness, another important criterion of useful mental models as defined by De Kleer and Brown [36]. A mental model is *correct* when it is accurate regarding how the device works. For our work, a *correct* mental model of arrays contains assertions that are valid in the Java language. This takes into consideration both syntactical and semantic properties.

2.1.4 Mental Model and Constructivism

The notion of *constructivism* first occurred in Piaget's [11] mind when he was observing molluscs' adaptation to their new habitats. Inspired by this event, Piaget [11] perceived the learning experience as an ever-evolving adaptation process where new knowledge modifies and re-organizes in relation to the learners' own ever-evolving structure of thinking. This learning experience was termed as *constructivism* [11]. Constructivism claims that knowledge is constructed by combining the experiential world with existing cognitive structures, and it evolves over time [11].

Constructivism has been widely used in science [44] and mathematics education [45]. Ben-Ari [46] was the pioneer in introducing constructivism in CS education. According to constructivism, learners build up new knowledge on existing similar models they are retaining [11]. Understanding that learners do not start from nothing, teachers are well advised to treat each individual's learning experience differently. The implications are then two-fold. First, it implies an explicit recognition of the learner as

someone possessing rich previous knowledge. Second, it implies a utilization of that rich existing knowledge to build new knowledge structure and further meaningful understandings [12].

Ben-Ari [47], from a constructivist perspective, emphasized the value of empirical research to determine the initial mental model of students. Hence, according to Ben-Ari [47], a teacher cannot disregard the students' existing knowledge. Most importantly, their initial mental model can serve as a guide to design instruction.

2.1.4.1 Purpose of Mental Model

The purpose of mental models makes it clear why it is crucial to study programmers' mental models. Below, I summarize the purposes of mental models.

Predicting outcomes: The primary purpose of the mental model is to create a mental simulation in mind to predict outcomes. Kleer and Brown [36], while constructing a mental model of a mechanistic device, explained that one of the purposes of a mental model is to envision. This envisioning process enables humans to predict future outcomes by inferring from current states and prior events. In the same vein, Rasmussen [48] from the domain of manual control identified the purpose of a mental model to predict future events. He referred to a mental model as an internal experiment that runs and produces results to predict. Veldhuyzen and Stassen [49], while reviewing the mental model on the concepts of manual control theory, highlighted the purpose is to estimate the "state variables" of the system that are hidden, the actions that will lead to the desired result. Norman [6] posited that one of the core functions of the mental model is to empower users to predict different outcomes of different interactions with the target system.

Performing troubleshooting: Another essential purpose of the mental model is to allow individuals to troubleshoot. Troubleshooters have to deploy their understanding of the device or system functioning to determine the fault. By acquiring a complete, robust mental model, they can successfully determine the fault and fix it.

A troubleshooter's mental model of the device contains components and functions of the components of the device. This mental model allows them to determine which faulty component is causing malfunctions. Kleer and Brown [37] explained this purpose with an appropriate example. Consider an operator of a malfunctioning nuclear power plant. A troubleshooter can not go inside the power plant and look for the error. What he can do is he can use his mental model of the reactor's functioning to conjecture and test what the fault is. Veldhuyzen and Stassen [49] refer to this action of troubleshooting as an understanding of unexpected phenomena that occur as a task progresses. Wickens [50] termed the mental model as a source of expectation. By running a mental model in the human mind's eye, an individual expects an outcome of interaction and determines when the outcome of that interaction occurs faulty or unexpected [50].

Determining actions: Mental model allows appropriate actions to achieve a task. While interacting with a system, mental model simulation appropriately maps an action and its resultant state changes. It provides the basis for creating a cause-and-effect model. People take actions to cause the change to produce desired outcomes [48, 49].

According to Norman [6], the mental model is a key part of our belief system. People act according to their belief system no matter how right or wrong the action is.

Storing knowledge: Within cognitive science, M. D. Williams et al. [51] claim that the purpose of a mental model is to serve as a mnemonic devices for remembering relations and events. From a mechanistic point of view, the mental model caches or stores the results of projection while problem-solving [37]. The implicit knowledge which is hard to articulate is stored within the mental model.

Creating a surrogate model: Theories of the mental model of a device informs us that the mental model acts as a simplified surrogate model [52]. A surrogate model

of a device follows the same working mechanism of the device but a less detailed one. Thus, it creates a connection between the states of a mental model and the states observed in the device [6].

Mirroring the belief system: Don Norman [6] proposed that a person's mental model is a mirror of a person's belief system. People act based on their belief systems. The mental models once formed act as the belief system. A person believes whatever their mental model dictates.

Facilitating Learning: De Kleer and Brown [36], in terms of understanding a new machine, speculated three kinds of learning that a mental model could facilitate. First, the mental model establishes a connection between the structure of the device and its functions which provides a coherent understanding of the device. Second, it makes the structure-function connection more robust by making implicit assumptions explicit.

Figure 2.1 summarizes the purpose of a mental model in terms of a system.

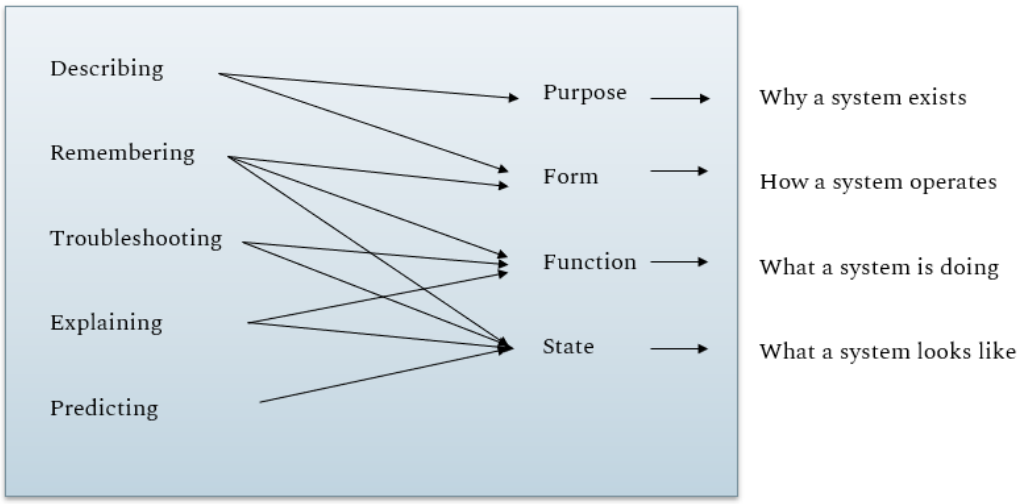


Figure 2.1: Many purpose of the mental models (diagram adapted and revised from [1]).

2.1.5 Impact of Diagrams in Mental Models

To fully grasp an abstract concept from a text involves building a mental model of the concept [32,36,53,54]. Current theories of mental models suggest that an effective teaching model is created when the model consists of visual representations of how a system looks and how it functions under various changes [32, 55–58]. According to dual coding theory extended to multimedia theory, people learn more deeply from texts with diagrams than text alone [4, 59].

Diagrams act as a powerful tool to aid the learners as they process, represent, organize, transform, and store information [60]. I know from previous studies that when students interact with appropriate visual representations depicted in textbooks, their competence in the relevant topic increases [61–63]. These benefits of diagrams have led to a significant number of studies that measured diagrams' effectiveness in improving students' understanding of the text in various textbooks [64]. In addition, diagrams can also serve as a memory aid since there is evidence that students memorize information in a text explained with diagrams more easily [60].

Mayer and Gallini [5] conducted three experiments providing participants with texts explaining three scientific devices. The group that was aided with illustrations explaining major *parts* and *state changes* (explanative illustrations) was able to recall the information more than the group using just text.

Butcher [65] conducted similar experiments to investigate learning outcomes and the influence of diagrams on novices' comprehension process. He found that diagrams, whether detailed or simple, supported mental model development, inference generation, and reduced comprehension errors. He concluded that visual representations appear to be most effective when they are designed to support the cognitive processes necessary for deep comprehension.

2.1.6 Novice programmers' mental model elicitation

In the literature, common methods to elicit mental models include verbal elicitation (e.g., interview [66–68], think-aloud protocol [69], teach-back protocol [70]), graphical elicitation (e.g., concept mapping [71], pathfinder [72]), hybrid (e.g., photo ethnography [73]), and multiple-choice questionnaires [74]. Most of those methods are time-consuming. As CS enrollment continues to increase in size, I wanted to use a method that could be feasible for classroom use. Thus, I adopted the use of a multiple-choice questionnaire in my work, purely for practical reasons.

Inspired by psychometric tests, in 2006, Dehnadi [18] devised an instrument with 12 multiple-choice questions to investigate novice programmers' mental models for primitive variable assignment. Dehnadi [18] considered each choice a mental model, with some of these choices being correct responses and others being distractors. The MCQs' distractors were placed based on the common misconceptions that can persist in a novice programmer's mental model. Distractors representing the same misconceptions were placed in multiple questions to see if a participant used the same mental model throughout the test. Dehnadi administered his test at the beginning of the semester before any classroom instructions to investigate if participants' initial knowledge state impacted their success. From participants' responses, Dehnadi analyzed the consistency of novice programmers' mental models. Dehnadi labeled his subjects' mental models consistent when subjects "used the same assignment model for all, or almost all, of the questions" [18, p.70]. When his subjects used different assignment models or unrecognizable models in different questions, he labeled them as inconsistent. Dehnadi [18] measured consistency in four levels C0, C1, C2, and C3. Level C0 demonstrates the highest rate of consistency, while sliding toward level C3 indicates a lower rate and a poorer sign of consistency. He found a significant effect of consistency on programming students' success in the course. This effect is retained even when the data is divided by previous programming experience.

Later, in 2007, Linxio Ma [17] extended Dehnadi's approach and instrument to measure the consistency of novice programmers' mental models for primitive and reference variable assignment. Ma's research goal was integrating an easier and less time-consuming mental model elicitation method into the classrooms. Ma defined a consistent mental model as "it 'always' has to match with the actual model" [17, p.35]. However, when Ma [17] measured the consistency of primitive variable assignment, he marked his participant's response as consistent if they used a consistent model to answer ten or more questions (out of 12 questions). Along with consistency, Ma [17] also measured correctness, something that Dehnadi [18] did not measure. Ma [17] formed three categories of mental models 1) consistently appropriate, 2) consistently inappropriate, and 3) inconsistent. Ma administered his test at the end of the course. Though his operational timeline differed from Dehandi's, both experiments showed the significance of consistency in a novice programmer's mental model. Ma found that the consistently appropriate group performed significantly better in CS1 final exam than the consistently inappropriate and inconsistent group. While the consistently inappropriate group performed worse than the inconsistent group for primitive variable assignment in the final exam, no statistical difference was found between those groups for reference variable assignment. Surprisingly, Ma [17] found only 17% of the participants holding viable mental models even after classroom instructions for reference variable assignment. This crucial topic is one of the basics of object-oriented programming, yet often difficult to understand for students. Later, Ma used his test as a pre-test and post-test to see the changes in the mental model after using his program visualization tool. Ma remarked that the pre-existing mental models could guide the instructors to design learning materials that could change the inappropriate mental models held by most students. Ma [17], reflecting on his findings, remarked that this poor result might be induced due to the traditional teaching approach, which does not consider students' pre-existing knowledge. The advocates of constructivism [46] state

that the design of learning materials must consider a student's pre-existing concepts and ideas.

Using Dehnadi's instrument, Radermacher et al. [75] evaluated students' pair programming performances based on their mental model consistency in a CS1 course. The authors found that students holding a consistent mental model from the very beginning performed better in paired programming tasks and individual exams [75].

Moreover, Ramalingam et al. [76] found that having developed a good mental model increased the feelings of self-efficacy of novice programmers. From their findings, they stated that previous experience, self-efficacy, and mental models contributed to students' performance in a CS1 course. The authors concluded that students' mental model development should remain a pedagogical goal in introductory programming courses.

Recently, Julie et al. [77] proposed an approach for developing a concept inventory that identifies mental models. A concept inventory is a validated, reliable, standardized multiple-choice-based questionnaire to assess students' knowledge of a set of concepts [77–80]. They focused on creating a concept inventory for variables, if statements, and functions. They included Dehanadi's [74] questionnaire in their concept inventory of variables. Their research article did not include results and validation of their instrument [77].

Inspired by Dehnadi and Ma's work, I have built an MCQ-based instrument to elicit CS1 students' mental models for arrays. Similar to their work, I placed common programming misconceptions as a distractor of each question (more details are in Chapter 4). However, there are several important differences between our instruments and theirs. First, they mapped each option of their MCQ to a mental model, whereas I believe that each distractor is mapped to an assertion. Following mental model theories, I call the set of assertions of *parts* and *state changes* components a mental model.

2.2 Knowledge Assessment Instruments

2.2.1 Origin and Develoment

Validated, standardized knowledge assessment instruments came into the practice of educational research from the earlier work of the physicists Ibrahim Abou Halloun and David Hestenes [81]. In physics education, prior work suggested that beliefs played a crucial role in introductory physics. With the motivation of assessing students' initial knowledge, Halloun and Hestenes [81] designed, developed, and validated an instrument for assessing the knowledge state of beginning physics students. They claimed their instrument could be used as a placement test, an instrument to evaluate instructions, and a diagnostic test to identify and classify misconceptions. They used their multiple choice-based instruments as pretests to assess introductory physics students' initial knowledge state.

To assess introductory physics students' initial knowledge state, Halloun, and Hestenes [81] identified the basic mechanics concepts, which are essential prerequisites for introductory physics courses. They designed the test to assess the students' qualitative conceptions of motion and identify common misconceptions noted by previous investigators. First, they created an early version of the test. It was a written test. They administered various versions of the test over a period of three years to more than 1,000 students in college-level intro physics courses. From the students' written answers, they identified the common misconceptions and used them as distractors in the multiple-choice version. By doing this, they created an easily graded test to identify common misconceptions. The authors claimed that the students' test score on this test is a measure of their qualitative understanding of mechanics. They also claimed that the test is a theoretically sound measure as the test is concerned exclusively with a systematic assessment of basic concepts. Using this instrument, the authors found that students' initial knowledge greatly affects their performance in introductory physics courses.

Hestenes et al. [82] later developed the Force Concept Inventory (FCI). The FCI was designed to improve the physics diagnostic tool from the author's earlier work. The FCI has the advantage of supplying a more systematic and complete profile of the various misconceptions. The authors again interviewed students about the questions students had missed on the inventory. While analyzing each item, they labeled some discriminators as weak. From their results, they identified major misconceptions about the concept of force. They also emphasized the importance of a well-designed and tested instrument to detect misconceptions early on.

Computing has few valid assessments for pedagogical or research purposes [83]. Tew and Guzdial [84] created the first assessment instrument for introductory computer science concepts (FCS1) that is applicable across various current pedagogies and programming languages. First, they defined the test specification - what is the test measuring (conceptual content, format of the questions, scoring procedure). Then, they identified common introductory programming concepts by analyzing CS1 textbooks and specified the contents to design the instrument. They developed a multiple-choice questionnaire instrument. Next, the instrument was reviewed by a panel of experts to provide content validity evidence. After finalizing the instrument, Tew [83] conducted empirical studies to establish validity and reliability.

The widespread use of FCS1 reached a point of saturation [85]. Parker and Guzdial [86] developed the same procedure to create the Second Computer Science I (SCS1) concept in need of a similar but new assessment instrument inventory. Later, Xie et al. [87] performed a more sophisticated statistical analysis to validate the contents of FCS1 and SCS1.

2.2.2 Validation

The validation process ensures that the instrument measures what it aims to do [88]. Through a validation process, we can iteratively build an argument to justify the applicability of an instrument. The evidence from the validation process can help us

understand if our questionnaire is too easy or too difficult for a population [89]. They can also identify if the surface features of questions are confounding the score. The results of the validation process are most effective at differentiating high and low-performing students [87]. The process can facilitate further refinement of specific questions and also the set of questions included in the test as a whole if found problematic.

According to Tew and Guzdial [84], there are two classes of evidence to claim validity: 1) content-related evidence and 2) construct-related evidence. Content-related evidence ensures the test measures all relevant parts of the subject it aims to measure. For example, if a test measures students' knowledge of elementary mathematics, the test should cover all relevant topics taught in elementary mathematics and avoid additional and irrelevant topics. Construct-related evidence ascertains that the items in the questionnaire are indeed measuring the intended constructs. It accumulates the evidence to support the interpretation of what a measure reflects [90–93]. For example, the results obtained from an instrument to probe elementary mathematics knowledge should correlate with the student's academic performance in the elementary mathematics exams. For validating the contents of their instrument FCS1, Tew, and Guzdial [84] asked a panel of experts to review their text specifications. Tew [83] also administered textbook content analysis for defining the contents of the assessment tool FCS1.

2.3 Misconceptions

Here, I review how educational researchers in other disciplines identified misconceptions. Then, I mention literature that presented studies to identify CS1 students' misconceptions. Lastly, I summarize common programming misconceptions among CS1 students found in the literature.

2.3.1 Studies of Misconceptions in Other Domains

Across many domains of educational science, researchers emphasize the identification of students' misconceptions early on. Gurel et al. [94] and Guzzetti et al. [95], in their literature review, identified numerous studies that identify students' misconceptions at the beginning of a course across multiple domains.

diSessa [96] defined misconceptions as 'false, persistent beliefs' which contradict reality. Taylor and Kowalski [79] termed a misconception 'inaccurate prior knowledge'. Studying students' misconceptions remains a pedagogical goal across various disciplines (e.g., in psychology [79,97], in physics [98], in biology [99], in computer science [29]). It is fundamental work, as research has shown that strongly held incorrect beliefs are harder to change [100] once they become fixed in a mental model, strengthening the misconceptions [101]. Researchers across various domains have agreed that acquiring knowledge of students' misconceptions early on in the course is crucial as they can be addressed through instruction [29,96,102]. Below, I describe some of the relevant studies that elicited misconceptions early on in introductory courses across multiple disciplines.

In psychology education research, Vaughan developed a true/false (T/F) test to determine introductory psychology students' misconceptions, known as the Test of Common Beliefs. Vaughan [97] administered the test to the introductory psychology class students on the first day of class. The test consisted of 80 statements representing ten conceptual contents. Incorrect statements marked as true by the students were determined to be misconceptions for each student. The author found a wide variety of misconceptions in students in the introductory psychology course. Later, Taylor and Kowalski [101] followed the same approach as Vaughan, namely a T/F test. In addition to determining the misconceptions of introductory psychology students, Taylor and Kowalski [101] also evaluated the strengths and the sources of the misconceptions. The authors developed 36 true/false questionnaire items adopt-

ing misconceptions from Vaughan's test, course contents, instructor's manuals, and common psychological myths. With each item statement, the authors asked their participants to rate their confidence in their response and asked how they learned about the information to measure the strength and source of the misconceptions. They administered the test at the beginning and at the end of the introductory psychology course. They found that introductory psychology classes reduced the number of misconceptions, as expected. They suggested instructors be mindful of the naive beliefs students bring to class. Later, Taylor and Kowalski [103, 104] found that refutational lectures and text that activate a misconception and immediately counter it with correct information significantly changed students' beliefs. Other educational researchers in the psychology domain also used (T/F) questionnaire to identify misconceptions [105].

In biological education research, Boyes and Stanisstreet [99] used a questionnaire to identify incoming freshmen students' misconceptions about the energy sources of living organisms. The questionnaire included statements about real and possible energy sources for plants and animals. For each statement, the students were asked to respond with a level of confidence that the statement was true. They reported and discussed the commonly persisting misconceptions and remarked that misconceptions might be developing in the students from their high school biology courses.

In physics education, researchers [98] believe that the intuitive beliefs of students before taking the first course impact the sources of difficulties students experience in physics. Therefore, numerous studies (e.g., [106, 107]) have been conducted to learn about physics students' misconceptions before formal instruction. The studies often involved a multiple-choice questionnaire to identify misconceptions. For example, Eryilmaz [98] used the Force Misconception Test to investigate students' misconceptions of force and motion. Halloun and Hestenes [82] surveyed college students enrolled in physics courses with their multiple choice mechanics diagnostic test, later

developed as Force Concept Inventory (FCI). In FCI, the most common misconceptions found in students' written answers from earlier studies were used as distractors of the multiple choice test.

Researchers in chemistry education also acknowledged the importance of identifying misconceptions before starting a formal course [108]. According to them, if students encounter new information that contradicts their initial conception, they might ignore, reject, or disbelieve the new information. Mulford and Robinson [108] developed a 22 multiple choice questionnaire-based chemistry concept inventory by 1) developing a content list covered by a first-semester college chemistry course and 2) placing distractors based on existing literature on introductory chemistry misconceptions. The authors then presented the frequency of participants selecting a distractor before and at the end of the semester.

Sadler et al. [102] researched 589 schools to determine the influence of teachers' knowledge on students' misconceptions about physical science. To collect data nationwide, they used a multiple choice-based questionnaire where the common students' misconceptions were placed as distractors. Sadler [109] remarked that when the items are written to include common misconceptions as distractors, the questionnaire serves well in diagnosing misconceptions. Sadler et al. [102] found that teachers who know their students' most common misconceptions are more effective than teachers who do not know.

2.3.2 Studies of Misconceptions in Introductory Programming

In the Computer Science education domain, there have been fewer studies of misconceptions in mental models when compared to other science education domains. Nevertheless, CS educators value the importance of finding programming misconceptions [7, 8, 29, 110]. According to Sorva [29], when programming misconceptions persist, students may struggle to appreciate further instruction and learn from it if the instructor does not utilize misconception-sensitive teaching. Kurvinen et al. [111]

remarked that finding the most relevant programming misconceptions would help teachers and lecturers address the main obstacles faced by the students. On this note, Sorva [29] also suggested knowing how the students commonly view the programming concepts and utilizing tests to assess students' prior knowledge. Keeping in mind the need for knowledge about students' programming misconceptions, Chiodini et al. [112] created an inventory of programming misconceptions.

Though CS educators value the importance of learning novice programmers' misconceptions, few researchers define a *misconception* accurately [112]. According to a literature review [110], some studies considered syntax error misconceptions, and some presented language-independent misconceptions. Chiodini et al. [112] provided an actionable definition of a programming misconception. According to them,

> "A programming language misconception is a statement that can be disproved by reasoning entirely based on the syntax and/or semantics of a programming language". [112, p.381]

They emphasized that misconceptions must be tied to a programming language; otherwise, one often cannot conclude that certain assertions are wrong. I also believe that a programming misconception should be language-specific.

Chiodini et al. [112] provided a list of papers studying programming misconceptions from the year 1983 to 2021. None of the studies mentioned identified programming misconceptions before classroom instruction. Caceffo et al. [113] used an open-response test to identify CS1 students' misconceptions at the end of the semester. Swidan et al. [8] identified school students' misconceptions of Scratch. They used Sorva's [114] list of Java misconceptions and created a multiple choice-based questionnaire with the applicable misconceptions as distractors. They recorded each wrong response of their participant as a misconception.

As shown in this section, researchers have used quantitative methods to identify misconceptions. Other researchers have also used qualitative approaches (e.g., think-

aloud, interviews, drawing, videotape, observation) [7, 28, 115, 116] to identify misconceptions. Because I envision our work being used in intro courses with large enrollments, I have adopted a quantitative approach to make it easy to administer.

2.3.3 Common misconceptions of CS1 students

In a study, Sorva [28] found a student believing that an assignment statement of primitives (e.g., `number2 = number` where both variables are of a primitive type) to be equivalent to an assignment of reference variables. A misconception that a primitive variable can hold multiple values at a time was reported by various studies. Students were also found to believe that values swap when one primitive variable is assigned to another (inverted assignment). Novice programmers tend to have a misconception that primitive variables do not have a default value. Moreover, some students believe that the natural language semantics of variable names (e.g., a variable named `books`) affects the value of a variable it can store.

The misconceptions of the primitive variable assignment also exist when a literal or another variable is assigned to an array's index as an element. In addition, confusion regarding the array's indexing [7, 111, 117], especially the start and ending index, is common among CS1 students. Moreover, novice programmers tend to think an array of elements are untyped [112].

In Java, arrays are manipulated with reference variables. Hence, misconceptions regarding reference variables are also tied to it. Sorva [28] and Ma [17] found that a reference variable does not hold a reference but a set of object properties instead. Moreover, Holland et al. [118] found that some students believe once a variable references an object, it will always refer to it. Regarding a reference variable or object declaration, Kaczmarczyk et al. [7], and Sorva [119] found that novice programmers tend to believe memory is allocated for objects after declaration. Whereas novices thought no memory is allocated for an object when the object is instantiated [7]. One more misconception that novice programmers commonly hold is reference variable

assignment copies objects [120].

In my work, I utilized these existing common misconceptions to create questions to probe if these misconceptions exist in CS1 students' mental models before classroom instruction of arrays.

In the next chapter, I present a study that analyzed the diagrams of commonly used CS1 textbooks. In the process of analysis, I identified the *parts* and *state changes* components of arrays.

CHAPTER 3: TEXTBOOK REPRESENTATION OF ARRAY'S PARTS AND STATE CHANGES

This chapter presents the results of a study that has been published in the Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE), 2020, in Trondheim, Norway. Full citation can be found here [121].

3.1 Introduction

Textbooks, whether they are online or traditional, are the most reliable source of knowledge a student seeks [122, 123]. Students consider the information in textbooks as reliable and trustworthy. Since textbooks have been the main didactic tool in many educational systems worldwide [124], their analysis could provide valuable information about a factor affecting students' construction of knowledge. In recent years, textbook analysis has been done in Computer Science to assess the quality of programming examples and to understand how thoroughly important programming concepts are covered [125–127]. As diagrams impact a mental model more than the text [65], I investigated textbook diagrams to understand novices' mental models in terms of what *parts* and *state changes* are shown with diagrams. In the initial stage of study, I analyzed the textbook diagrams of three programming concepts: *variables*, *arrays*, and *objects*.

Theories of mental models [32, 55–57] have led Mayer and Gallini [5] to pinpoint two features of diagrams[1] that help learners build runnable mental models - *parts* and *state changes*[2]. Here, *parts* refers to the major components of the scientific concept

[1] I am using "diagrams" as a more common alternative to the term "illustrations", which is the term used by Mayer and Gallini

[2] Mayer and Gallini [5] used the term *Parts* to refer to major *parts* and *State changes* to refer to

or device. And *state changes* represents each major state of the components and the changes of the states. My research aims to explore what *parts* and *state changes* are provided in the diagrams of introductory programming textbooks for variables, arrays, and objects. I surveyed 15 commonly used introductory computer science (CS1) Java textbooks and analyzed diagrams of these fundamental programming concepts. In this paper, I summarize my findings based on the following questions:

RQ What major *parts* and *state changes* are shown in the textbook diagrams and how?

3.2 Mayer's Decomposition of *Parts* and *State Changes*

Mayer and Gallini [5] propose two types of decomposition of a concept, when portrayed with diagrams, could help learners build a mental model of a scientific device or concept: **parts** and **state changes**. By *parts* Mayer and Gallini [5, p. 715] refer to each major component within the structure of a system. This is exemplified in the work undertaken by Mayer and Gallini as the major *parts* of a braking system include the tube, wheel cylinder, smaller piston, brake drum, and brake shoe. In addition to portraying the *parts*, Mayer and Gallini [5] believed the portrayal of each major state that each component can be in and the corresponding state changes also influences the mental model.

In the context of a braking system, the major *state changes* were exemplified in the paper as "before braking" and "after braking" states for each of the major components such as the tube, smaller pistons, brake shoes, and wheel. They also emphasized on the portrayal of the relational state change, such as how the state change of the brake shoe causes changes in the wheel and other components.

The presence of these two features makes the diagrams **explanative**. By conducting three experiments Mayer and Gallini [5] concluded that explanative diagrams improve mental models.

---

major *state changes*

Though Mayer and Gallini tested this framework for spatial, mechanical, and scientific systems, I found this framework suitable to analyze how variables, arrays, and objects are depicted in terms of their *parts* and *state changes* because programming is a scientific process which entails a systematic structure. Similar to a system, the components of a programming concept interact with one another in a defined way specified by the programming language's semantics.

3.3 Methodology

3.3.1 Defining Programming Concept's *Parts* and *State Changes*

The first step in my analysis was to define the *parts* and *state changes* for the three concepts I studied: variables, arrays, and objects. I then investigated how the *parts* and *state changes* are portrayed in the diagrams found in popular CS Introductory textbooks. This section provides my proposed definition of *parts* and *state changes* for my three studied programming concepts.

3.3.1.1 The *Parts*

For a primitive variable, I considered the *parts* to be the *name*, the *value* stored, the *memory location* of the variable, possibly the amount of memory occupied by the variable (*size*), and the corresponding *code* that defines/uses the variable.

For an array, I considered the *parts* to be the *name* of the array, the *reference* (pointer) to the array elements, the *elements* that form the array, the *indices* of the individual slots, the *memory location* of the array, and the corresponding *code* that creates/uses the array.

For an object, the *parts* were the *name* of the object, the *reference* (pointer) to the object, the *fields* of the object, the *methods*, the *memory location* of the object, and the corresponding *code* that creates/uses the object.

3.3.1.2 The *State Changes*

For defining the major *state changes*, I created a list of states that a variable, array, and object can have.

Variables

In my proposed framework, the *state changes* for *variables* includes the following states:

Declaration: Declaration is the state of a variable when you declare it with a given type (e.g., `int number;`). When a declaration statement is processed at run-time, memory for that variable is allocated, and the default values are set.

Initialization: I counted as initialization the step when a variable is initialized to a value other than the default value, and it happens as part of the variable creation/declaration:

`int number = 5;`.

Assignment: I considered an assignment any time when the variable value is changed using an assignment (=) operator (e.g., `number = 7;`) after declaration.

Arrays

In my framework, the State changes for *arrays* includes the following steps:

Declaration: When I declare an array, for example

`double[] itemPrice;` a reference variable is created with a default value of `null`.

Instantiation: I am referring to the instantiation state when allocating an array with a statement such as:

`itemprice = new double[10];`

This statement allocates sequential blocks of memory, assigns default values to them, and stores the reference to the block in the variable (e.g. `itemprice`).

Assigning values: This state shows the change in elements when an element is assigned new values, for example:

`itemprice[7] = 12.7;`.

Objects

In my framework, the State changes for *objects* includes the following steps:

Declaration: The creation of an object begins with declaring the object's reference variable, such as:

`Clock myClock;` where `Clock` is a class. At runtime, this statement creates a reference variable `myClock` with a default value of `null`.

Instantiation: When an object is instantiated using *new*:

`myclock = new Clock();` memory is allocated for that object and the reference variable `myClock` now points to the new object. This object's fields will get initialized with default values or with other computed values depending on the class constructor.

Field Assignment: This state captures the object's fields being changed through some assignment operation, by calling a setter method, or as a side effect of another method call.

Assignment: This state represents a reference variable change to point to another object, such as

`myClock = yourClock;`

3.3.2 Textbook Selection

With the framework presented above, I set out to analyze diagrams in introductory textbooks. To obtain a representative sample of commonly used introductory Java textbooks, I used surveys of educators and looked at best-selling lists. I posted a survey on the SIGCSE members mailing list on 31st July 2019 and a poll in Facebook's

CS Education discussion forum group. I also looked at Amazon's and Barnes & Noble's best-seller programming textbook lists [128, 129]. I filtered these lists, selecting textbooks for novices with no prior programming experience. From these sources, I selected books that appeared in multiple lists (e.g., mentioned in the survey and appearing in a best-seller list) and highly recommended books (e.g., multiple votes in one of the polls or high on one of the best-selling lists). In addition, I wanted to include an electronic textbook, as those are becoming more popular.

To avoid interpreting my analysis as a vote in favor of a particular textbook or even as an evaluation of the quality of the textbook, I have given a random identifier to each book (e.g., B1, B2, etc.) and used that identifier to refer to the book in the paper. All 15 books used are cited in the references [130–144] but there is no relationship between the identifier used in the paper and the textbook.

3.3.3 Data Collection

For each textbook, I examined the chapters where each concept (e.g., variable of primitive type, array, and object) was introduced. If the textbook did not have a designated chapter for a concept, I examined the section where the concept was first defined.

For each of the concepts, I recorded whether a textbook provided a diagram or not. If the textbook provided a diagram for the concept, then to capture the *parts* components, I focused on the first diagram provided in the textbook, typically when the concept was introduced. I tallied each of the *parts* components if the book presented the *parts* in their diagram.

For analyzing the *state changes*, I examined all the diagrams in the chapter, not just the first one. I looked for the depiction of states and an explanation of state changes for each concept. Furthermore, I looked to see if the diagrams included the code that produced the state change.

The results are shown in Tables 3.1, 3.2, and 3.3 and discussed in the next section.

Table 3.1: *Parts & State changes* components of Primitive Variables.

Parts	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**	**B8**	**B9**	**B10**	**B11**	**B12**	**B13**	**B14**	**B15**
Name	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Value	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Memory Location							✓		✓	✓					
Size							✓				✓				
Code	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
State changes															
Declaration	✓		✓			✓			✓	✓					
Initialization	✓		✓			✓			✓	✓					
Assignment	✓	✓	✓	✓			✓		✓						

Table 3.2: *Parts & State changes* components of Arrays.

Parts	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**	**B8**	**B9**	**B10**	**B11**	**B12**	**B13**	**B14**	**B15**
Name	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓			✓
Reference	✓	✓	✓		✓	✓		✓		✓	✓	✓			
Elements	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓
Indices	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
Memory location									✓						
Code	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓			✓
State changes															
Declaration											✓				
Instantiation	✓		✓		✓	✓		✓	✓		✓				
Assigning values	✓		✓		✓	✓		✓	✓		✓				

3.4 Results

3.4.1 Variables

Twelve (12) of the fifteen (15) textbooks illustrated primitive variables. Variables were usually represented as a rectangular box with a value in it and the variable name on the left side. B7 presented sequential boxes as memory slots and put values on the side to demonstrate how many memory slots are occupied by the variable. B9 also included sequential memory location with values in it.

3.4.1.1 Parts

According to my framework, the components of variables that could be depicted included variable name, value, memory location, and size. All 12 books labeled variable name and value in their diagrams (see Table 3.1). Only B7 and B9 presented variables with memory location as slots of sequential boxes. B10 presented a memory address at the left of the variable box, depicting the variable name on top. B7 also annotated the size of the variable in the diagrams, as well as B11.

3.4.1.2 State changes

Table 3.1 lists the program states a variable can have and shows which books had illustrations of which program states. The books are checked off if they contain illustrated states after explaining the execution of the equivalent code. Eight out of 12 books explained and portrayed some states of a variable. The remaining four books either had code annotation and diagrams or just diagrams with no explanation of the states. I describe the descriptions of the program states and books that portray them below:

Declaration: Only 5 out of 15 books (B1, B3, B9, B6, B10) presented this state in their introductory chapters. Four of them show the box empty, which inaccurately suggests that there are no default values assigned. Only B10 presented 0 as a default value of an `int` variable.

Initialization: The same five books contained illustrations of the state change of a variable after initialization by showing the execution of equivalent code. Other books did not contain illustrations of the initialization state.

Assignment: Along with declaration and initialization, B1, B3, and B9 also showed changes in the diagrams when an initial value was changed to a different value. B2, B4, and B7 did not present previous states and only showed the variable state after an assignment. The remaining nine books did not provide any illustration of this state. B1, B3, and B9 also provided an illustration of the change after executing a statement that caused a change on a variable.

3.4.2 Arrays

Thirteen textbooks visually represented an array as a horizontal set of boxes or slots, along with indices. Some books included a small reference box pointing to the long horizontal rectangle. Figure 3.1 shows a representative example of the diagrams found in the textbooks.

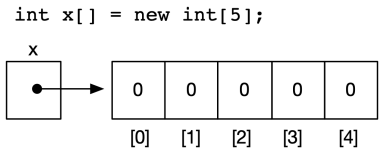


Figure 3.1: Representative example of a diagram for arrays found in some textbooks.

3.4.2.1 Parts

Twelve textbooks showed indices in their diagrams (see Table 3.2). Eight textbooks provided a detailed diagram showing names, references, elements, and indices in their diagrams. B12 did not illustrate elements and memory locations but provided the other three components. Only B9 visually represented memory blocks of an array along with names, elements, and indices. Only B7 visually represented an array

without any corresponding code; the rest of the books provided equivalent code to explain their diagram.

3.4.2.2 State changes

Only seven textbooks explained some of the following program states after code execution with diagrams; the remaining eight books either did not provide any diagrams, or they provided only the Parts (see Table 3.2):

Declaration: Only B11 illustrates this state. Though B11 portrayed this important state, it showed no default value for the reference variable.

Instantiation: This state was visually represented by seven books. While most of them have shown the appropriate default value of the elements after instantiation, B5 and B6 used blanks in the slots of the elements. The other eight books did not provide any visual explanation of this state.

Assigning values: All of the seven textbooks that portrayed instantiation also illustrated how an array looks after assigning values to particular slots.

Table 3.3: Parts & State changes of Objects.

Parts	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**	**B8**	**B9**	**B10**	**B11**	**B12**	**B13**	**B14**	**B15**
Name	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓		✓
Reference	✓	✓	✓		✓			✓		✓	✓		✓		
Fields	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓		✓
Methods	✓		✓								✓				
Memory location									✓						✓
Code	✓	✓	✓		✓			✓	✓	✓	✓		✓		✓
State changes															
Decalaration									✓				✓		✓
Instantiation		✓	✓		✓			✓	✓	✓			✓		✓
Field assignment	✓				✓			✓	✓	✓					✓
Assignment		✓	✓					✓	✓						

3.4.3 Objects

I found diagrams of objects in 12 of the 15 books. Objects were visually represented with boxes that contained fields and methods and a reference pointing to the box (see Figure 3.2). B15 and B9 depicted objects differently, using vertical slots of memory with the reference variable stored in one of the memory slots pointing to the address block where the fields were stored. B6 portrayed objects as Universal Markup Language (UML) diagrams with subclasses and fields with values within the class diagrams.

```
class Ticket {
   Integer price;
   void method1() {};
   void method2() {};
   void method3() {};
}

Ticket concert = new Ticket();
```

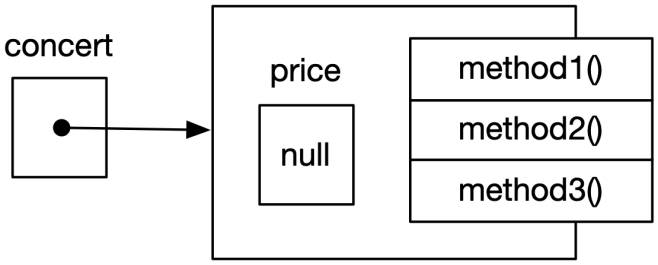


Figure 3.2: Representative example of a diagram for objects found in some textbooks.

3.4.3.1 Parts

All of the 12 books depicting objects showed the name and fields of an object in their diagrams (see Table 3.3) except B10, which left out the fields. The object reference was also depicted in all of them except B6 and B9. B1, B3, and B11 were the only books showing all major components of an object: name, reference, fields, and methods. Only B9 and B15 illustrated memory locations as part of an object. Two books did not provide any corresponding code for their diagrams.

3.4.3.2 State changes

Nine of 15 books explained some states of an object with code and diagrams.

Declaration: This state was visually represented by only three books (see Table 3.3).

Where B13 and B9 showed null as the default value, B15 showed a question mark (?) in the reference.

Instantiation: Only eight textbooks illustrated this state. Other than B2 and B3, all 8 visualized the object state when it is instantiated with default values. B8 showed the instantiation both with a default value and other values by calling a constructor. B3 instantiated the object by initializing fields with other values. While B8 and B9 provided appropriate field-specific default values, B13 did not provide any default value, B10 displayed blanks in the fields, and B15 showed question marks (?).

Initialization: Four textbooks visually represented this state. B5 and B10 portrayed the change in fields by initializing them. B15 and B9 initialized fields with set methods.

Field assignment: Six books illustrated changes in fields after an assignment operation on fields, either by method call, constructor, or direct assignment.

Object Assignment: Four textbooks portrayed this change by showing the reference of one object now pointing to another object after the assignment. The other 11 books did not portray this state change.

3.5 Discussion

Variables: Most textbooks presented variables by depicting the name, value, and code associated with the variable. Few, however, showed how variables are related to memory (location and size). Only three of the textbooks had four of the five components from the *parts* of variables. For the *state changes*, only three textbooks showed all states.

Arrays: Array coverage is more uniform across all the textbooks analyzed. Twelve of the fifteen books cover at least four components from the *parts*. The depiction of

state changes for arrays is also pretty uniform, but only about half of the books show at least two of three states.

Objects: Object coverage is much more sparse than the other two concepts. I postulate that this might be because of the nature of the introductory textbooks; object behavior is an advanced topic and thus less covered than variables and arrays.

This variability of the *parts* depiction shows inconsistency among the books. For example, diagrams from some books presented references to arrays, but others left them out. This type of inconsistency and wide representations in the visual representations of programming concepts has also been found in pedagogical animation or debugging tools [145].

Overall, my analysis showed more coverage of the *parts* than the *state changes*. Also, most textbooks that showed a default state after creating a variable used a question mark (?) or blank as the default value. The Java language has clearly defined default values for each data type, and these depictions are, therefore, misleading. This kind of misrepresentation can contribute to one of the common misconceptions regarding variables: variables of primitive type have no default value [7].

Though most of the textbooks provided diagrams of these concepts, in terms of Mayer and Gallini's definition, not a single textbook portrayed an explanative diagram showing all of the major components of *parts* and *state changes* for variables, arrays, and objects. Based on Mayer et al.'s work, it appears that textbooks are missing an opportunity to help students build a runnable mental model of these basic concepts. Authors can utilize textbooks as a tool to help students become more autonomous in trying to understand programming rather than rely on other sources (e.g., lectures and tutorials) by incorporating explanative diagrams. As one-to-one guidance is becoming less available in CS1 courses because of high enrollment, textbooks, program visualization tools, and debuggers can provide quality content to improve students' learning and decrease the shortcomings of classrooms.

3.6 Limitations

The implications of my work face the following limitations. First, my sample of books may not be representative of textbooks most frequently used in CS1. The Amazon and Barnes & Nobles bestseller lists of Java textbooks only indicate how many people bought these books, not whether they were used in a CS1 class. Also, I could not obtain some of the newest editions of books and considered older editions in my analysis. I limited my study to only Java textbooks in this research because diagrams can vary with language semantics. For now, I defined the *parts* and *state changes* in terms of Java semantics.

Another limitation is that Mayer et al.'s work entailed spatial and often mechanical systems. The concepts explained in the diagrams of my study are abstract concepts. How do you depict data type? How do you depict memory locations? How do you show state changes for a system with no moving parts? These questions can be answered via graphical rendering that might not be obvious to the students, thus possibly obfuscating the diagrams themselves.

Finally, the context and purpose of where the diagrams are used also must be considered when evaluating their effectiveness. For example, the first time an array is shown in a textbook, it might have all the details proposed in my work. But later in a textbook, as arrays are used to explain other concepts (e.g., sorting), the diagram might focus on other concepts. The array, in those examples, might be one part of a larger example. In those cases, the diagram can be drawn in a simplified manner, abstracting out some of the details. Similarly, the memory address of the variables might not be normally depicted in the books in my study but might be essential in books that cover system architecture, for example.

3.7 Conclusion

In this chapter, I proposed a framework of *parts* and *state changes* to analyze diagrams in introductory computer science textbooks based on Mayer's framework for *explanative* diagrams. I found this framework suitable to analyze how variables, arrays, and objects are depicted in 15 introductory programming textbooks.

My work has certain implications and suggestions for diagrams. In Appendix B, I describe the development of *explanative* diagrams of arrays covering all the parts of my proposed framework. The particular graphical properties used (position, color, shade, etc.) can differ in other renderings of this diagram, but the *parts* and *state changes* components must be depicted to meet my framework (and Mayer et al.'s definition of explanative diagrams).

My analysis found a wide variability in how these concepts are depicted in the textbooks. This suggests that the diagrams for variables, arrays, and objects shown in many CS1 textbooks fall short of meeting the conditions for an explanative diagram. The shortcomings of diagrams motivated me to understand the shortcomings of novice programmers' mental models. Therefore, for my doctoral work, I aimed to investigate novice programmers' mental models based on the breakdown of array's *parts* and *state changes* mentioned in this chapter. With this intent, I adopted a questionnaire-based approach to elicit and analyze novice programmers' mental models. The next chapter presents my approach to develop the questionnaire.

CHAPTER 4: THE MENTAL MODEL TEST: AN INSTRUMENT TO ELICIT MENTAL MODELS

4.1 Introduction

In the previous chapter, I described my study to find the shortcomings of textbook diagrams depicting fundamental CS1 concepts: primitive variables, arrays, and objects. In our College of Computing and Informatics at UNC Charlotte, the CS1 course is offered in a combination of these two courses: ITSC 1212: Introduction to Computer Science I and ITSC 1213: Introduction to Computer Science II. At the time of my data collection, arrays were taught at the end of our CS1 course ITSC 1212: Introduction to Computer Science I using the programming language Java. Students start the following course, ITSC: 1213, by reviewing arrays. Therefore, understanding arrays is crucial in assessing whether students can advance into a CS2 course. Moreover, arrays have certain characteristics that are in common with primitive variables and objects. Arrays are manipulated with reference variables similar to an object. Assigning values to an array's element works the same way as a primitive variables assignment. Therefore, the understanding of students' mental models of arrays might help us to perceive their mental models of primitive variables and objects.

My approach to eliciting novice programmers' mental models adopts a multiple-choice questionnaire to ensure two criteria of mental models: correctness and consistency. In this chapter, I describe the development and deployment of my questionnaire-based approach to elicit mental models: the Mental Model Test of Arrays (MMT-A).

4.2 Definitions and Examples

This section reiterates the definitions of related terms that I use to develop the MMT-A. Though the definitions are stated in previous sections, here, I am highlighting them to clarify their scope with examples:

Mental Model: As mentioned in Section 2.1.1, I am implementing one combined mental model definition from two sources. While forming the mental models, I am using the definition that a mental model is a set of assertions [35]. I will represent novice programmers' mental models with assertions in their minds. These assertions were elicited based on the array's *parts* and their behavior (*state changes*) [31, 37, 146, 147]. In summary, I aim to elicit novice programmers' assertions of the array's each *part* and *state changes* components.

For example, let's consider the mental model of the moon. I have a series of assertions on the moon's structure and states.

Factual assertions of the moon's structure (*parts*):

1. The moon's shape is an oblate spheroid.
2. The moon's color is grey.
3. The primary composition of the moon's surface is rocks and dust.

Factual assertions of the moon's *state changes*:

1. The moon reflects the sun's light.
2. The moon orbits around the Earth.
3. The moon is visible at night.

These assertions are a subset of correct factual assertions that will form the accurate mental model of the moon. MCQ questions have the following components: 1) a

question stem, 2) a correct answer, and 3) distractors [148]. A possible MCQ question to elicit a learner's mental model of the moon's *parts* can be:

> **Question Stem:** What is the primary composition of the Moon's surface?
>
> **Answer(X) and Distractors:**
>
> a) Iron.
>
> b) Silicon.
>
> c) Helium.
>
> d) Rock and dust. (X)

Here, the correct answer (option d) was generated from the correct assertion *the primary composition of the moon's surface is rocks and dust.*. The distractors are generated from plausible misconceptions and thus are incorrect assertions, such as *the primary composition of the moon's surface is iron.*

Similar multiple-choice questions can be developed for eliciting learners' mental model assertion related to the moon's *state changes*. An exemplar question can be:

> **Question Stem:** Why does the moon appear to shine?
>
> **Answer(X) and Distractors:**
>
> a) The moon produces its own light.
>
> b) It reflects sunlight. (X)
>
> c) It reflects the earth's light.
>
> d) It absorbs sunlight and releases it at night.

Here, the common misconceptions regarding the moon's source of light are placed as the distractors.

Correctness: A mental model is correct when it is faithful to the behavior of the actual device [37]. Correctness will ensure which portions of students' mental models are correct and which are incorrect. For example, accurate answers to all of the questions regarding the moon will prove correct mental models for the moon.

Consistency: A mental model is consistent when there is no internal contradiction [37]. To clarify, a mental model is consistent where the assertions do not contradict each other. Two assertions contradict each other when they can not be true at the same time [35]. In his approach to proving contradiction, Laird [35] took an approach of negating the assertions. As stated earlier, a mental model is a collection of assertions. In my context, the set of assertions is within the components of a system's *parts* and *state changes*. According to Laird [35], to determine the contradiction, we need to choose any assertion in the set. Afterwards, we try to prove its negation from the remaining assertions in the set. If a negation is found, then the set of assertions is inconsistent.

For example, I have two assertions about the moon's shape: 1) the moon is a circle, 2) the moon is a square. From the assertion *the moon is a square*, I can derive its negation that *the moon is not a circle*. Since I found a negation by following Laird's approach to determine inconsistency, these two assertions can not be true at the same time; thus, they are contradictory. If a mental model includes no contradictory assertions, then the mental model is deemed consistent. The opposite will be deemed as inconsistent.

I utilized these definitions and approaches to elicit novice programmers' mental models of arrays.

4.3 Development of MMT-A

As stated above, I am eliciting the mental model as a set of assertions for each component. From the textbook content analysis (mentioned in Chapter 3), I derived array's *parts* components to be *name*, *type*, *index*, and *elements*. The *state changes* components include *declaration*, *instantiation*, *assigning elements*, and *assignment*.

I listed the correct factual assertions of each component. For example, the correct assertions of a state change *declaration* are:

1. After declaration, an array reference variable is created.

2. After declaration, the default value of the array reference variable is set to null.

Then, I began to list incorrect assertions. I utilized the literature on misconceptions [7, 17, 28, 119, 149–152, 152–155] and my teaching experience to enumerate incorrect assertions. I generated some incorrect assertions by inferring from the misconception of other related concepts. For example, Kaczmarczyk et al. [7] reported finding that students believed memory is allocated for Objects that have been declared but not instantiated. Since an array declaration is the same as an object declaration, I utilized this misconception as an incorrect assertion.

After enumerating the correct and incorrect assertions for each concept, I developed the questionnaire where these assertions would be placed as options. Here, I placed the correct assertions as keys and incorrect assertions as distractors. I used only four choices per question, following current pedagogical conventions. Each distractor maps to an incorrect assertion, as shown in Figure 4.1. In some cases, a distractor is mapped to multiple assertions based on the context. Most of the wrong assertions were placed as distractors in multiple questions to measure the consistency of the mental model. For example, the wrong assertion that 'indexing starts with 1' is placed in eight questions as a distractor. Students holding a misconception about the array 'indexing starts with 1' will have multiple opportunities to select this distractor in several questions. This helps us to identify the misconceptions in a student's mental model. Not every component has equal sets of incorrect assertions and questions (shown in Table 4.1), as some components (e.g., *name* or *type*) have fewer documented misconceptions than others.

Additionally, to retain construct validity, I only aimed to develop questions that elicit the mental models of arrays. Thus, I avoided incorporating other concepts (e.g., nested loop) into the questionnaire.

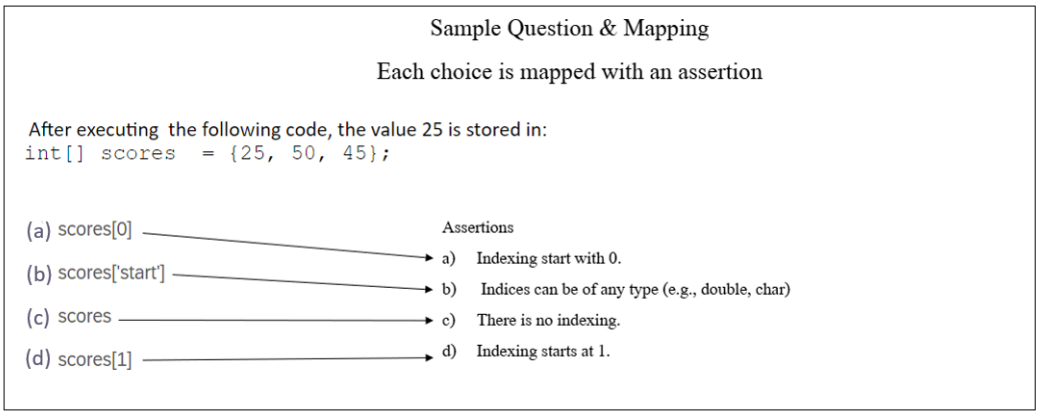


Figure 4.1: An example showing the mapping between each choice and the corresponding assertion.

Table 4.1: List of mental model concepts, concept types and the number of questions in which they appear.

Concept Type	**Concept Name**	**Num. of Questions**
Part	Name	2
Part	Index	5
Part	Type	2
Part	Element	5
State Change	Declaration	6
State Change	Instantiation	4
State Change	Assigning Elements	6
State Change	Array assignment	6
Total		36

4.4 Collection of Mental Model Assertions

I mapped each participant's multiple-choice selection (response) to an assertion, as shown in Figure 4.1. For example, if a participant selects option (d) (scores[1]), the participant's recorded assertion for this question is 'MI4: Array index starts at 1'. Thus, for each participant, I collected the selected assertions for all of the 36 questions of the MMT-A.

As each assertion could appear multiple times, I counted:

- *How many participants always selected an assertion.* The frequency distribution of the counts will allow me to hypothesize about the correctness and consistency of novice programmers' mental models. The majority of the participants always selecting the correct assertion can inform us that most students are learning. On the contrary, if the majority of the participants always select the wrong assertion, it can indicate that a misconception persists among the group of students.

- *How many participants sometimes selected an assertion.* This measure can give us an estimation of uncertainty in their mental models. If a participant selected an assertion only once while having multiple opportunities, this could indicate the uncertainty of their mental models and, in some cases, inconsistency.

- *How many participants never selected an assertion.* The frequency distribution of the counts will allow me to hypothesize which assertion is not present in the participants' mental models. For example, if the majority of the participants never selected a wrong assertion, this can indicate they don't have that incorrect assertion in their mental models.

I collected the three frequency counts for each assertion from the data to investigate novice programmers' mental models. In chapter 5, I report the selection frequency for each assertion before a classroom instruction of arrays. In chapter 6, I report the selection frequency for each assertion after a classroom instruction of arrays. The frequencies show what assertions are present in their mental models. In my work, the selection of assertions will allow me to measure and analyze the characteristics of their mental models.

4.5 Measurement of Consistency

The first step to measure the consistency of a mental model is to aggregate participants' assertions for each component of the *parts* and *state changes*. According to

Laird's definition of consistency [35], a mental model is defined as consistent when the assertions residing in the mental model do not contradict each other; that is, there are no two assertions can not be true at the same time. By following this definition, I determined the contradiction among a set of assertions. First, I grouped the assertions based on the components. For example, all the assertions related to the array's *part* component *index* are considered as a set of assertions of *index*. Then, I determined which assertions are contradictory to each other.

For example, let us consider the incorrect assertion that *'array indexing begins with 1'*. If I follow the definition of consistency, I will conclude that it contradicts with *'indexing starts with 0'*. The two assertions *'indexing starts with 0'* and *'indexing starts with 1'* cannot be true at the same time. Therefore, I mark these assertions as contradictory assertions.

By following Laird's [35] definition, I devised steps to determine if two assertions can not be true at the same time. The steps are:

Step 1: Group the assertions based on the *parts* and *state changes* components.

Step 2: Take one assertion from the n assertions in the set.

Step 3: Take another assertion from the n-1 assertions in the set.

Step 4: Logically compare the two assertions with each other.

If they can not be true at the same time, mark them as contradictory.

Otherwise, take another assertion from the set.

Step 5: Follow step 4 as long as all the comparisons are not done.

Step 6: Repeat Steps 2, 3, and 4 until all of the assertions in the set are compared with each other.

I listed the contradictory assertions for all *parts* and *state changes* components in a marking sheet (see Appendix C).

After collecting each participant's assertions mapped from their choices, I grouped the sets of assertions based on *parts* and *state changes* components. For example, a participant's assertions related to *index* are grouped together. Then, I utilized my marking sheet to determine if there was a contradiction in the set of the participants' assertions. If I find a contradiction, I label the participant's mental model of *index* *inconsistent*, otherwise *consistent*. I follow a similar approach to label a participant's mental model consistency for each *part* or *state changes* components.

4.6 Measurement of Correctness

Measuring correctness is the same as measuring the accuracy of the answers. If a participant chooses a correct answer, it reflects the correctness of the participant's mental model. The test had pre-determined correct answers, and I graded participants' responses against the correct answers. I gave the participants a score of 1 for a correct answer and a 0 for an incorrect answer. The question was not scored if a student did not answer a question (left the question blank). The correctness score was the number of questions answered correctly as a percentage of all questions answered.

4.7 Classification of Mental Models

In this section, I describe an integrated model to classify mental models based on consistency and correctness. A mental model can be labeled as consistent and inconsistent based on consistency. Correctness can include 1) incorrect, 2) mostly correct, 3) mostly incorrect, and 4) correct. Based on both consistency and correctness, I classify a participant's mental model into one of 7 categories (see Table 4.2):

- *Inconsistent and Incorrect (II):* When a participant's response contains no correct answer, I labeled the mental model as incorrect. Additionally, if some of the participant's incorrect assertions contradict each other, the mental model will be labeled as inconsistent and incorrect.

- *Consistent and Incorrect (CI):* When a participant's response contains no cor-

rect answer, and the incorrect assertions do not contradict each other, then labeled the mental model as consistent and incorrect.

- *Inconsistent and Mostly Incorrect (IMI):* I refer as mostly incorrect a mental model that has more wrong answers than right answers. I also label it as 'mostly incorrect' if there are the same number of right and wrong answers. If there is evidence of contradiction in the assertions and it is mostly incorrect, then the mental model is labeled IMI.

- *Consistent and Mostly Incorrect (CMI):* I term a mental model as CMI when a mostly incorrect mental model (see previous bullet) showed no evidence of contradictions among assertions.

- *Inconsistent and Mostly Correct (IMC):* I label a mental model as mostly correct when a participant's responses contain more right answers than wrong answers. A participant's mental model is thus labeled IMC when there is evidence of contradiction in the assertions.

- *Consistent and Mostly Correct (CMC):* A participant's mental model is labeled consistent and mostly correct if the responses have more right answers than wrong and there is no evidence of contradictions in the assertions.

- *Correct (C):* I categorize a participant's mental model as correct when all the responses are correct. When the answers are 100% correct, there can not be any inconsistency. Therefore, a correct mental model will always be consistent. I label this category as simply correct (C).

A correct mental model will have a mental model rank 6 (see Table 4.2). As the correctness and consistency decrease, the rank will become lower, with the lowest ranking being incorrect and inconsistent. I use this classification of mental models to label and rank mental models for each component of *parts* and *state changes*.

Table 4.2: A classification (ranking) of mental models based on correctness and consistency.

Consistency	Inconsistent	Consistent	Inconsistent	Consistent	Inconsistent	Consistent	Consistent
Correctness	Incorrect		Mostly Incorrect		Mostly Correct		Correct
Abbreviation	II	CI	IMI	CMI	IMC	CMC	C
Ranking	0	1	2	3	4	5	6

4.8 Measurement of a Mental Model

I use the mental model classification presented in the previous section and shown in Table 4.2 to rank each participant's mental models of the *parts* and *state changes* components. Here, the highest mental model score a participant can achieve is 48, ranking 6 for all the eight components of *parts* and *state changes* and the lowest to be 0; incorrect and inconsistent mental model for all eight components. For example, consider a participant with a consistent and mostly correct (CMC) mental model (rank: 5) for the part *index* and *type*, a mostly incorrect inconsistent mental model (rank: 2) for the state changes *declaration*, *instantiation*, and *assignment*, and mostly incorrect consistent mental model (rank: 3) for the part *name*, *element*, and state changes *assigning elements*. Then, I calculated the participant's total mental model score as (5+5+2+2+2+3+3+3) = 25.

4.9 Identification of Misconceptions

As discussed before, I placed several assertions in multiple questions as options so that participants had multiple opportunities to select these incorrect assertions. A participant choosing an incorrect assertion all the time when it was available resulted in that assertion being labeled as a misconception. I identified which assertions were marked as a misconception and how many participants were found to hold the misconception. In Chapter 8, I present the misconceptions identified in novice programmers' mental models before classroom instruction and after. I also describe the change in misconception after having classroom instruction.

4.10 Example: A Case Study

In this section, I present a case study demonstrating how I analyzed a single participant's mental model. I refer to this case as participant *X*.

First, I mapped participant *X*'s responses to the corresponding pre-defined assertions (as shown in Figure 4.1). Then, I grouped the assertions based on the *parts* and *state changes* components. As each assertion is presented more than once, a participant can choose an assertion multiple times. Table 4.3 lists participant *X*'s chosen assertions and the percentage of selection of that particular assertion. Next, I analyzed the contradiction between assertions with the marking sheet, which lists the contradictory assertions to measure the consistency of the mental model based on the method described in Section 4.5.

For the part *name*, I found only one assertion in the mental model: *'MN2: Array type (e.g., int[]) is the name of the array'*. This assertion was presented twice in the MMT-A, and participant *X* selected the option corresponding to this assertion both times. Since I found no contradictory assertions regarding the component *name*, I marked the mental model for *name* as consistent.

I computed the correct answers for measuring correctness, as mentioned in Section 4.6. For the part *name*, participant *X* answered all the questions wrong. Therefore, I assigned the correctness score as 0 (reported in Table 4.4).

Afterward, based on consistency and correctness, I classified and ranked the mental model of array's *name*. As I found the mental model to be consistent and incorrect, I classified the mental model of the array's *name* to be consistent and incorrect (CI) (as per Section 4.2). Thus, the mental model rank for the array's part *name* was 1.

I performed a similar analysis for the part *index*. I obtained four assertions from participant *X*'s response (listed in Table 4.3). Assertions MI1.1, MI3, and MI4 are in the group of contradictory assertions that were pre-determined in the marking sheet. Thus, I marked them as contradictory assertions. Similarly, I marked assertions MI3

and MI5 to be contradictory. Based on my approach to determine mental model consistency (see Section 4.5), I determined participant X's mental model of the *index* to be inconsistent.

Then, I investigated the correctness score of participant X for the concept *index*. Table 4.4 depicts that participant X answered all the questions related to *index* incorrectly. Therefore, I determined the correspondence score of *index* to be 0. As participant X answered all the questions regarding *index* wrong and the mental model is inconsistent, I classified the mental model of the *index* to be inconsistent and incorrect with rank 0 (reported in Table 4.4).

I conducted a similar analysis with all the rest of the assertions and answers obtained from participant X's response. Lastly, I added the mental model ranks of each component and calculated a total mental model score. I calculated participant X's mental model score to be 16 (out of 48).

Following the above-mentioned approach, I computed the mental model score of our entire data set. These results allow us to understand the mental model of the entire data set, which can also be analyzed with different demographics. The detailed results are presented in the later chapters.

Besides mental model consistency, correctness, and classification, mental model assertions from the responses also allow me to identify misconceptions. I labeled an incorrect assertion as a *misconception* when that assertion appeared in more than one question in the MMT-A, and when at least one student chose the assertion 100% of the time.

Based on my definition of misconception (see Section 4.9), Table 4.3 shows participant X has selected one assertion related to the concept *name* 100% of the times while it was presented twice in the MMT-A. Therefore, I marked the assertion *'MN2: Array type (e.g., int[]) is the name of the array'* as a misconception of participant X.

Table 4.3: Participant X's obtained (partial) mental model assertions grouped with the corresponding concept, marked contradiction, consistency status, and percentage of selecting the corresponding assertion.

Concepts	Assertions obtained from participant X	Contradictory to each other	Consistency	Percentage % (Selected/Presented)
Name	MN2: Array type (e.g. int[]) is the name of the array. (incorrect)		Consistent	100%
Index	MI1.1: Indexing starts with 0. (correct)	xx	Inconsistent	11.11%
	MI3: There is no indexing. (incorrect)	xx yy		50%
	MI4: Indexing starts at 1. (incorrect)	xx		87.50%
	MI5 Indexing ends with n. (incorrect)	yy		75%

Table 4.4: Participant X's (partial correctness and consistency status along with mental model classification and ranks.

Concepts	Number of questions	Number of correct answers	Correctness score	Consistency*	Mental model classification	Mental model rank (0-6)
Name	2	0	0	Consistent	Consistent and Incorrect (CI)	1
Index	5	0	0	Inconsistent	Inconsistent and Incorrect (II)	0
...	...	...	...	...	...	...
Total	-	-	-	-	-	16

Note: Asterisks(*) refer that mental model consistency was obtained from Table 4.3.

4.11 MMT-A is not a Concept Inventory

The approach to developing MMT-A has some similarities with the development approach of a conception inventory (CI) (details of CI are described in Section 2.2). The similarities are as follows: 1) CI is a multiple-choice-based questionnaire similar to MMT-A, 2) CIs place common misconceptions as distractors similar to MMT-A.

However, our purpose and aim to develop MMT-A differentiates it from a CI. The purpose of MMT-A is to elicit learners' mental models and is developed utilizing the definitions from the theories of mental models. CIs intend to assess knowledge, not mental models. The second key difference is that MMT-A is developed by decomposing a programming concept- *arrays* into *parts* and *state changes*. The development of a CI does not follow this approach. Thus, I claim MMT-A as an approach to

Table 4.5: Data Collection Timeline

Dataset	Semester	Course	Timeline	Number of Participants
Spring 2021-Pre-test	Spring 2021	ITSC: 1212	Before instruction	113
Spring 2021-Post-test	Spring 2021	ITSC: 1212	After instruction	101
Spring 2023-Post-Instruction	Spring 2023	ITSC: 1212	After instruction	85
Fall 2023-Qualitative	Fall 2023	ITSC: 1212 ITSC: 1213 ITSC: 2214	After instruction	10

elicit mental models. Even though, it has some similarities with a CI, I do not claim MMT-A to be a concept inventory.

4.12 Deployment of MMT-A

I administered MMT-A to collect data from novice programmers in various timelines (see Table 4.5). The study was approved by the Institutional Review Board (IRB) (IRB Protocol 21-0067). Below, I summarize the timeline of the data collection:

Spring 2021-Pre-test: At first, I administered MMT-A to collect students' responses in Spring 2021 in the course *ITSC 1212: Introduction to Computer Science I* at UNC Charlotte. The course *ITSC 1212* is the first course in the Computer Science major at the University of North Carolina at Charlotte and introduces the Java programming language to the students. The course curriculum followed a traditional CS1 course with labs and activities following a Media Computation approach [143]. I collected data in the Spring 2021 semester. Four instructors taught all 21 sections of the course. Due to the COVID-19 global pandemic, all sections were taught online and synchronously. The author of this dissertation was neither an instructor nor a TA of the course.

The CS1 course follows a common curriculum that is centrally coordinated. The course contents, textbook, and course materials were identical for all sections. Because I was interested in the mental model of novice programmers before they get instruction

in arrays and to explore whether prior experience made a difference or not, I collected data with the MMT-A BEFORE they had instruction on arrays in the course. I collected data during the 6th week of a 14-week semester after the course covered the concepts of data types, literals, primitive variables, assignment, operators, casting, classes, objects, constructors, method calling, parameters, and if statements. The Qualtrics link of the MMT-A was embedded in each section's learning management system (Canvas). Students completed the test as part of their required homework. Students received 2 points for completing the test. As a preamble of the test, the instruction stated that there were no points for answering the correct answer, nor was there a penalty for answering incorrectly. Students were informed that the assignment was not assessing their knowledge of Java but rather interested in their intuitions. Out of 248 students who completed the MMT-A, 113 gave us their consent to analyze their data. After removing incomplete and duplicate cases, I analyzed 93 participants' responses.

Spring 2021-Post-test: To document how novice programmers' mental models changed after formal classroom instruction, I administered the MMT-A in the same sections of ITSC 1212 on week 13. I received 187 responses in this round of data collection. However, among them, only 101 participants gave their consent to analyze their data. When I paired each individual participant's responses with the pre-test, I obtained 66 data points (paired pretest-posttest).

Spring 2023-Post-Instruction: To collect more data and to understand more how the novice programmers' mental models changes after classroom instruction, I again administered MMT-A in Spring 2023. I attempted to collect data from our two CS1 courses *Introduction to Computer Science I* and *Introduction to Computer Science II* and one CS2 course ITSC 2214: Data Structures and Algorithms. As per the study design, participation in my study was completely voluntary. Instructors had the choice to give the students participatory points or make it a required activity.

Unfortunately, I only received responses from the *ITSC 1212* course. MMT-A was administered among 11 sections of *ITSC 1212*. Six instructors taught all 21 sections of the course. All the sections were running in person. The data was collected on week 13 of a 14-week semester when all students received formal instruction on arrays. The course curriculum followed a traditional CS1 course with labs and activities following CS Awesome AP CS A Java Course [156]. A total of 85 participants responded to the MMT-A with their consent.

Summer 2023-Post-Instruction: Furthermore, I administered MMT-A in Summer 2023 from the sections of *ITSC 1212* and *ITSC 1213*. Like the previous studies, I collected data from *ITSC 1212* when students were given formal instructions on arrays. The course curriculum and material of *ITSC 1212* was the same as Spring 2023. The data was collected from two sections of *ITSC 1212*; one instructor was in charge of them. I received 26 data from *ITSC 1212* course.

Moreover, I collected data in the first week of *ITSC 1213*. *ITSC 1213* course is the follow-up course of *ITSC 1212*, where students start the course by revising arrays. I collected data from two sections of *ITSC 1213*, taught by two instructors. I obtained 50 responses from *ITSC 1213* course.

Fall 2023-Qualitative: To ensure participants interpret the MMT-A questions correctly and the choices truly represent the mental model assertions I mapped, I collected data from a think-aloud [157] qualitative approach with the MMT-A. I interviewed ten novice programmers who have learned arrays while asking to answer a subset of questions of MMT-A while performing think-aloud. The study details are presented in Chapter 10.

4.13 Conclusion

In this chapter, I describe the development, deployment of MMT-A. MMT-A has been used to elicit novice programmers' mental model assertions. I analyzed the mental model assertions based on consistency and correctness. The analysis led me

to classify and rank mental models. With the ranks, I scored the mental models. In the next chapters, I present the results collected from the deployment of MMT-A among different subjects in various timelines.

CHAPTER 5: INCOMING NOVICE PROGRAMMERS' MENTAL MODELS

5.1 Introduction

CS1 students come with a wide variety of programming backgrounds, experiences, and misconceptions. According to the theory of constructivism, learners build up new knowledge on existing similar models [39]. Therefore, learners utilize their existing knowledge to build the next further meaningful understanding. As the purpose of my dissertation is to investigate novice programmers' mental models, it is crucial for me to study CS1 students' initial mental models to document and address the gaps in their knowledge. Perceiving their knowledge gap ahead of time might help focus the target instruction.

In this chapter, I report the findings based on the following research questions:

RQ1. What are the characteristics of incoming novice programmers' mental models?

RQ2. How are the incoming novice programmers' initial mental models of array's *parts* components in comparison with the *state changes* components?

RQ3. What impact do prior programming experience and demographics have on the mental model of the participants in our study before classroom instruction?

Based on my findings, most incoming novice programmers exhibit correct and consistent mental models for array components *name* and *type*. However, for array's *parts*: *index*, *elements*, and all the *state changes* components: *declaration*, *instantiation*, *assigning elements*, and *assignment* most students exhibit incorrect and inconsistent mental models. Moreover, the results show that the participants' mental model *consistency* and *correctness* score (labeled as the mental model score) is significantly

higher for *parts* than for *state changes*. Additionally, the participants with prior programming knowledge held more *consistent* and *correct* mental models than those who did not have prior knowledge. Moreover, male participants' mental model *consistency* and *correctneess* scores were significantly higher than the female's, providing further evidence of the preparatory gap across genders in computing.

5.2 Data Collection

Because I was interested in the mental model of novice programmers before they get instruction in arrays and to explore whether prior experience made a difference or not, I am using the data set **Spring 2021-Pre-test** (details mentioned in Section 4.12). The test took place during the 6th week of a 14-week semester after the course covered the concepts of data types, literals, primitive variables, assignment, operators, casting, classes, objects, constructors, method calling, parameters, and if statements. The Qualtrics link of the MMT-A was embedded in each section's learning management system (Canvas). Students completed the test as part of their required homework. Students received 2 points for completing the test. As a preamble to the test, the instruction stated that there were no points for answering the correct answer, nor was there a penalty for answering incorrectly. Students were informed that the assignment was not assessing their knowledge of Java but rather interested in their intuitions.

5.3 Participants

Out of 248 students who completed the MMT-A, 113 gave me their consent to analyze their data. After removing incomplete and duplicate cases, I could include 93 participants' responses in our analysis. The majority identified as male (71%), 20.4% identified as female. Of the 93 participants, 52.7% reported having CS as their major of study. Over half of the participants (54.8%) had no prior programming experience before enrolling in the CS1 course. Among the 46 students who had prior programming experience, 12 participants had experience with Java, and 33 participants had

experience with other programming languages (e.g., Python, JavaScript, C#, C++, SQL, HTML, Snap).

5.4 Results

Sections 5.4.1 enumerate the assertions residing in novice programmers' mental models of an array. I present the correctness and consistency of incoming novice programmers' mental models in Sections 5.4.2, 5.4.3, and 5.4.4. Lastly, the effect of prior programming experience and demographics on novice programmers' mental models is discussed in Section 5.4.5.

5.4.1 Mental Model Assertions

As previously described, concepts in the mental model of arrays are broken down into *parts* and *state changes*. Below, I present the findings for each concept. Each section contains two tables. The first shows all the assertions for each concept with an indication of whether the assertion was correct or incorrect and the number of times each assertion appears in the MMT-A. The second table shows the results of the participant selections for each assertion. It indicates if the assertion was *always selected*, *sometimes selected*, or *never selected*. It is worth noting that for assertions that are correct (e.g., assertion MN1 in Table 5.1 and Table 5.2), I expect participants always to select it, as that indicates a correct/consistent mental model of the concept. However, for assertions that are incorrect (e.g., assertion MN2 in Table 5.1 and Table 5.2), the expected result is for the participant to *never select* it.

5.4.1.1 Assertions for *Parts*

This section presents the results for all the *parts* components of the array mental model included in the MMT-A.

Part: Name

The assertions MN1, MN2, and MN3 appeared in two questions each, and MN4 and MN5 appeared in only one question each (see Table 5.1). The results show that

69 out of 93 students selected MN1 when it appeared as the correct option, and an additional 12 students selected sometimes. For the incorrect choices, a total of 75 students avoided (i.e., never selected it) MN2, 92 avoided MN3, 88 avoided MN4, and 93 (all 100%) avoided MN5. Table 5.2 summarizes the frequency distribution for each kind of selection.

Table 5.1: List of assertions for the *part* **name**.

Assertions	Correctness	Num. of Questions
MN1: Array reference variable is the name of the array	Correct	2
MN2: Array type is the name of the array	Incorrect	2
MN3: Keyword new is the name of the array	Incorrect	2
MN4: Whatever comes after equal sign in an initialization is the name of the array	Incorrect	1
MN5: First element in the array is the name of the array	Incorrect	1

Table 5.2: Frequency distribution of the selection of assertions for the *part* **name**.

	MN1 correct	MN2 incorrect	MN3 incorrect	MN4 incorrect	MN5 incorrect
Always selected	74.2% (69)	12.9% (12)	0% (0)	5.4% (5)	0% (0)
Sometimes selected	12.9% (12)	6.5% (6)	1.1% (1)	0% (0)	0% (0)
Never selected	12.9% (12)	80.6% (75)	98.9% (92)	94.6% (88)	100% (93)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

Part: Index

Table 5.3 lists the assertion used in our MMT-A. The frequency distribution of each assertion is presented in Table 5.4. For *indices*, I have broken down the correct assertion into two separate assertions, MI1.1 and MI1.2. The assertions MI1.1 appeared nine times and MI1.2 6 times. Together, those two are the correct assertions regarding the use of indices in arrays. The other incorrect assertions are listed in Table 5.4. It is worth noting that MI4 and MI5 represent common misconceptions on the use of array *indices*, namely that the index of the first element in the array is [1] and that the last element is stored at [n] instead of the correct answer of [n-1].

The correct assertion of MI1.1 was selected consistently by 28% of the participants. In addition, 71% (66) of the participants sometimes selected this option. Surprisingly, the complimentary assertion of MI1.1, MI1.2, had a different distribution than MI1.1. None of the participants selected it consistently, and an additional 73.1% (68) selected it some of the time (at least once).

The incorrect choices show a high percentage of participants who avoided those options. MI2 was never selected consistently; 3.2% of the participants selected it at least once, and 96.8% of the participants avoided it altogether. Table 5.4 shows similar numbers for MI3, MI6, and MI7.

MI4 and MI5, however, have different numbers. These assertions reflect a misunderstanding of how indices work in Java. They reflect an understanding that the index starts at 1 (MI4) and ends at n instead of n-1 (MI5). The start/end of an array is a documented misconception [7, 111], so it is no surprise that these incorrect assertions drew higher responses from participants. Only 4 participants selected MI4 consistently (all eight times that it appeared), but 62.4% (58) of the participants selected it at least once. The number of participants was lower for MI5, with 2.2% (2) of the participants consistently selecting this option (all seven times that it appeared), and 66.7% (62) of them selecting at least once.

Table 5.3: List of assertions for the *part* **index**.

Assertions	Correctness	Num. of Questions
MI1.1: Array index starts with 0	Correct	9
MI1.2: Array index ends with n-1 (index of last element)	Correct	6
MI2: Index of an array can be of any type, not just integers	Incorrect	3
MI3: There is no indexing into the array	Incorrect	2
MI4: Array index starts with 1	Incorrect	8
MI5: Array index ends with n (index of last element)	Incorrect	7
MI6: Array index does not map to its corresponding location in the array	Incorrect	3
MI7: Students think the index is the element.	Incorrect	3

Table 5.4: Frequency distribution of the selection of assertions for the part **index**.

	MI1.1 correct	MI1.2 correct	MI2 incorrect	MI3 incorrect	MI4 incorrect	MI5 incorrect	MI6 incorrect	MI7 incorrect
Always selected	28% (26)	0% (0)	0% (0)	2.2% (2)	4.3% (4)	2.2% (2)	0% (0)	1.1% (1)
Sometimes selected	71% (66)	73.1% (68)	3.2% (3)	17.2% (16)	62.4% (58)	66.7% (62)	10.8% (10)	26.9% (25)
Never selected	1.1% (1)	26.9% (25)	96.8% (90)	80.6% (75)	33.3% (31)	31.2% (29)	89.2% (83)	72% (67)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

Part: Type

The assertions MT1, MT2, and MT3 each appear in two questions in the instrument (see Table 5.5). The correct assertion, MT1, represents a clear understanding of the difference between array *name* and array *type* in an array declaration statement. Table 5.6 shows the frequency distribution of each assertion. The results show that 89.2% (83) of the participants selected MT1 consistently. Only 2 participants avoided MT1 both times it appeared in the instrument. A handful of participants (9) selected MT2 sometimes. These students seemed to confuse the array *name* with the array *type*.

Table 5.5: List of assertions for the *part* **type**.

Assertions	Correctness	Num. of Questions
MT1: Name appearing before the array name is the type of the array	Correct	2
MT2: Name of the array is the type of the array	Incorrect	2
MT3: Keyword new is the type of the array	Incorrect	2

Part: Element

The assertions ME1.1 and ME1.2 are the correct assertions regarding array *elements*. Each of these appeared 2 times in the instrument. Assertions ME2 (5), ME3 (4), ME4 (4), ME5(2), ME6 (2), and ME7 (2) were invalid assertions about array elements (see Table 5.7). As shown in Table 5.8, ME1.1 (size of the array) was selected

Table 5.6: Frequency distribution of the selection of assertions for the *part* **type**.

	MT1 correct	MT2 incorrect	MT3 incorrect
Always selected	89.2% (83)	1.1% (1)	0% (0)
Sometimes selected	8.6% (8)	9.7% (9)	1.1% (1)
Never selected	2.2% (2)	89.2% (83)	98.9% (92)
Total	100% (93)	100% (93)	100% (93)

consistently by only 3.2% (3) of the participants. In addition, an additional 96.8% (90) of the participants selected this option at least once. ME1.2 was on the other hand, a very common choice with 75.3% (70) participants selecting it consistently. Interestingly, 21.5% (20) of the participants never selected this option. This assertion (ME1.2, elements stored in the array can be of the same time) seem to divide the participants in the two extremes, 75.3% consistently selected, and 21.5% never selected it.

Worthy of note is option ME5. This (erroneous) assertion appeared twice in the instrument and implied that the name of an array was semantically related to the value stored in the array. For example an array of type String and named "books" can only store strings containing names of books, such as "Harry Potter". The results show that 12.9% (12) of the participants selected this option consistently and an additional 6.5% (6) of the participants selected this assertion some of the time.

5.4.1.2 Assertions for *State Changes*

This section presents the results for all the *state changes* of the array mental model included in the MMT-A. The sections that follow cover the *state changes* in our instrument: *declaration* (section 5.4.1.2), *instantiation* (section 5.4.1.2), *assigning elements* (section 5.4.1.2) and *assignment* (section 5.15).

Table 5.7: List of assertions of the part **element**.

Assertions	Correctness	Num. of Questions
ME1.1: The array contains n number of elements	Correct	7
ME1.2: Elements stored in the array can be only of the declared type	Correct	2
ME2: The array contains size+1 elements	Incorrect	5
ME3: The array contains size-1 elements	Incorrect	4
ME4: After instantiation, the array doesn't have space to store any elements (size 0)	Incorrect	4
ME5: Element values and array names are related (e.g., books and "Harry Potter")	Incorrect	2
ME6: Keyword 'new' is an element in the array	Incorrect	2
ME7: Type of values stored do not match type of array	Incorrect	2

Table 5.8: Frequency distribution of the selection of assertions for the part **element**.

	ME1.1 correct	ME1.2 correct	ME2 incorrect	ME3 incorrect	ME4 incorrect	ME5 incorrect	ME6 incorrect	ME7 incorrect
Always selected	3.2% (3)	75.3% (70)	0% (0)	0% (0)	0% (0)	12.9% (12)	2.2% (2)	1.1% (1)
Sometimes selected	96.8% (90)	3.2% (3)	61.3% (57)	30.1% (28)	4.3% (4)	6.5% (6)	1.1% (1)	6.5% (6)
Never selected	0% (0)	21.5% (20)	38.7% (36)	69.9% (65)	95.7% (89)	80.6% (75)	96.8% (90)	92.5% (86)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

State change: Declaration

The assertions MD1.1, MD1.2, MD1.3, and MD1.4 are the correct assertions for array *declaration* (see Table 5.9). The assertion MD1.1 was selected consistently by 38.7% (36) of the participants, with an additional 14% (13) participants selecting MD1.1 sometimes (see Table 5.10). The remaining 47.3% (44) of the participants never selected MD1.1. These results indicate a high level of misunderstanding among the participants in our study.

I observed similar results for MD1.2, with participants almost equally distributed between always selecting it (41.9%), sometimes (33.3%), and never selecting it (24.7%). MD1.3 and MD1.4 each appeared only once in our instrument, and as a result, participants either selected that assertion (MD1.3 47.3% and MD1.4 29%) or not (MD1.3

52.7% and MD1.4 71%).

For the incorrect assertions, MD2 stands out as 21.5% (20) of the participants consistently selected it. An additional 12.9% (12) of the participants selected this option at least once. At the other extreme, 65.6% (61) of the participants never selected this option.

The remaining incorrect assertions (MD3, MD4, MD5, MD6, and MD7) were mostly selected sometimes or not selected at all. Still, these assertions should not have been selected (i.e., incorrect assertions). The distribution of responses shows that these assertions were selected, if not always, sometimes. Thus, this concept is problematic for lots of students.

Table 5.9: List of assertions of the *state change* **declaration**.

Assertions	Correctness	Num. of Questions
MD1.1: After declaration, default value of array reference variable is set to null	Correct	2
MD1.2: After declaration, an array reference variable is created	Correct	2
MD1.3: After declaration, no memory is allocated for the array	Correct	1
MD1.4: After declaration, no elements can be stored.	Correct	1
MD2: There is no default value for the elements of the array (blank/no value)	Incorrect	2
MD3: The default value for the array reference is the default value for type (e.g., int is 0, boolean is false)	Incorrect	4
MD4: The default value for the array reference is stored as '?'	Incorrect	3
MD5: After declaration, memory is allocated for the elements	Incorrect	3
MD6: After declaration, the number of elements that can be stored is unlimited	Incorrect	3
MD7: After declaration, there is a default size for an array	Incorrect	1

State change: Instantiation

As shown in Table 5.12, MIn1.1 was selected consistently by 53.8% (50) of the participants. The rest of the participants, 46.2% (43), never selected this option. A second valid assertion in this concept, MIn1.2 had no participants selected consistently. About half of the participants, 50.5% (47), selected this option some of the time with 49.5% (46) of the participants never selecting this option.

Table 5.10: Frequency distribution of the selection of assertions for the *state change* **declaration**.

	MD1.1 correct	MD1.2 correct	MD1.3 correct	MD1.4 correct	MD2 incorrect	MD3 incorrect	MD4 incorrect	MD5 incorrect	MD6 incorrect	MD7 incorrect
Always selected	38.7% (36)	41.9% (39)	47.3% (44)	29% (27)	21.5% (20)	0% (0)	4.3% (4)	4.3% (4)	2.2% (2)	7.5% (7)
Sometimes selected	14% (13)	33.3% (31)	-	-	12.9% (12)	38.7% (36)	34.4% (32)	49.5% (46)	64.5% (60)	0% (0)
Never selected	47.3% (44)	24.7% (23)	52.7% (49)	71% (66)	65.6% (61)	61.3% (57)	61.3% (57)	46.2% (43)	33.3% (31)	92.5% (86)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

MIn2 most participants (67.7%) never selected this option and 29% of the participants selected it sometimes. Only a handful (3.2%) selected this option consistently. Similarly, MIn3 was selected by 41.9% (39) of the participants sometimes and 48.4% (45) never selected it. Only a few participants, 9.7% (9), selected MIn3 consistently. For MIn4, no participant selected it consistently. Just over half of the participants, 52.7% (49), selected it sometimes and the rest, 47.3% (44), avoided this option.

Table 5.11: List of assertions for the state change **instantiation**.

Assertions	Correctness	Num. of Questions
MIn1.1: After instantiation, memory is allocated for the array	Correct	1
MIn1.2: After instantiation, the appropriate default value is assigned to the elements	Correct	4
MIn2: After instantiation, '?' is stored as a default value	Incorrect	3
MIn3: After instantiation, there is no default value (blank) stored for the elements	Incorrect	4
MIn4: After instantiation, no memory is allocated	Incorrect	4

Table 5.12: Frequency distribution of the selection of assertions for the state change **instantiation**

	MIn1.1 correct	MIn1.2 correct	MIn2 incorrect	MIn3 incorrect	MIn4 incorrect
Always selected	53.8% (50)	0% (0)	3.2% (3)	9.7% (9)	0% (0)
Sometimes selected	0% (0)	50.5% (47)	29% (27)	41.9% (39)	52.7% (49)
Never selected	46.2% (43)	49.5% (46)	67.7% (63)	48.4% (45)	47.3% (44)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

State change: Assigning Elements

Table 5.13 shows all the assertions for the concept, *Assigning Elements* with two correct and four incorrect assertions (see Table 5.14). For the correct assertions, MAE1.1 was never selected consistently. Most participants, 87.1% (81), selected this assertion some of the time, and 12.9% (12) of the participants avoided it completely. For the other correct assertion, MAE1.2, a third of the participants, 33.3% (31), selected it consistently, and an additional 43% (40) participants selected it sometimes. 23.7% (22) participants avoided this option.

For the incorrect assertions, MAE2, MAE3, MAE4, and MAE5, zero (0) participants selected these options consistently. For MAE2 and MAE3, the participants were divided almost evenly between selecting it sometimes (MAE2 58.1% and MAE3 51.6%) and avoiding it altogether (MAE2 41.9% and MAE3 48.4%).

MAE4 and MAE5 appeared only twice in the instrument. For MAE4, no participant selected it consistently, 16.1% selected it some time, and 83.9% avoided the option altogether. For MAE5, 9.7% selected it consistently, an additional 6.5% selected it sometimes, and the majority of the participants, 83.9%, never selected this option.

Table 5.13: List of assertions for the *state change* **assigning elements**.

Assertions	Correctness	Num. of Questions
MAE1.1: Assignment copies the value from right to left.	Correct	6
MAE1.2: The variable on the right-hand side remains the same after assigning.	Correct	2
MAE2: The value of a variable never changes.	Incorrect	8
MAE3: A variable can hold multiple values at a time / 'remembers' old values.	Incorrect	10
MAE4: Assignment swaps values of the left and right hand side.	Incorrect	2
MAE5: Primitive assignment is the same as reference assignment.	Incorrect	2

Table 5.14: Frequency distribution of the selection of assertions for the *state change* **assigning elements**.

	MAE1.1 correct	MAE1.2 correct	MAE2 incorrect	MAE3 incorrect	MAE4 incorrect	MAE5 incorrect
Always selected	0% (0)	33.3% (31)	0% (0)	0% (0)	0% (0)	9.7% (9)
Sometimes selected	87.1% (81)	43% (40)	58.1% (54)	51.6% (48)	16.1% (15)	6.5% (6)
Never selected	12.9% (12)	23.7% (22)	41.9% (39)	48.4% (45)	83.9% (78)	83.9% (78)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

State change: Array Assignment

Table 5.15 lists all assertions related to the *state change array assignment*, also labeled as *assignment* in short. MA1, the correct assertion, only had 2.2% (2) of the participants selecting it consistently. An additional 57.0% (53) of the participants selected this option sometimes, and 40.9% (38) of the participants avoided this option.

The first of the incorrect assertions, MA2, had 89.2% (83) of the participants avoiding this option, and the remainder of the participants 10.8% (10), selected it consistently. The next incorrect assertion, MA3, had a fairly even distribution of participants, with 34.4% (32) selecting it consistently, 45.2% (42) selecting it sometime, and 20.4% (19) avoiding it.

The last two incorrect assertions, MA4 and MA5, did not have any participant select it consistently. The participants were divided between selecting it sometimes (MA4 22.6% and MA5 24.7%), with the majority in both cases avoiding it altogether (MA4 77.4% and MA5 75.3%).

Table 5.15: List of assertions for the *state change* array **assignment**.

Assertions	Correctness	Num. of Questions
MA1: Array assignment copies reference from right to left.	Correct	3
MA2: Array assignment appends value at the end of the array.	Incorrect	1
MA3: Array assignment copies the values.	Incorrect	3
MA4: Array assignment transfers (cuts) values.	Incorrect	2
MA5: Array assignment copies the reference but does not share memory.	Incorrect	2

Table 5.16: Frequency distribution of the selection of assertions for the state change **array assignment**

	MA1 correct	MA2 incorrect	MA3 incorrect	MA4 incorrect	MA5 incorrect
Always selected	2.2% (2)	10.8% (10)	34.4% (32)	0% (0)	0% (0)
Sometimes selected	57% (53)	0% (0)	45.2% (42)	22.6% (21)	24.7% (23)
Never selected	40.9% (38)	89.2% (83)	20.4% (19)	77.4% (72)	75.3% (70)
Total	100% (93)	100% (93)	100% (93)	100% (93)	100% (93)

Table 5.17: Participant's overall correctness and mental model score with total scores for *parts* and *state changes*. The statistically significant difference between the correctness score and mental model score of arrays *parts* and *state changes* is shown in the last row.

	Correctness Score		Mental Model Score	
	Mean (%)	SD	Mean (%)	SD
Overall	53.75	5.94	61.54	6.45
Parts	68.64	2.99	71.29	4.46
State changes	44.27	3.67	51.79	3.03
Parts vs. State changes	$p < 0.001$		$p < 0.001$	

5.4.2 Mental Model Correctness

I scored each participant based on the correctness of their answers, as discussed in Section 4.6. I referred to this score as the *correctness score*. As there were 36 questions, the possible highest score could be 36, and the lowest score could be 0. I report on the total correctness score and the correctness score for each *part* and *state change* component.

For the total correctness score, the participants answered an average of 19.35 (53.75%, $\sigma = 5.94$, N = 93) questions correctly (see Table 5.17). A participant's minimum correctness score is six, and the maximum score is 32.

Moreover, I computed the correctness score for the *parts* and *state changes*. I had

Table 5.18: Participants' correctness, consistency, and mental model classification for each *part* and *state changes*.

Components	# Qs.	Correctness		Consistency				Mental Model Classification											
				Consistent		Inconsistent		II		CI		IMI		CMI		IMC		CMC		C	
		Mean % (N=93)	SD	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%
P:Name	2	80.65	35.38	81	87.10	12	12.90	0	-	12	12.90	12	12.90	0	-	0	-	0	-	69	74.20
P:Index	5	50.11	41.98	42	45.20	51	54.80	16	17.20	9	9.70	23	24.70	0	-	9	9.70	6	6.50	28	30.10
P:Type	2	93.55	19.82	84	90.30	9	9.70	3	3.20	1	1.10	6	6.50	42	45.20	0	-	0	-	41	44.10
P:Elements	5	72.47	26.20	87	93.50	6	6.50	1	1.10	2	2.20	4	4.30	5	5.40	1	1.10	45	48.40	35	37.60
S:Declaration	6	47.49	24.69	85	91.40	8	8.60	0	-	1	1.10	8	8.60	55	59.10	0	-	23	24.70	6	6.50
S:Instantiation	4	35.48	28.39	72	77.40	21	22.60	6	6.50	13	14.00	14	15.10	43	46.20	1	1.10	10	10.80	6	6.50
S:Assigning Elements	6	49.82	29.84	63	67.70	30	32.30	3	3.20	2	2.20	24	25.80	32	34.40	3	3.20	21	22.60	8	8.60
S:Assignment	6	41.40	25.49	26	28.00	67	72.00	3	3.20	5	5.40	41	44.10	18	19.40	23	24.70	1	1.10	2	2.20

14 questions for *parts* and 22 questions covering *state changes*. The mean correctness score for *parts* was 9.61 (68.64%, $\sigma = 2.99$, N = 93) and for *state changes* was 9.74 (44.27%, $\sigma = 3.67$, N = 93).

Table 5.18 (column 3) shows the detailed breakdown of the correctness score for the concepts of parts and state changes. I present the mean correctness score in percentage as the number of questions varies for each concept. Note that participants' mean correctness scores were higher for *parts* than for *state changes*. A paired t-test shows that participants scored significantly higher ($t = 13.14, p < 0.001$) in *parts* than in *state changes* (shown in Table 5.17).

5.4.3 Mental Model Consistency

Based on the contradictions in a participant's mental model, I categorized mental models into two categories: consistent and inconsistent (see Section 4.5 for the definition). I measured mental model consistency for each *part* and *state change*. Table 5.18 (columns 4 and 5) includes the absolute frequency (N) and relative frequency (%) obtained from the analysis of consistency.

The majority of the participants (more than 50%) had a consistent mental model for *name* (87.1%, 80), *type* (90.3%, 84), *elements* (93.5%, 87), *declaration* (91.4%, 85), *instantiation* (77.4%, 72) and *assignment of elements* (67.7%, 63).

Two of the components of *parts* and *state changes*, however, resulted as inconsistent based on the participants' answers. The majority of the participants had an

inconsistent mental model for *index* (54.8%, 51) and array *assignment* (72%, 67).

5.4.4 Mental Model Score: Combining Correctness & Consistency

As described in Section 4.7, I categorized and ranked a mental model based on its consistency and correctness score. The correct mental model (see Table 4.2), which is always consistent, is ranked the highest (6). As a mental model rank is given for each *part* and *state changes*, a total mental model score is calculated by adding the mental model ranks for the components of each *part* and *state changes*. There are, in total, eight components. Therefore, the highest mental model score a participant can earn is 48, considering everything is correct. The lowest mental model score a participant can achieve is 0. The participants scored a mean total mental model score of 29.54 (61.54%, $\sigma = 6.45$) (see Table 5.17). For the *parts* components the mean score is 17.11 (71.29%, $\sigma = 4.46$) and for the *state changes* components the mean score is 12.43 (51.79%, $\sigma = 3.03$). I performed a paired t-test to compare the differences within the *parts* and *state changes* mental model score. Similar to the *parts* and *state changes* correspondence score, I found a statistically significant difference between these two scores. The participants' mental model score for the *parts* components was higher than the *states changes* components ($t = 10.85$, $p < 0.001$) (shown in Table 5.17).

5.4.5 Mental Model Score and Demographics

I performed additional analysis on participants' mental model score and their demographics (details are in Table 5.19). I performed a one-way ANOVA and found a statistically significant difference ($F(2, 90) = 6.47, p = 0.002$) in participants' mental model scores based on the prior programming language experience. Tukey's HSD Test for multiple comparisons found that participants who had experience with programming language other than Java scored significantly higher than participants who had no programming experience ($p = 0.002$, 95% $C.I.$ = [-8.29, -1.64]).

Table 5.19: Participants' correctness score and mental model score by demographics.

Demographics		Correctness Score			MMS		
		Mean (%)	*SD*	*p*	Mean (%)	*SD*	*p*
Previous Programming Experience	(a) Yes (n = 42)	57.47	6.07	n.s.	65.08	6.43	n.s.
	(b) No (n = 49)	50.69	5.65		58.5	6.16	
Programming Language Learned	(a) None (n = 48)	46.75	5.55	c > a	57.63	6.46	c > a
	(b) Java (n = 12)	47.92	6.31		59.73	7.79	
	(c) Other Language (n = 33)	58.58	4.59		67.98	4.68	
Gender	(a) Female (n = 19)	46.78	4.79	n.s.	56.83	5.56	b > a
	(b) Male (n = 66)	55.56	6.20		62.63	6.79	
CS Major	(a) Yes (n = 49)	53.64	5.99	n.s.	62.94	7.22	a > b
	(b) No (n = 44)	48.31	5.12		60.04	5.50	

Moreover, I found a statistically significant difference in the total mental model score when I conducted an independent t-test based on the participants' major concentration. Participants who are CS majors ($\bar{x} = 30.21$, $\sigma = 7.22$) had significantly higher mental model scores than participants who are not CS majors ($\bar{x} = 28.82$, $\sigma = 5.50$) ($F = 3.96$, $p \leq 0.05$).

Additionally, I performed statistical analysis with gender as an independent variable. My analysis revealed that male participants ($\bar{x} = 12.59$, $\sigma = 3.33$) scored significantly higher than female participants ($\bar{x} = 11.74$, $\sigma = 2.02$) in *states changes* mental model score ($F = 7.95$, $p \leq 0.05$). When I analyzed the participants' correspondence score and mental model score based on previous programming experience (programming learned or not), I found no statistical difference (see Table 5.19).

5.4.6 Mental Model Classification Frequency Distribution

I classified mental models into eight categories, as described in Section 4.7. Similar to the frequency distribution of each assertion (described in Sections 5.4.1.1 and 5.4.1.2), I describe the frequency distribution of each mental model category (see Figure 5.1) for each component below.

5.4.6.1 *Part*: Name

As shown in Table 5.18, none of the participants could be classified in the category of inconsistent and incorrect for the concept *name*. Based on the analysis, I found twelve (12.9%) participants to hold consistent and incorrect mental models. These 12 participants were holding wrong assertions consistently without any internal contradiction. I found 12 participants (12.9%) to be in the IMI category. These participants' mental models were mostly incorrect and had evidence of internal contradiction. For name, I did not find any participants in the CMI, IMC, and CMC categories. However, the majority of our participants' (N = 69, 74.2%) mental models were in the correct (C) category.

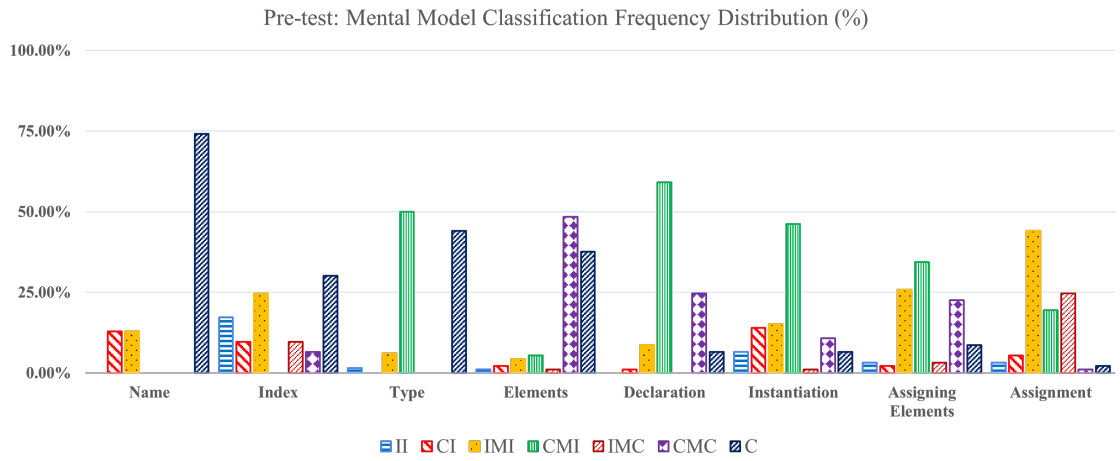


Figure 5.1: Frequency distribution (in percentage) of the categories of the mental models. Here, II: Inconsistent and Incorrect mental model, CI: Consistently Incorrect mental model, IMI: Inconsistent and Mostly Incorrect mental model, CMI: Consistent and Mostly Incorrect mental model, IMC: Inconsistent and Mostly Correct mental model, CMC: Consistent and Mostly Correct Mental Model, C: Correct mental model.

5.4.6.2 *Part*: Index

For the part *index*, I found a spread in the distribution. My findings placed 16 (17.2%) participants in the II category, 9 (9.7%) participants in the CI category, and 23 (24.7%) participants in the IMI category. On the other hand, I found 9 (9.7%) participants in the IMC category, 6 (6.5%) participants in the CMC category, and 28 (30.1%) in the correct category. Interestingly, for *index*, the frequency distribution was higher in the incorrect categories. As the concept of *index* was not taught to the students at the time of this study, this finding is not surprising.

5.4.6.3 *Part*: Type

For the component *type*, I found the majority of participants (N = 42, 45.2%) in the consistent and mostly incorrect (CMI) category. Then, I found 41 (44.1%) participants in the correct category. The rest of the participants were dispersed in IMI (N= 6, 6.5%), CI (N = 1, 1.1%), and II (N = 3, 3.2%) categories (see Table 5.18).

5.4.6.4 *Part*: Elements

Although, similar to *index*, array *elements* is a new concept, I found the majority of participants (N = 45, 48.4%) in the consistent and mostly correct category (CMC). The correct category was filled with 35 (37.6%) participants. The rest of the participants were dispersed in the other categories (see Table 5.18).

Taken together, the participants' distribution was more inclined towards the correct assertion for the *parts* component. Interestingly, for the *state changes* components, that is not the case. Below, I describe the frequency distribution of the participants in each mental model category for the *state changes* components.

5.4.6.5 *State Change*: Declaration

For *declaration*, the majority of the participants (N = 55, 59.1%) were found to be in the consistent and mostly incorrect (CMI) category. Next, I found 23 (24.7%) participants in the consistent and mostly correct (CMC) group. My results showed only six (6.5%) participants could be placed in the correct category. The rest of the sample was distributed into CI (N = 1, 1.1%) and IMI (N = 8, 8.6%) categories.

5.4.6.6 *State Change*: Instantiation

Similar to *declaration*, I found majority of the participants (N = 43, 46.2%) belonged in CMI category for *instantiation*. Consecutively, I found 14 (15.1%) participants in the IMI category, followed by the CI category (N = 13, 14%). Strikingly, for *instantiation*, more frequency distribution was found in the incorrect categories (total N = 76, 81.8%). Ten (10.8%) participants were in the category of consistent and mostly correct, and only 6 participants consistently answered correctly all the questions of *instantiation* (see Table 5.18).

5.4.6.7 *State Change*: Assigning Elements

For *assigning elements*, I found 32 (34.4%) participants in the CMI category. Consecutively, 24 (25.8%) and 21 (22.6%) participants were found in the IMI and CMC

categories. I found 8 (8.6%) participants in the correct category for this concept.

5.4.6.8 *State Change*: Array Assignment

When one array reference variable is assigned to another array reference variable, I labeled the *state change* as *assignment*. I found the majority of participants (N = 41, 44.1%) belonged to the inconsistent and mostly incorrect (IMI) group. Next, I found 23 (24.7%) participants in the IMC category. It is noteworthy to mention that over 50% participants were distributed in the incorrect categories. I noticed only 2 (2.2%) participants in the correct category.

The frequency distribution in mental model categories indicates that the *state change* components were more challenging for our participants than the *parts* components.

5.5 Discussion

In this section, I discuss the key findings regarding incoming novice programmers' mental models based on the research questions stated in Section 5.1. A discussion geared towards RQ1 is discussed in Section 5.5.1, RQ2 in Section 5.5.2, and RQ3 in Section 5.5.3.

5.5.1 Incoming Novice Programmers' Mental Models

To answer my first research question **RQ1. What are the characteristics of incoming novice programmers' mental models?** I draw evidence from the frequency distribution of selection of assertions for each component of array's *parts* and *state changes* as well as the correctness scores and frequency distribution of mental model consistency and classification. Below I discuss the findings reflecting upon the RQ1.

In summary, I can conclude with the following comments on incoming novice programmers' mental models:

Name: Incoming CS1 students' mental model on *name* seemed well developed in

light of correctness and consistency.

Index: Since the concept of indexing is novel to incoming CS1 students, their mental model seemed inconsistent and not well developed.

***Type*:** The majority of the incoming CS1 students' mental models of array's *type* is well developed.

***Elements*:** Most of the students' mental models are well developed for identifying suitable elements an array can store. However, their mental model seemed inconsistent regarding the number of elements an array can store.

***Declaration*:** Incoming CS1 students' mental models of *declaration* seemed incomplete based on their selection of assertions and their consistency.

***Instantiation*:** Similar to *declaration*, students do not have a clear mental model of the state changes that occur after instantiation, such as memory allocation and initialization of default values to the elements.

***Assigning Elements*:** Incoming CS1 students' mental models of *assigning elements* seemed inconsistent on the correct assertions and consistent for one of the incorrect assertions.

***Assignment*:** The most of incoming CS1 students' mental models of array's *assignment* are incorrect and inconsistent.

Below, I elaborate on my discussion of each comment.

For the component *name*, the majority of the students consistently selected the right assertion (MN1). Even though the students have not learned arrays yet in the classroom, they have seen primitive variables and objects, and they know how to identify their *name*. Therefore, it's not surprising that the *name* of the array was consistently correct for most students. The data from Table 5.18 further solidifies this argument. Table 5.18 shows over 80% of the students' mental models were correct and consistent for the component *name*. In addition, almost 75% of the students were categorized as having the correct (abbreviated as C) mental models. Moreover, even

before learning arrays, students could differentiate between an array's name and an element it is storing. Hence, the incorrect assertion 'MN5: First element in the array is the name of the array' was not present in the student's mental models at all.

Unlike *name*, the selection of assertion for *index* was scattered. Only a handful (28%) of the students consistently selected the correct assertion about the array's start index. Additionally, none of the participants consistently (always) selected the right assertion about an array's last index. However, over 70% of the students sometimes selected the correct assertions MI1.1 and MI1.2. I can interpret this occurrence as since the students are not taught the array's structure yet, their mental model of array's *index* is inconsistent. The evidence that approximately 55% of the students were deemed inconsistent (Table 5.18, column 8) strengthens the claim. Students' mental model classification was scattered between correct (C), inconsistent and mostly incorrect (IMI), and inconsistent and incorrect (II) (shown in Table 5.18). I observed the commonly known misconception regarding arrays that their indices start with 1 (62.4%) and end with n (66.7%), gaining similar selection (sometimes) as the correct counterparts MI1.1 and MI1.2. Since students are not familiar with the concept of indexing, among the *parts* components, the index has the least correct mental models.

Similar to the component *type*, I found most students (89.2%) consistently (always) selecting the correct assertion. I apply the similar observation that since data type has already been taught; the students applied their mental models of data type to identify the type of the array. Table 5.18 provides further evidence for this claim. Over 90% of students were correct in answering the questions regarding array's *type*. The consistent support (75.3%) for the correct assertion 'ME1.2: Elements stored in the array can be only of the declared type' also supports students' mental model development of the array's data type.

The component array's *elements* was composed of two knowledge units: the number and the type of element it can store. As mentioned earlier, most of the students

always selected the correct assertion about what type of element a given array can store. However, only three students always selected the correct assertion about the number of elements a given array can store. Most of the student's responses on this were inconsistent (96.8% sometimes selected).

Although most of the students (47.3%) did not think that after the declaration, an array reference variable is set to `null` (MD1.1), many of them (41.9%) consistently believed that after the declaration, an array reference variable is created (MD1.2). However, a fair portion of them selected this assertion sometimes (33.3%). Therefore, I can conclude that this assertion has not been solidified in their mental models. Although being correct, over half of the students never selected that there no memory allocation after declaration (52.7%). Data from Table 5.18 further shows the incompleteness of students' mental models on *declaration*. Most of the students answered incorrectly to the declaration questions. The distribution of mental model classification further showed that most students were holding consistent and mostly incorrect mental models.

Though over half of the students supported the fact that after array instantiation, memory is allocated for the array (MIn1.1), they are not sure about what default value is set for each of the elements. In Java, once instantiated, the default value of each element is set to 0 for integer arrays, 0.0 for deciMAE arrays, and '\u0000' for character arrays. Some of them thought there was no default value or blank in the elements after an array instantiation. This evidence suggests that students do not have a clear understanding of memory allocation and the default values of the array once declared and instantiated. Among all the components, students demonstrated the least correctness (35.35%) on questions of array *instantiation*. Analysis of their responses' consistency revealed that they were mostly consistent in selecting the incorrect assertion. This may indicate that there lies evidence of misconceptions in their mental models of array *instantiation*. Since they have not been instructed in

the classroom, these findings are not surprising.

Assigning a value to an element of an array is the same as assigning a value to a primitive variable. At the point of data collection, students are expected to gain proficiency with primitive variable assignments. However, though almost 90% of students selected sometimes, surprisingly, none of the students consistently chose the correct assertion that the assignment copies the values from right to left (MAE1.1). In previous research, Ma et al. [158] found that even after classroom instruction, approximately only one-third of students held consistent and correct mental models of value assignment. Strikingly, almost 60% students selected sometimes that the value of a variable never changes (MAE1.2). On the other hand, nine participants believed consistently that primitive variable assignments work the same as the reference assignments. From Table 5.18, we can see almost 50% students answered incorrectly the questions regarding array *assigning elements*. Even though almost 70% of them were found to be consistent, the mental model classification further revealed that most of the students' mental models were consistent and mostly incorrect (CMI).

In the CS education domain, students are commonly known to have inconsistent mental models for reference variable assignments [17, 28]. Our findings provide evidence to support the claim. Only 41.4% of students answered all the questions correctly regarding array *assignment*. Most importantly, the majority of the students' (72%) mental models were inconsistent. Further investigation revealed most (44.1%) of their mental models were inconsistent and mostly incorrect (IMI) (see Table 5.18). The assertion level analysis revealed that many of the students consistently believed that array assignment copied the values (MA3). This is a predominant misconception regarding a reference variable assignment established in the literature [17, 28, 152].

5.5.2 Parts vs. State Changes

To answer the research question **How are the incoming novice programmers' initial mental models of array's *parts* components in comparison with the**

***state changes* components?** I draw evidence from the correctness score and the mental model score. I found data supporting the evidence that the mental model components of *state changes* were more challenging to novice programmers than the *parts*. I found that our participants' correctness score was significantly higher for *parts* components than *states*. Similarly, I found *parts* mental model score significantly higher than *state changes* mental model score. Moreover, for *parts* components, I saw the frequency distribution of selecting the assertions 100% of the time was skewed to the correct assertion. However, for *state changes* components, the percentage of selection was scattered. I observed a similar pattern for the frequency distribution of mental model categories (see Figure 5.1). From these findings, I can answer that the novice programmers' mental model correctness and consistency for array's *parts* components are better than the *state changes* components.

Prior studies have also found both the novices and upper-level students struggle to understand interactions between states [26, 80, 152]. Krishnamurthi et al. [30] describe that comparative studies between stateful and non-stateful programming concepts are understudied in computing education. He remarked, "state is a powerful tool that must be introduced with responsibilities" [30, p.385]. State changes in introductory programming concepts such as variables, arrays, and objects were also seen to be poorly portrayed by most of the common CS1 textbooks [159]. Psychologist Mayer [5,160] showed through several studies that illustrations of a scientific concept's parts and state changes improved the students' performance of recall and creative problem-solving.

5.5.3 Students' Demographics and Mental Models

Below, I discuss the following research question **RQ3. What impact do prior programming experience and demographics have on the mental model of the participants in our study before classroom instruction?** The results revealed several correlations between participants' demographics and their mental

models. First, I observed a significant difference in correctness score and mental model score based on participants' programming experience. Those who learned a programming language other than Java performed better than those who have no prior programming experience (see Table 5.19). Clearly, prior programming knowledge seemed to affect mental model consistency and correctness even before classroom instruction. Researchers have found that prior programming experience can also impact the classroom climate for a CS1 course [161, 162]. Students having prior programming knowledge can create a defensive climate, which is detrimental to those without prior experience [162]. Prior programming knowledge is also considered a predictor of success in a CS1 course [162, 163]. In a study, most students who had prior programming knowledge felt that their prior knowledge was a factor of success, and they felt more confident [162]. Moreover, prior programming knowledge also impacts CS1 students' perception of pair programming. Research suggests CS1 students should form partnerships based on comparable levels of prior programming knowledge [162]. The MMT-A can be an instrument to learn the initial existing knowledge about arrays of CS1 students. By knowing that student partnerships can be formed to make a more effective pair programming experience.

Second, I found the participants who are CS majors scored significantly higher in the mental model score than non-majors. Worthy of note that 51% (n = 25) of the CS major participants had previous programming experience, whereas only 38.6% (n = 17) of non-majors had prior programming experience.

Lastly, the female participant's mental model score was significantly lower than the male participants in our study. My results mirror research reported in the literature documenting the inequities faced by female students in introductory programming courses. My work expands on work by Ramalingam et al. [76], which showed that student's mental models influence their self-efficacy. Self-efficacy has been found to impact course performance and persistence in college courses. This also aligns with

prior work focused on introductory programming that repeatedly has shown female students with lower self-efficacy [164–167].

5.6 Summary

In this chapter, I presented the results gathered from investigating novice programmers' mental models of arrays before classroom instruction. The use of MMT-A elicited mental model assertions for each of the array components. From the mental model assertion, I analyzed the consistency and correctness of incoming novice programmers' mental models. The results revealed that components I classified as *state changes* were more challenging for novice programmers than *parts*. My findings also revealed that prior programming knowledge has an impact on novices' initial mental models' consistency and correctness. This finding supports the belief of constructivism that learners are not clean slates. Moreover, mental model consistency and correctness analysis revealed gender inequity. I found female participants held significantly lower mental model consistency and correctness scores than male participants. Moreover, their mental model score was lower than the average for the whole group.

The study contributes to our understanding of novices' initial mental models of arrays based on their mental model assertions, mental model consistency, and correctness. The next chapter presents the results of investigating novice programmers' mental models after they have received classroom instruction on arrays.

CHAPTER 6: NOVICE PROGRAMMERS' MENTAL MODELS AFTER INSTRUCTION

6.1 Introduction

In the previous chapter, I summarized my findings from investigating novice programmers' mental models before they were given any formal instruction on arrays. This chapter presents the results of my investigation of the novice programmers' mental models after they were given formal instructions along with formative and summative assessments on arrays. The research questions around this investigation are similar to Section 5.1. However, the research questions are answered by centering on novice programmers who already have learned arrays.

Therefore, I state the research questions below:

RQ4. What are the characteristics of novice programmers' mental models after they have learned arrays?

RQ5. How are the novice programmers' mental models of the array's *parts* components in comparison with the *state changes* components after they have learned arrays?

RQ6. What impact do prior programming experience and demographics have on the mental model of the participants after they have learned arrays?

I found that the novice programmers' mental models of the array's *parts* components seemed well-developed. However, even after classroom instruction, the mental models of array's *state changes* components of students suffer inconsistency and inaccuracy. The mental model gap between array's *parts* and *state changes* components

that were found in incoming novice programmers' remained intact after the formal classroom instruction. Further analysis showed the mental model difference based on prior programming experience and gender, which existed prior to instruction diminished after the classroom instruction. Interestingly, the mental model difference based on correctness and consistency between CS majors and non-majors persists. In this chapter, I describe the data I collected and present the results in detail.

6.2 Data Collection

As this study aimed to uncover novice programmers' mental models after they have learned arrays, I used the data set collected after formal classroom instruction on arrays. Therefore, I used the **Spring 2021-Post-test**, **Spring 2023-Post-Instruction**, and **Summer 2023-Post-Instruction** (described in Section 4.12) to answer my research questions.

6.3 Participants

As mentioned earlier, I administered MMT-A post-instruction in three semesters: Spring 2021, Spring 2023, and Summer 2023. In total, 262 participants gave their consent to analyze their data. After removing incomplete and duplicate cases, I could include 144 participants' responses in my analysis. Most of our participants identified themselves as male (70.1%), 20.8% identified as female, 4.86% preferred not to answer, and the rest (4.2%) reported being gender non-binary. Of the 144 participants, 67.4% reported having CS as their major of study. Over half of the participants (54.2%) had no prior programming experience before enrolling in the CS1 course. Among the students who had prior programming experience, 12 participants had experience with Java, and 54 participants had experience with other programming or markup languages (e.g., Python, JavaScript, C#, C++, SQL, HTML, Snap).

6.4 Results

Sections 6.4.1 enumerate the assertions residing in novice programmers' mental models of an array after they have learned arrays. I present the correctness and consistency of novice programmers' mental models in Sections 6.4.2, 6.4.3, and 6.4.4. Lastly, the effect of prior programming experience and demographics on novice programmers' mental models is discussed in Section 6.4.5.

6.4.1 Mental Model Assertions

As previously described, concepts in the mental model of arrays are broken down into *parts* and *state changes*. Below, I present the selection of mental model assertion for each component of arrays *parts* and *state changes*.

6.4.1.1 Assertions for *Parts*

Part: Name

The assertions MN1, MN2, and MN3 appeared in two questions each, and MN4 and MN5 appeared in only one question each (see Table 6.1). The results show that 132 out of 144 students selected MN1 when it appeared as the correct option, and an additional seven students selected sometimes. For the incorrect choices, a total of 135 students avoided (i.e., never selected it) MN2, 144 (all) avoided MN3, 141 avoided MN4, and 142 (all 100%) avoided MN5. Table 6.2 summarizes the frequency distribution for each kind of selection.

Table 6.1: List of assertions for the *part* **name**.

Assertions	Correctness	Num. of Questions
MN1: Array reference variable is the name of the array	Correct	2
MN2: Array type is the name of the array	Incorrect	2
MN3: Keyword new is the name of the array	Incorrect	2
MN4: Whatever comes after equal sign in an initialization is the name of the array	Incorrect	1
MN5: First element in the array is the name of the array	Incorrect	1

Table 6.2: Frequency distribution of the selection of assertions for the *part* **name**.

	MN1 correct	MN2 incorrect	MN3 incorrect	MN4 incorrect	MN5 incorrect
Always selected	91.7% (132)	2.1% (3)	0% (0)	2.1% (3)	1.4% (2)
Sometimes selected	4.9% (7)	4.2% (6)	0% (0)	-	-
Never selected	3.5% (5)	93.8% (135)	100% (144)	97.9% (141)	98.6% (142)
Total	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)

Part: Index

Table 6.3 lists the assertions used in the MMT-A. The frequency distribution of each assertion is presented in Table 6.4. I have broken down the correct assertion into two separate assertions, MI1.1 and MI1.2. The assertions MI1.1 appeared nine times and MI1.2 six times. Together, those two are the correct assertions regarding the use of indices in arrays. The other incorrect assertions are listed in Table 6.4. It is worth noting that MI4 and MI5 represent common misconceptions on the use of array *indices*, namely that the index of the first element in the array is [1] and that the last element is stored at [n] instead of the correct answer of [n-1].

The correct assertion of MI1.1 was selected consistently by 56.9% (82) of the participants. In addition, 43.1% (62) of the participants sometimes selected this option. Surprisingly, the complimentary assertion of MI1.1 and MI1.2 had a different distribution than MI1.1. None of the participants selected it consistently, and an additional 96.5% (139) selected it some of the time (at least once).

The incorrect choices show a high percentage of participants who avoided those options. MI2 was never selected consistently; 1.4% of the participants selected it at least once, and 98.6% of the participants avoided it altogether. Table 6.4 shows similar numbers for MI3, MI6, and MI7.

MI4 and MI5, however, have different numbers. These assertions reflect a misunderstanding of how indices work in Java. They reflect a misunderstanding that the index starts at 1 (MI4) and ends at n instead of n-1 (MI5). The start/end of an array is a documented misconception [7, 111], so it is no surprise that these incorrect assertions drew higher participant responses. Although none of the participants always selected, 37.5% of participants selected MI5 at least once. Similarly, none of the participants always selected MI5, but 32.6% (47) of them selected it at least once.

Table 6.3: List of assertions for the *part* **index**.

Assertions	Correctness	Num. of Questions
MI1.1: Array index starts with 0	Correct	9
MI1.2: Array index ends with n-1 (index of last element)	Correct	6
MI2: Index of an array can be of any type, not just integers	Incorrect	3
MI3: There is no indexing into the array	Incorrect	2
MI4: Array index starts with 1	Incorrect	8
MI5: Array index ends with n (index of last element)	Incorrect	7
MI6: Array index does not map to its corresponding location in the array	Incorrect	3
MI7: Students think the index is the element.	Incorrect	3

Table 6.4: Frequency distribution of the selection of assertions for the part **index**.

	MI1.1 correct	MI1.2 correct	MI2 incorrect	MI3 incorrect	MI4 incorrect	MI5 incorrect	MI6 incorrect	MI7 incorrect
Always selected	56.9% (82)	0% (0)	0% (0)	1.4% (2)	0% (0)	0% (0)	0% (0)	0% (0)
Sometimes selected	43.1% (62)	96.5% (139)	1.4% (2)	6.3% (9)	37.5% (54)	32.6% (47)	9% (13)	6.3% (9)
Never selected	0% (0)	3.5% (5)	98.6% (142)	92.4% (133)	62.5% (90)	67.4% (97)	91% (131)	93.8% (135)
Total	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)

Part: Type

The assertions MT1, MT2, and MT3 each appear in two questions in the instrument (see Table 6.5). The correct assertion, MT1, represents a clear understanding of the difference between array *name* and array *type* in an array declaration statement.

Table 6.6 shows the frequency distribution of each assertion. The results show that 97.2% (140) of the participants selected MT1 consistently. Only one participant avoided MT1 both times it appeared in the instrument. Only one participant selected MT2 both times, and two participants selected MT2 sometimes. These students seemed to confuse the array *name* with the array *type*.

Table 6.5: List of assertions for the *part* **type**.

Assertions	Correctness	Num. of Questions
MT1: Name appearing before the array name is the type of the array	Correct	2
MT2: Name of the array is the type of the array	Incorrect	2
MT3: Keyword new is the type of the array	Incorrect	2

Table 6.6: Frequency distribution of the selection of assertions for the *part* **type**.

	MT1 correct	MT2 incorrect	MT3 incorrect
Always selected	97.2% (140)	0.01 (1)	0% (0)
Sometimes selected	2.1% (3)	1.4% (2)	0.01 (1)
Never selected	0.01 (1)	97.9% (141)	99.3% (143)
Total	100% (144)	100% (144)	100% (144)

Part: Element

The assertions ME1.1 and ME1.2 are the correct assertions regarding array *elements*. Each of these appeared two times in the instrument. Assertions ME2 (5), ME3 (4), ME4 (4), ME5(2), ME6 (2), and ME7 (2) were invalid assertions about array elements (see Table 6.7). As shown in Table 6.8, ME1.1 (size of the array) was selected consistently by only 4.2% (6) of the participants. In addition, an additional 95.8% (138) of the participants selected this option at least once. ME1.2 was, on the other hand, a very common choice, with 82.6% (119) participants selecting it

Table 6.7: List of assertions of the part **element**.

Assertions	Correctness	Num. of Questions
ME1.1: The array contains n number of elements	Correct	7
ME1.2: Elements stored in the array can be only of the declared type	Correct	2
ME2: The array contains size+1 elements	Incorrect	5
ME3: The array contains size-1 elements	Incorrect	4
ME4: After instantiation, the array doesn't have space to store any elements (size 0)	Incorrect	4
ME5: Element values and array names are related (e.g., books and "Harry Potter")	Incorrect	2
ME6: Keyword 'new' is an element in the array	Incorrect	2
ME7: Type of values stored do not match type of array	Incorrect	2

consistently. Interestingly, 13.2% (19) of the participants never selected this option. This assertion (ME1.2, Elements stored in the array can be only of the declared type) seems to divide the participants into two extremes: 82.6% consistently selected, and 13.2% never selected it.

Worthy of note is option ME5. This (erroneous) assertion appeared twice in the instrument and implied that the name of an array was semantically related to the value stored in the array. For example, an array of type String and named "books" can only store strings containing names of books, such as "Harry Potter". The results show that 7.6% (11) of the participants selected this option consistently, and an additional 2.1% (3) selected this assertion sometimes.

Table 6.8: Frequency distribution of the selection of assertions for the part **element**.

	ME1.1 correct	ME1.2 correct	ME2 incorrect	ME3 incorrect	ME4 incorrect	ME5 incorrect	ME6 incorrect	ME7 incorrect
Always selected	4.2% (6)	82.6% (119)	0% (0)	0% (0)	0% (0)	7.6% (11)	0.01 (1)	1.4% (2)
Sometimes selected	95.8% (138)	4.2% (6)	25.7% (37)	26.4% (38)	5.6% (8)	2.1% (3)	3.5% (5)	5.6% (8)
Never selected	0% (0)	13.2% (19)	74.3% (107)	73.6% (106)	94.4% (136)	90.3% (130)	95.8% (138)	93.1% (134)
Total	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)

6.4.1.2 Assertions for *State Changes*

This section presents the results for all the *state changes* of the array mental model included in the MMT-A. The sections that follow cover the *state changes* in our instrument: *declaration*, *instantiation*, *assigning elements*, and *assignment*.

State change: Declaration

The assertions MD1.1, MD1.2, MD1.3, and MD1.4 are the correct assertions for array *declaration* (see Table 6.9). The assertion MD1.1 was selected consistently by 32.6% (47) of the participants, with an additional 16% (23) participants selecting MD1.1 sometimes (see Table 6.10). The remaining 51.4% (74) of the participants never selected MD1.1. These results indicate a high level of misunderstanding among the participants in our study.

The assertion MD1.2 was selected consistently by 45.8% (66) of the participants, with an additional 29.2% (42) participants selecting MD1.1 sometimes (see Table 6.10). The remaining 25% (36) of the participants never selected MD1.2. MD1.3 and MD1.4 each appeared only once in our instrument, and as a result, participants either selected that assertion (MD1.3 67.4% and MD1.4 45.8%) or not (MD1.3 32.6% and MD1.4 54.2%).

For the incorrect assertions, MD2 stands out as 27.8% (40) of the participants consistently selected it. An additional 13.9% (20) of the participants selected this option at least once. At the other extreme, 58.3% (84) of the participants never selected this option.

The remaining incorrect assertions (MD3, MD4, MD5, MD6, and MD7) were mostly selected sometimes or not selected at all. Still, these assertions should not have been selected (i.e., incorrect assertion). Particularly, MD5 (42.4%) and MD6 (50.7%) have a higher selection rate at least once. The distribution of responses shows that these assertions and, thus, this concept is problematic for lots of students.

Table 6.9: List of assertions of the *state change* **declaration**.

Assertions	Correctness	Num. of Questions
MD1.1: After declaration, default value of array reference variable is set to null	Correct	2
MD1.2: After declaration, an array reference variable is created	Correct	2
MD1.3: After declaration, no memory is allocated for the array	Correct	1
MD1.4: After declaration, no elements can be stored.	Correct	1
MD2: There is no default value for the elements of the array (blank/no value)	Incorrect	2
MD3: The default value for the array reference is the default value for type (e.g., int is 0, boolean is false)	Incorrect	4
MD4: The default value for the array reference is stored as '?'	Incorrect	3
MD5: After declaration, memory is allocated for the elements	Incorrect	3
MD6: After declaration, the number of elements that can be stored is unlimited	Incorrect	3
MD7: After declaration, there is a default size for an array	Incorrect	1

Table 6.10: Frequency distribution of the selection of assertions for the *state change* **declaration**.

	MD1.1 correct	MD1.2 correct	MD1.3 correct	MD1.4 correct	MD2 incorrect	MD3 incorrect	MD4 incorrect	MD5 incorrect	MD6 incorrect	MD7 incorrect
Always selected	32.6% (47)	45.8% (66)	67.4% (97)	45.8% (66)	27.8% (40)	1.4% (2)	0% (0)	0% (0)	5.6% (8)	4.9% (7)
Sometimes selected	16% (23)	29.2% (42)	-	-	13.9% (20)	36.1% (52)	0.01 (1)	42.4% (61)	50.7% (73)	-
Never selected	51.4% (74)	25% (36)	32.6% (47)	54.2% (78)	58.3% (84)	62.5% (90)	99.3% (143)	57.6% (83)	43.8% (63)	95.1% (137)
Total	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)

State change: Instantiation

As shown in Table 6.12, MIn1.1 was selected consistently by 76.4% (110) of the participants. The rest of the participants, 23.6% (34), never selected this option. A second valid assertion in this concept, MIn1.2 had no participants selected consistently. Just over half of the participants, 62.5% (90), selected this option some of the time, with 37.5% (54) of the participants never selecting this option.

MIn2, most participants (88.9%) never selected this option, and 8.3% of the participants selected it sometimes. Only a handful (2.8%) selected this option consistently. Similarly, MIn3 was selected by 21.5% (31) of the participants sometimes, and 78.5%

(113) never selected it. None selected MIn3 consistently. For MIn4, which is about no memory allocation after instantiation, two participants selected it consistently. 46.5% (67) selected it sometimes, and the rest, 52.1% (75), avoided this option.

Table 6.11: List of assertions for the state change **instantiation**.

Assertions	Correctness	Num. of Questions
MIn1.1: After instantiation, memory is allocated for the array	Correct	1
MIn1.2: After instantiation, the appropriate default value is assigned to the elements	Correct	4
MIn2: After instantiation, '?' is stored as a default value	Incorrect	3
MIn3: After instantiation, there is no default value (blank) stored for the elements	Incorrect	4
MIn4: After instantiation, no memory is allocated	Incorrect	4

Table 6.12: Frequency distribution of the selection of assertions for the state change **instantiation**

	MIn1.1 correct	MIn1.2 correct	MIn2 incorrect	MIn3 incorrect	MIn4 incorrect
Always selected	76.4% (110)	0% (0)	2.8% (4)	0% (0)	1.4% (2)
Sometimes selected	-	62.5% (90)	8.3% (12)	21.5% (31)	46.5% (67)
Never selected	23.6% (34)	37.5% (54)	88.9% (128)	78.5% (113)	52.1% (75)
Total	79.2% (114)	79.2% (114)	79.2% (114)	79.2% (114)	79.2% (114)

Table 6.13: List of assertions for the *state change* **assigning elements**.

Assertions	Correctness	Num. of Questions
MAE1.1: Assignment copies the values from right to left.	Correct	6
MAE1.2: The variable on the right-hand side remains the same after assigning.	Correct	2
MAE2: The value of a variable never changes.	Incorrect	8
MAE3: A variable can hold multiple values at a time / 'remembers' old values.	Incorrect	10
MAE4: Assignment swaps values of the left and right hand side.	Incorrect	2
MAE5: Primitive assignment is the same as reference assignment.	Incorrect	2

State change: Assigning Elements

Table 6.13 shows all the assertions for the concept, *Assigning Elements* with two correct and four incorrect assertions (see Table 6.14). For the correct assertions,

MAE1.1 was never selected consistently. However, almost all participants, 97.9% (141), selected this assertion some of the time, and only 2.1% (3) of the participants avoided it completely. For the other correct assertion, MAE1.2, 66% (95) selected it consistently, and an additional 19.4% (28) participants selected it sometimes. 14.6% (21) participants avoided this option.

For the incorrect assertions, MAE2 and MAE3 zero (0) participants selected these options consistently. Most participants avoided MAE2 (66.7%) and MAE3 (70.1%).

MAE4 and MAE5 appeared only twice in the instrument. For MAE4, only three participants selected it consistently, 11.1% selected it sometime, and 86.8% avoided the option altogether. For MAE5, 4.2% selected it consistently, an additional 9.7% selected it sometimes, and the majority of the participants, 86.1%, never selected this option.

Table 6.14: Frequency distribution of the selection of assertions for the *state change* **assigning elements**.

	MAE1.1 correct	MAE1.2 correct	MAE2 incorrect	MAE3 incorrect	MAE4 incorrect	MAE5 incorrect
Always selected	0% (0)	66% (95)	0% (0)	0% (0)	2.1% (3)	4.2% (6)
Sometimes selected	97.9% (141)	19.4% (28)	33.3% (48)	29.9% (43)	11.1% (16)	9.7% (14)
Never selected	2.1% (3)	14.6% (21)	66.7% (96)	70.1% (101)	86.8% (125)	86.1% (124)
Total	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)

Table 6.15: List of assertions for the *state change* array **assignment**.

Assertions	Correctness	Num. of Questions
MA1: Array assignment copies reference from right to left, sharing the memory.	Correct	3
MA2: Array assignment appends value at the end of the array.	Incorrect	1
MA3: Array assignment copies the values.	Incorrect	3
MA4: Array assignment transfers (cuts) values.	Incorrect	2
MA5: Array assignment copies the reference but does not share memory.	Incorrect	2

State change: Array Assignment

Table 6.15 lists all assertions related to the *state change array assignment*, also labeled as *assignment* in short. MA1 is the correct assertion; nine of the participants selected it consistently. However, over half of the total participants (63.9%) sometimes selected this option, and 29.9% of participants avoided it.

The first of the incorrect assertions, MA2, had 93.1% (134) of the participants avoiding this option, and the remainder of the participants, 6.9% (10), selected it consistently. The next incorrect assertion, MA3, had a distribution of participants, with 10.4% (15) selecting it consistently, 70.8% (102) selecting it some time, and 18.8% (27) avoiding it.

The last two incorrect assertions, MA4 and MA5, did not have any participant select it consistently. The participants were divided between selecting it sometimes (MA4 16.7% and MA5 22.9%), with the majority in both cases avoiding it altogether (MA4 83.3% and MA5 77.1%).

Table 6.16: Frequency distribution of the selection of assertions for the *state change* **array assignment**.

	MA1 correct	MA2 incorrect	MA3 incorrect	MA4 incorrect	MA5 incorrect
Always selected	6.3% (9)	6.9% (10)	10.4% (15)	0% (0)	0% (0)
Sometimes selected	63.9% (92)	-	70.8% (102)	16.7% (24)	22.9% (33)
Never selected	29.9% (43)	93.1% (134)	18.8% (27)	83.3% (120)	77.1% (111)
Total	100% (144)	100% (144)	100% (144)	100% (144)	100% (144)

6.4.2 Mental Model Correctness

I scored each participant based on the correctness of their answers in MMT-A, as discussed in Section 4.6. I referred to this score as the *correctness score*. As there were 36 questions, the possible highest score could be 36, and the lowest score could be 0. I report on the total correctness score and the correctness score for each *part* and *state change* component.

Table 6.17: Participant's overall correctness and mental model score with individual scores for *parts* and *state changes*. Participants' correctness score and mental model score for the *state changes* is statistically significantly lower than the *parts* (shown in the last row).

	Correctness Score		Mental Model Score	
	Mean (%)	SD	Mean (%)	SD
Overall	70.79	6.49	77.62	6.82
Parts	88.14	2.63	90.97	3.56
State changes	59.75	4.45	64.26	4.12
Parts vs. State changes	*p <0.001*		*p <0.001*	

For the total correctness score, the participants answered an average of 25.49 out of 36 (70.79%, $\sigma = 6.49$, N = 144) questions correctly (see Table 6.17). A participant's minimum correctness score is seven, and the maximum score is 36.

Moreover, I computed the correctness score for the *parts* and *state changes*. I had 14 questions for *parts* and 22 questions covering *state changes*. The mean correctness score for *parts* was 12.34 out of 14 (88.14%, $\sigma = 2.63$, N = 144) and for *state changes* was 13.14 out of 22 (59.75%, $\sigma = 4.45$, N = 144).

Table 6.18 (column 3) shows the detailed breakdown of the correctness score for the concepts of parts and state changes. I present the mean correctness score in percentage as the number of questions varies for each concept. Note that participants' mean correctness scores were higher for *parts* than for *state changes*. A paired t-test shows that participants scored significantly higher ($t = 21.07, p < 0.001$) in *parts* than in *state changes* (shown in Table 6.17).

6.4.3 Mental Model Consistency

Based on the contradictions in a participant's mental model assertion, I categorized mental models into two categories: consistent and inconsistent (see Section 4.5 for the

definition). I measured mental model consistency for each *part* and *state change*. Table 6.18 (columns 4 and 5) includes the absolute frequency (N) and relative frequency (%) obtained from the analysis of consistency.

Almost all participants (over 90%) had a consistent mental model for *name* (93.75%, 135), *type* (97.92%, 141), *elements* (93.75%, 135), and *declaration* (95.83%, 138). For *instantiation*, over 80% of participants were consistent in their mental model assertions.

Among the *parts* components, participants demonstrated the least mental model consistency for array's *index* (68.06%, 98). Worthy to note, for the *state change assignment*, most of the participants (77.78%, 112) demonstrated inconsistency in their mental models.

6.4.4 Mental Model Score: Combining Correctness & Consistency

As described in Section 4.7, I categorized and ranked a mental model based on its consistency and correctness score. The correct mental model (see Table 4.2), which is always consistent, is ranked the highest (6). As a mental model rank is given for each *part* and *state changes*, a total mental model score is calculated by adding the mental model ranks for the components of each *part* and *state changes*. There are, in total, eight components. Therefore, a participant's highest mental model score is 48, considering everything is correct. The lowest mental model score a participant can achieve is 0. The participants scored a mean total mental model score of 37.26 (77.62%, $\sigma = 6.82$) (see Table 6.17). For the *parts* components, the mean score is 21.83 (90.97%, $\sigma = 3.56$) and for the *state changes* components the mean score is 15.42 (64.26%, $\sigma = 4.12$). I performed a paired t-test to compare the differences within the *parts* and *state changes* mental model score. Similar to the *parts* and *state changes* correspondence score, I found a statistically significant difference between these two scores. The participants' mental model score for the *parts* components was higher than the *states changes* components ($t = 21.42$, $p < 0.001$) (shown in Table

Table 6.18: Participants' correctness, consistency, and mental model classification for each *part* and *state changes* (N = 144). Here, II: Inconsistent and Incorrect mental model, CI: Consistently Incorrect mental model, IMI: Inconsistent and Mostly Incorrect mental model, CMI: Consistent and Mostly Incorrect mental model, IMC: Inconsistent and Mostly Correct mental model, CMC: Consistent and Mostly Correct Mental Model, C: Correct mental model.

Components	# Qs.	Correctness		Consistency				Mental Model Classification													
				Consistent		Inconsistent		II		CI		IMI		CMI		IMC		CMC		C	
		Mean % (N=144)	SD	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%
P:Name	2	94.10	20.90	135	93.75	9	6.25	2	1.39	3	2.08	7	4.86	0	-	0	-	0	-	132	91.67
P:Index	5	85.69	26.09	98	68.06	46	31.94	5	3.47	0	0	11	7.64	0	-	25	17.36	8	5.56	90	62.50
P:Type	2	98.26	10.92	141	97.92	3	2.08	0	-	1	0.69	3	2.08	0	-	0	-	0	-	140	97.22
P:Elements	5	84.17	28.17	135	93.75	9	6.25	2	1.39	3	2.08	5	3.47	8	5.56	2	1.39	23	15.97	101	70.14
S:Declaration	6	52.55	27.26	138	95.83	6	4.17	1	0.69	2	1.39	5	3.47	72	50.00	0	-	50	34.72	14	9.72
S:Instantiation	4	56.08	36.84	121	84.03	23	15.97	0	-	14	9.72	21	14.58	48	33.33	2	1.39	8	5.56	51	35.42
S:Assigning Elements	6	74.54	28.49	115	79.86	29	20.14	3	2.08	0	-	17	11.81	19	13.19	8	5.56	43	29.86	53	36.81
S:Assignment	6	54.63	23.02	32	22.22	112	77.78	3	2.08	4	2.78	52	36.11	19	13.19	57	39.58	2	1.39	7	4.86

6.17).

6.4.5 Mental Model Score and Demographics

I analyzed participants' correctness scores and mental model scores based on their demographics (details are in Table 6.19). When I analyzed the participants' correspondence score and mental model score based on previous programming experience (programming learned or not), I found no statistical difference (see Table 6.19). Moreover, I performed a one-way ANOVA and found no statistically significant difference in participants' mental model scores based on the prior programming language experience.

However, I found a statistically significant difference in the total mental model score when I conducted an independent t-test based on the participants' major concentration. Participants who are CS majors ($\bar{x} = 26.23$, $\sigma = 6.10$) had significantly higher correctness scores than participants who are not CS majors ($\bar{x} = 23.96$, $\sigma = 7.07$) ($F = 0.97, p \leq 0.05$). Similarly, participants who are CS majors ($\bar{x} = 38.21$, $\sigma = 6.21$) had significantly higher mental model scores than participants who are not CS majors ($\bar{x} = 35.30$, $\sigma = 7.63$) ($F = 1.30$, $p \leq 0.05$).

Additionally, I performed statistical analysis with gender as an independent variable. I found no significant difference in correctness and mental model scores based

Table 6.19: Participants' correctness score and mental model score by demographics. Participants' correctness and mental model scores were statistically significantly higher for those who are CS majors (last two rows).

Demographics		Correctness Score			MMS Score		
		Mean (%)	*SD*	*p*	Mean (%)	*SD*	*p*
Previous Programming Experience	(a) Yes (n = 66)	72.14	6.53	n.s.	79.04	6.48	n.s.
	(b) No (n = 78)	69.66	6.48		76.42	7.08	
Programming Language Learned	(a) None (n = 78)	68.98	6.58	n.s.	75.88	7.14	n.s.
	(b) Java (n = 12)	65.51	7.58		74.48	8.04	
	(c) Other Language (n = 54)	74.59	5.98		80.83	5.83	
Gender	(a) Female (n = 30)	72.42	4.93	n.s.	78.75	5.02	n.s.
	(b) Male (n = 101)	69.91	7.00		76.94	7.43	
CS Major	(a) Yes (n = 97)	72.85	6.10	$p \leq 0.05$ (a > b)	79.60	6.21	$p \leq 0.05$ (a > b)
	(b) No (n = 47)	66.55	7.07		73.41	7.63	

on gender.

6.4.6 Mental Model Classification Frequency Distribution

I classified mental models into eight categories, as described in Section 4.7. Similar to the frequency distribution of each assertion (described in Sections 6.4.1.1 and 6.4.1.2), I describe the frequency distribution of each mental model category (see Figure 6.1) for each component below.

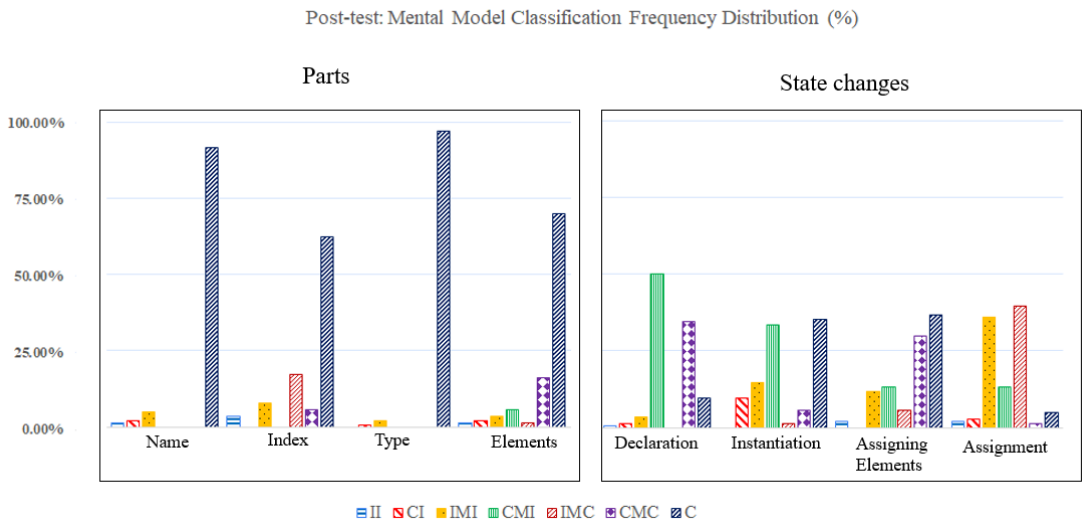


Figure 6.1: Frequency distribution (in percentage) of the categories of the mental models. Here, II: Inconsistent and Incorrect mental model, CI: Consistently Incorrect mental model, IMI: Inconsistent and Mostly Incorrect mental model, CMI: Consistent and Mostly Incorrect mental model, IMC: Inconsistent and Mostly Correct mental model, CMC: Consistent and Mostly Correct Mental Model, C: Correct mental model.

6.4.6.1 *Part*: Name

As shown in Table 6.18, only two out of 144 of the participants could be classified in the category of inconsistent and incorrect (II) for the concept *name*. Based on the analysis, I found three participants to hold consistent and incorrect (CI) mental models. These three participants were holding wrong assertions consistently without any internal contradiction. I found seven participants (4.86%) to be in the IMI category. These participants' mental models were mostly incorrect and had evidence

of internal contradiction. For name, I did not find any participants in the CMI, IMC, and CMC categories. However, most of our participants' (N = 132, 91.67%) mental models were in the correct (C) category.

6.4.6.2 *Part*: Index

For the part *index*, I found a spread in the distribution. My findings placed 5 (3.47%) participants in the II category, zero participants in the CI category, and 11 (7.64%) participants in the IMI category. On the other hand, I found 25 (17.36%) participants in the IMC category, 8 (5.56%) participants in the CMC category, and 90 (62.5%) in the correct category. Interestingly, for *index*, the frequency distribution was higher in the correct categories. As the concept of *index* was taught to the students at the time of this study, this finding is not surprising.

6.4.6.3 *Part*: Type

For the component *type*, I found the majority of participants (N = 140, 97.22%) in the consistent and correct (C) category. Then, I found 41 (44.1%) participants in the correct category. The rest of the participants were dispersed in IMI (N= 3, 2.08%) and CI (N = 1, 0.69%) categories (see Table 6.18). None of the participants had incorrect and inconsistent mental models for *type*.

6.4.6.4 *Part*: Elements

I found the majority of participants (N = 101, 70.14%) in the consistent and correct category (C). Only two participants were found to hold inconsistent and incorrect mental models. The remaining participants were dispersed in the other categories (see Table 6.18). For *elements*, over 85% mental models were in the correct spectrum.

Taken together, the participants' distribution was more inclined towards the correct assertion for the *parts* component. Interestingly, that is not the case for the *state changes* components. Below, I describe the frequency distribution of the participants in each mental model category for the *state changes* components.

6.4.6.5 *State Change*: Declaration

Unlike the *parts* components of arrays, only fourteen participants hold consistent and correct mental models for array *declaration* after they have learned arrays. Half of the participants (N = 72) were found to be in the consistent and mostly incorrect (CMI) category. Next, I found 50 (34.72%) participants in the consistent and mostly correct (CMC) group. The rest of the sample was divided into CI (N = 2, 1.39%), IMI (N = 5, 3.47%), and II (N = 1, 0.69) categories.

6.4.6.6 *State Change*: Instantiation

I found one-third of the participants (N = 51, 35.42%) belonged in correct (C) category for *instantiation*. Consecutively, I found 48 (33.33%) participants in the CMI category, followed by the CI category (N = 14, 9.72%). Strikingly, for *instantiation*, more frequency distribution was found in the incorrect categories (total N = 83, 57.64%). Eight (10.8%) participants were in the category of consistent and mostly correct, and only 2 participants were in the inconsistent and mostly incorrect (IMI) category (see Table 6.18).

6.4.6.7 *State Change*: Assigning Elements

For *assigning elements*, most of the participants belonged to the correct (C) (36.81%) and consistent and mostly correct (CMC) (29.86%) category. Consecutively, 19 (13.9%) and 17 (11.81%) participants were found in the CMI and IMI categories. None of the participants were consistently incorrect for the component *assigning elements*. I found 3 (2.08%) participants in the incorrect and inconsistent category for this component.

6.4.6.8 *State Change*: Array Assignment

When one array reference variable is assigned to another array reference variable, I labeled the *state change* as *assignment*. I found most participants (N = 41, 44.1%) belonged to the inconsistent and mostly correct (IMC) category. Next, I found 52

(36.11%) participants in the IMC category. It is noteworthy to mention that over 50% of the participants were distributed in the incorrect categories. I noticed only seven (4.86%) participants were in the correct category.

The frequency distribution in mental model categories indicates that the *state change* components were more challenging for our participants than the *parts* components.

6.5 Discussion

In this section, I discuss the key findings on novice programmers' mental models after they have learned arrays based on the research questions stated in Section 6.1. A discussion geared towards RQ1 is discussed in Section 6.5.1, RQ2 in Section 6.5.2, and RQ3 in Section 6.5.3.

6.5.1 Novice Programmers' Mental Models after Learning Arrays

To answer my first research question **RQ4. What are the characteristics of novice programmers' mental models after they have learned arrays?** I draw evidence from the frequency distribution of selection of assertions for each component of array's *parts* and *state changes* as well as the correctness scores and frequency distribution of mental model consistency and classification. Below, I discuss the findings reflecting upon the RQ4.

In summary, I can conclude with the following comments on incoming novice programmers' mental models:

Name: Students' mental model on *name* after they have learned arrays seemed well developed in light of correctness and consistency. For the component *name*, similar to the finding from the pre-test, most students consistently selected the right assertion (MN1). The data from Table 6.18 further solidifies this argument. Table 6.18 shows over 90% of the students' mental models were correct and consistent for the component *name*.

Index: Novice programmers' mental models of array indexing seemed to be developing. More practice and exposure are needed to make it well-developed. Over 60% of the students were placed in the mental model category (C), deeming them to be consistent and correct. Although over 96.5% of students selected the correct assertion on the array's last index sometimes, none was found to select it all the time it was presented. Moreover, some of the students were still confused about the start index being one and the end index being n.

***Type*:** The majority of the incoming CS1 students' mental models of array's *type* is well developed.

***Elements*:** Most of the students' mental models are well developed for identifying suitable elements an array can store. The frequency distribution of the selection of assertions was similar to the pre-test results. However, fewer students chose the wrong assertion on the array's size. After instruction, students seem more accurate about the array's size. However, a similar percentage of students in the pre-test chose ME5 consistently, even after instruction. ME5 was identified as a misconception in both the pre-test and post-test (described in Chapter 8). Addressing this misconception in class may reduce the frequency.

***Declaration*:** Similar to the incoming CS1 students' mental models of *declaration*, the outgoing CS1 students' mental models of array *declaration* seemed incomplete based on their selection of assertions and their consistency. Further comparison of the pre-test and the post-test data in Chapter 7 highlights that students' mental model of array *declaration* did not improve significantly.

***Instantiation*:** Similar to *declaration*, students do not have a clear mental model of the state changes that occur after instantiation, such as memory allocation and initialization of default values to the elements. Nearly half of the participants selected sometimes that no memory is allocated after instantiation. Similar to the pre-test data, none of the students could always accurately select the correct default of an

array after instantiation.

***Assigning Elements*:** Majority of students' mental models of *assigning elements* seemed mostly accurate but, for some, not always consistent.

***Assignment*:** Most of the students' mental models of array's *assignment* are incorrect and inconsistent. Over 70% of students who have learned arrays sometimes selected that array of reference assignment copy values.

6.5.2 Parts vs. State Changes

To answer the research question **How are the novice programmers' mental models of the array's *parts* components in comparison with the *state changes* components after they have learned arrays?** I draw evidence from the correctness score and the mental model score. I found data supporting the evidence that the mental model components of *state changes* were more challenging to novice programmers than the *parts*. I found that our participants' correctness score was significantly higher for *parts* components than *states*. Similarly, I found *parts* mental model score significantly lower than *state changes* mental model score. Moreover, for *parts* components, I noticed the frequency distribution of selecting the assertions 100% of the time was skewed to the correct assertion. However, for *state changes* components, the percentage of selection was scattered. I observed a similar pattern for the frequency distribution of mental model categories (see Figure 6.1). From these findings, I can answer that even after classroom instruction, the novice programmers' mental model correctness and consistency for array's *parts* components are better than the *state changes* components.

6.5.3 Students' Demographics and Mental Models

Below, I discuss the following research question **RQ6. What impact do prior programming experience and demographics have on the mental model of the participants in our study?** Interestingly, I found no significant difference be-

tween the correctness and mental model scores based on students' prior programming experience or their learned programming language. The score difference between the students who have learned other programming languages and those who had no experience that existed in the pre-test was not found after the classroom instruction. Moreover, Chapter 5 mentions the gender difference (male students scored higher than female) that existed in the incoming novice programmers' mental models. Interestingly, that gender difference was not found in the post-instruction dataset. I found no significant difference between the correctness and mental model scores based on students' gender. However, the correctness and mental model score difference between the CS majors and non-majors students persist even after classroom instruction. Perhaps CS majors may put more effort into learning programming than those who do not want to pursue CS as an academic major.

6.6 Summary

In this chapter, I presented the results gathered from investigating novice programmers' mental models of arrays as they received classroom instruction on arrays. The use of MMT-A elicited mental model assertions for each of the array components. From the mental model assertion, I analyzed the consistency and correctness of incoming novice programmers' mental models. Surprisingly, even after formal classroom instruction, students' mental models of *state changes* components remained mostly inconsistent and incorrect. Particularly, I only found a handful of students (less than 20 among 144) holding the correct and consistent (abbreviated as C in Table 6.18) mental models for an array *declaration* and *assignment*. The mental model difference between the array's *parts* components and *state changes* components still exists. This may indicate that more emphasis and clear articulation are needed to explain the hidden state changes to students.

The previous chapter presented the results of investigating mental models before classroom instruction, and this chapter presented novices' mental models of arrays

after classroom instruction. The next chapter demonstrates the change in novice programmers' mental models from the pre-test to the post-test.

CHAPTER 7: NOVICE PROGRAMMERS' MENTAL MODEL SHIFTS FROM PRE- TO POST-INSTRUCTION

7.1 Introduction

As part of a journey to explore novice programmers' mental models, I was also interested in exploring how novice programmers' mental models change after classroom instruction. Classroom instruction plays a crucial role in affecting mental models. One of the goals of formal classroom instruction is to shift learners' mental models toward more accuracy. In Chapter 5, I presented my findings exploring novice programmers' mental models before classroom instruction. In Chapter 6, I presented novice programmers' mental models after classroom instruction. In this Chapter, I present my findings based on **RQ7. How do the correctness and consistency of novice programmers' mental models of arrays change after (pre-test vs. post-test) classroom instruction?**

The results showed that overall, the mental model correctness and consistency of arrays improved after classroom instruction. However, even though improved, I found that participants' mental model correctness and consistency of the *parts* components were better than the *state change* components regardless of prior programming experience. Mental model correctness and consistency improved for all the array's components except for *declaration*.

7.2 Data Collection

In Spring 2021, I collected data with MMT-A in a quasi-experimental design. In Section 4.12, I describe the **Spring 2021-Pre-test** and **Spring 2021-Post-test** data collection timeline and procedure. To understand the effect of instruction on

novice programmers' mental models, I used the paired dataset from **Spring 2021-Pre-test** and **Spring 2021-Post-test**. I found 66 participants who participated in both the pre-test and the post-test and consented to data analysis. The students were allowed to complete the MMT-A test as a pre-test during the 6th week of a 14-week semester after the course covered the concepts of data types, literals, primitive variables, assignment operators, casting, classes, objects, constructors, method calling, parameters, and if statements. The instructors asked the students to complete the MMT-A again, this time as a post-test after the instruction on arrays (week 13).

7.3 Participants

The majority of the participants among the sixty-six participants identified as male (47, 71.2%), with 13 (19.7%) identified as female, 1 as gender non-conforming, and the rest (5, 7.57%) preferred not to answer. Of the 66 participants, 32 (48.5%) reported having CS as their study major. Almost half of the participants (32, 48.5%) had no programming experience before enrolling in the CS1 course. Among the 34 participants with prior programming experience, 5 had experience with Java, and 29 had experience with other programming languages (e.g., Python, JavaScript, C#, C++, SQL, HTML, Snap).

7.4 Results

7.4.1 Mental Model Correctness

I performed a paired t-test to compare participants' pre-test and post-test correctness scores. I found that the post-test correctness score ($\bar{x}$ = 25.15, 69.86%, σ = 6.31) was significantly higher ($t = 8.18, df = 65, p < 0.001$) than the pre-test ($\bar{x} = 20.30, 56.39\%, \sigma = 5.76$).

I measured each participant's correctness score change from the pre-test to the post-test. The mean change was 4.85 ($\sigma = 4.81$), where one participant scored seven points lower in the post-test, and one scored fifteen points higher. The correctness

Table 7.1: Pre-test and Post-test Mean Correctness Score of the Components of Arrays. (N = 66)

Components P = part, S = state	Correctness Score				T-test		
	Pre-test		Post-test		Pre vs. Post		
	Mean	SD	Mean	SD	t	df	p
P:Name	87.12%	29.50	93.18%	21.28	1.73	65	$p < 0.05$
P:Index	52.73%	42.91	88.48%	23.55	6.82	65	$p < 0.001$
P:Type	96.21%	13.33	100.00%	0.00	2.31	65	$p < 0.05$
P:Elements	71.52%	27.64	79.70%	27.29	1.93	65	$p < 0.05$
P:Total	70.57%	3.00	87.64%	2.39	6.99	65	$p < 0.001$
S:Declaration	50.76%	25.22	47.73%	29.94	n.s.		
S:Instantiation	36.74%	28.83	62.50%	37.53	-4.72	65	$p < 0.001$
S:Assigning Elements	54.29%	30.71	70.96%	32.20	4.19	65	$p < 0.001$
S:Assignment	44.19%	23.84	54.29%	22.13	3.72	65	$p < 0.001$
S:Total	47.36%	3.55	58.54%	4.50	5.85	65	$p < 0.001$
Total Score	56.39%	5.76	69.86%	6.31	8.18	65	$p < 0.001$

score from the pre-test to the post-test decreased for 13.6% of the participants. On the other hand, I observed an increase in the correctness score for 80.3% of the participants. For 6% (4) of the participants, their correctness scores did not change from the pre-test to the post-test.

As can be seen from Table 7.1 (P:Total), the participants' post-test *parts* score ($\bar{x} = 87.64\%, \sigma = 2.39$) is significantly higher ($t = 6.99, df = 65, p < 0.001$) from their pre-test scores ($\bar{x} = 70.57\%, \sigma = 3.00$).

Additionally, the difference between the pre-test and post-test correctness scores for *state change* was statistically significant (t-test $t = 5.85, df = 65, p < 0.001$) (S:Total in Table 7.1). Participants' mean correctness score for the *state change* components was higher in the post-test ($\bar{x} = 58.54\%, \sigma = 4.50$) than the pre-test ($\bar{x} = 47.36\%, \sigma = 3.55$).

I compared the correctness score for each component of *parts* and *state changes*. Table 7.1 shows the participants' pre-test and post-test means and standard deviations (SD) for all the components of *parts* and *state changes*. For the component *name*, a

paired t-test (t-test significant at $\alpha = 0.05, t = 1.73, df = 65, p = 0.04$) revealed that participants' post-test scores ($\bar{x} = 93.18\%, \sigma = 21.28$) were significantly higher than their pre-test score ($\bar{x} = 87.12\%, \sigma = 29.50$). Similarly, I found an increase for the *part* component *index* (t-test significant at $\alpha = 0.001, t = 6.82, df = 65, p < 0.001$), *type* (t-test significant at $\alpha = 0.05, t = 2.31, df = 65, p = 0.01$), and *elements* (t-test significant at $\alpha = 0.05, t = 1.93, df = 65, p = 0.03$).

For the *state change* components, I found that the participants' post-test scores were significantly higher than their pre-test scores: *instantiation* ($t = -4.72, df = 65, p < 0.001$), *assigning elements* ($t = 4.19, df = 65, p < 0.001$), and *assignment* ($t = 3.72, df = 65, p < 0.001$). I found no significant difference between the mean correctness score of the pre-test and the post-test for *declaration*.

7.4.1.1 Correctness Level

As mentioned in Section 4.7, I categorized correctness into four levels: incorrect, mostly incorrect, mostly correct, and correct. Table 7.2 shows the classification of the correctness levels for the pre-test and post-test. In either pre-test or post-test, I found no participant in the *incorrect* category (all incorrect answers). The scores show that 36.4% of the participants in the pre-test scored in the *mostly incorrect* level, and in the post-test 13.6% participants scored in this level. In the pre-test, 63.6% of participants were at the *mostly correct*, and 84.8% scored at this level in the post-test. Finally, no participant answered all the questions correctly in the pre-test, and only one did so in the post-test.

7.4.2 Mental Model Consistency

I labeled participants' components of *parts* and *state changes* consistent or inconsistent based on the definition mentioned in Section 4.5. Table 7.3 shows the consistency classification (in percentages) for each component in the pre-test and post-test.

I performed a crosstabulation (see Table 7.4) to analyze the shift of mental model

Table 7.2: Frequency distribution of participants from the pre-test and the post-test across different correctness levels.

Correctness Level	Pre-Test		Post-Test	
	n	%	n	%
Incorrect	0	0.0%	0	0.0%
Mostly Incorrect	24	36.4%	9	13.6%
Mostly Correct	42	63.6%	56	84.8%
Correct	0	0.0%	1	1.5%
Totals	66	100.0%	66	100.0%

Table 7.3: Mental Model Consistency Scores per Component: Pre-test and Post-test (N = 66).

Components P=part, S=state	Mental Model Consistency				McNemar's-test
	Pre-test		Post-test		Pre vs. Post
	Consistent	Inconsistent	Consistent	Inconsistent	p
P:Name	89.39%	10.61%	92.42%	7.58%	n.s.
P:Index	51.52%	48.48%	74.24%	25.76%	$p < 0.05$
P:Type	92.42%	7.58%	100.00%	0.00%	n/a
P:Elements	93.94%	6.06%	98.48%	1.52%	n.s.
S:Declaration	93.94%	6.06%	84.38%	15.62%	n.s.
S:Instantiation	74.24%	25.76%	83.33%	16.67%	n.s.
S:Assigning Elements	68.18%	31.82%	78.79%	21.21%	n.s.
S:Assignment	25.76%	74.24%	25.76%	74.24%	n.s.

consistency from the pre-test to the post-test. Below, I describe the shift for each component of *parts* and *state changes*.

For the *part name*, consistency status did not change significantly from the pre-test to the post-test (McNemar's test not significant at $\alpha = 0.05$ level, $p = 0.75$). For the *name* component most participants remained consistent in the post-test.

I found *index* to have a significant shift. A McNemar's test determined a statistically significant difference in the proportion of mental model shift for *index* based on consistency in the pre-test and post-test at $\alpha = 0.05$ level, $p = 0.006$.

For the part *type*, only 5 participants were found to have an inconsistent mental model in the pre-test. After the post-test, all of them shifted to the consistent mental model. Because the frequency of the consistent participants in the post-test was zero, I could not perform McNemar's test.

The mental model shifts from the pre-test to the post-test for the *part elements* were not statistically significant (McNemar's test). Similarly, I found no statistically significant difference (McNemar's test) in the proportion of mental model consistency shifts for the *state change declaration*, *instantiation*, nor *assigning elements*.

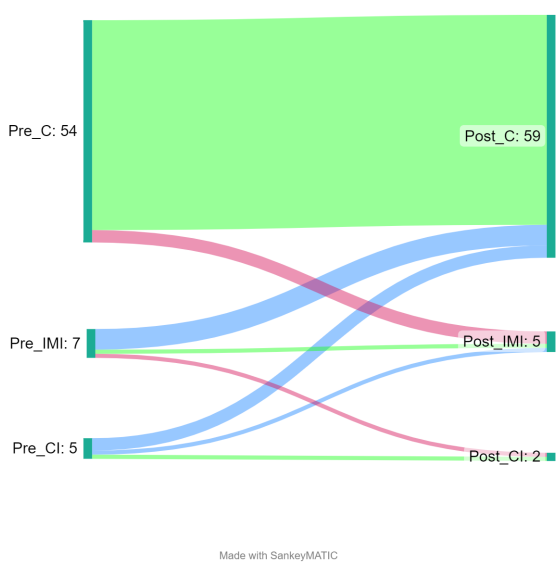
For the state change *assignment*, I did not find a statistically significant difference (McNemar's test) in the proportion of mental model shifts. Out of the 49 participants who held inconsistent mental models, most (38) remained inconsistent even after classroom instruction of arrays. Only 11 of the 49 participants' mental models shifted from inconsistent to consistent. On the other hand, eleven participants shifted from consistent to inconsistent. Only Six participants remained consistent in the post-test.

7.4.3 Mental Model Classification

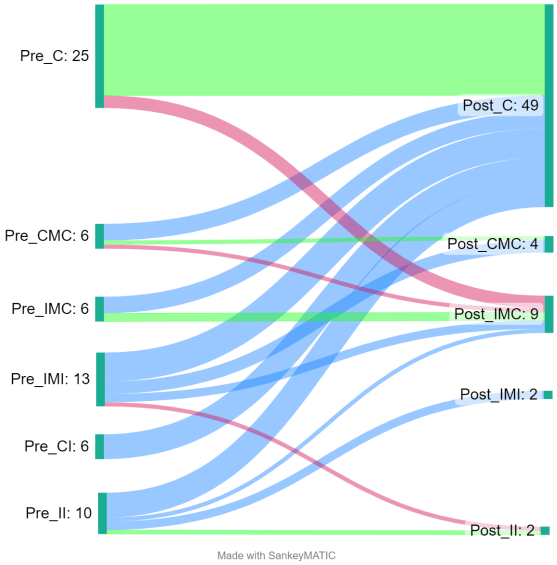
Below, I describe the participants' mental model classification shift from the pre-test to the post-test for each component of *parts* and *state changes*. I label the increase in mental model rank from the pre-test to the post-test as a positive shift, a decrease as a negative shift, and ranks remaining the same as neutral. Table 7.5 portrays the

Table 7.4: Crosstabulation showing the changes in participants' frequency of mental model consistency shift from pre-test to post-test for each component.

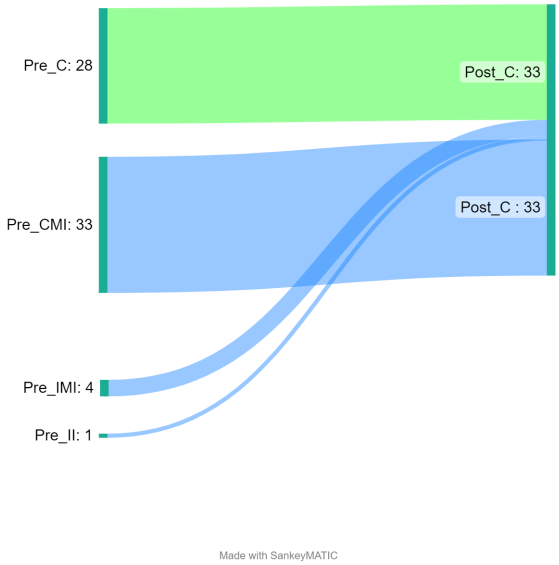
		Post-test		
		Inconsistent	Consistent	Total
Part: Name				
Pre-test	Inconsistent	1	6	7
	Consistent	4	55	59
	Total	5	61	66
Part: Index				
Pre-test	Inconsistent	11	21	32
	Consistent	6	28	34
	Total	17	49	66
Part: Type				
Pre-test	Inconsistent	0	5	5
	Consistent	0	61	61
	Total	0	66	66
Part: Element				
Pre-test	Inconsistent	1	3	4
	Consistent	0	62	62
	Total	1	65	66
State Change: Declaration				
Pre-test	Inconsistent	1	3	4
	Consistent	9	51	60
	Total	10	54	64
State Change: Instantiation				
Pre-test	Inconsistent	4	13	17
	Consistent	7	42	49
	Total	11	55	66
State Change: Assigning Elements				
Pre-test	Inconsistent	7	14	21
	Consistent	7	38	45
	Total	14	52	66
State Change: Assignment				
Pre-test	Inconsistent	38	11	49
	Consistent	11	6	17
	Total	49	17	66



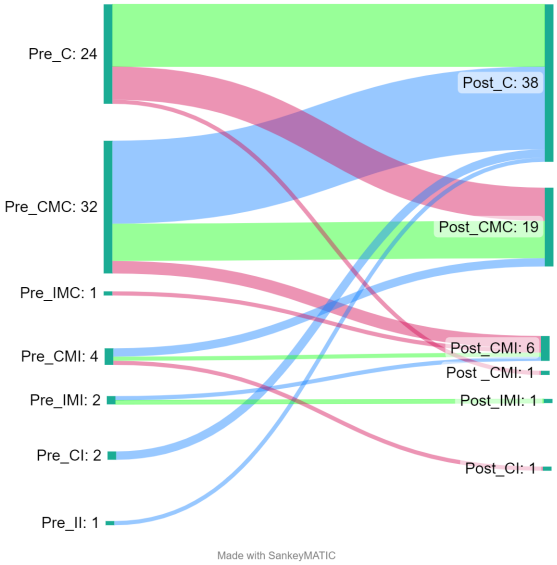
(a) Name



(b) Index



(c) Type

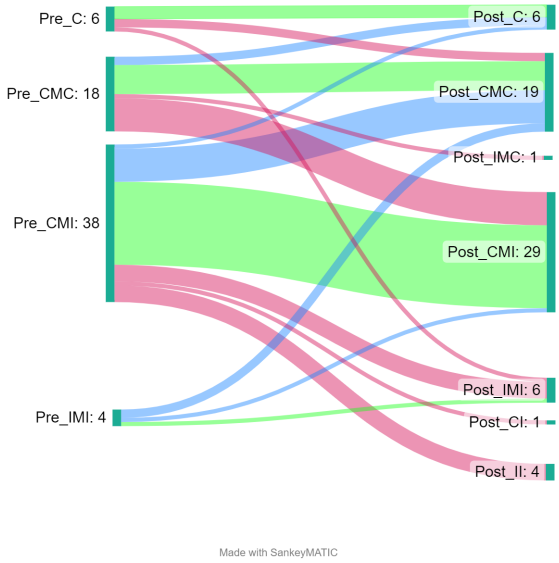


(d) Elements

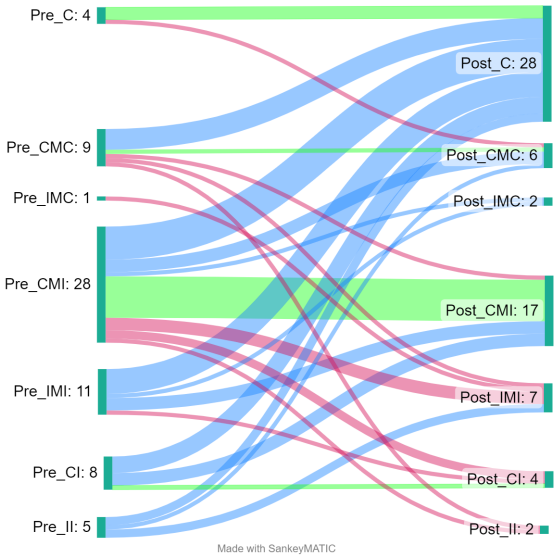
Figure 7.1: Mental model classification shift for the four *part* components. Here, we have portrayed positive rank shifts (color blue), negative rank shifts (color red), and neutral (color green). As an example, 54 participants' mental models were correct in the pre-test (Pre_C), and 59 participants' mental models were correct in the post-test (Post_C) for *name*.

frequencies of participants in all three kinds of shifts for all eight components.

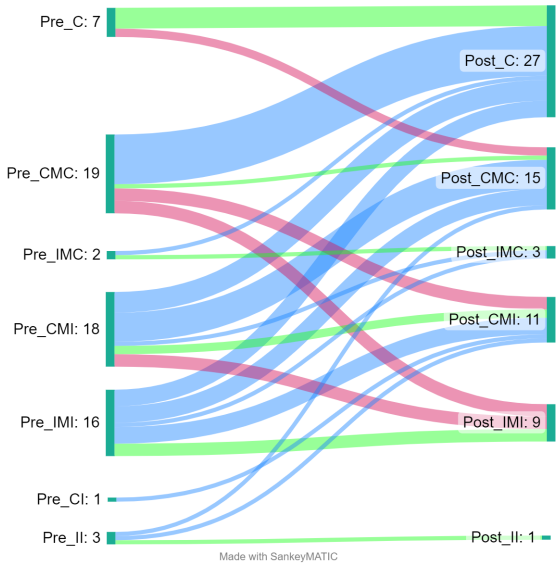
Part Name: A Wilcoxon signed-rank test did not find a significant change in mental model classification shifts ($Z = -1.76, p = 0.078$). For *name*, I found that most



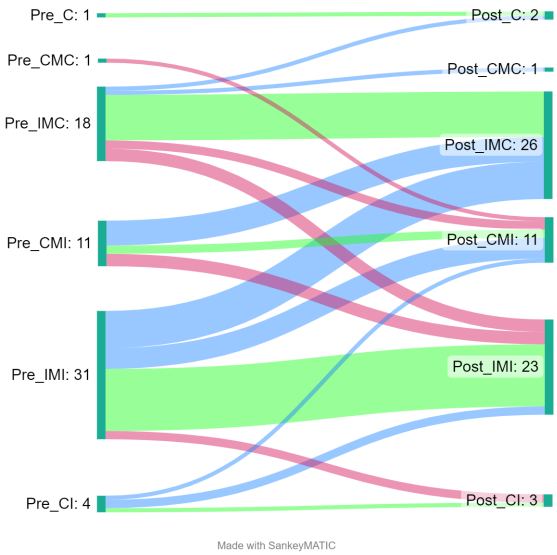
(a) Declaration



(b) Instantiation



(c) Assigning Elements



(d) Assignment

Figure 7.2: Mental model classification shift for the four *state change* components. Here, we have portrayed positive rank shifts (color blue), negative rank shifts (color red), and neutral (color green).

participants (53 of the 66 participants) mental model classification did not shift from the pre-test to the post-test. A deeper analysis revealed that, out of those 53 participants, 51 mental models were correct in the pre-test and remained correct in the post-test (C to C).

Part Index: A Wilcoxon signed-rank test showed a statistically significant shift in the mental models ($Z = -4.89, p < 0.001$) for *index*. Thirty-seven participants experienced a positive mental model shift after the classroom instruction of arrays *index*. Figure 7.1b shows the variation of the shifts. On the other hand, five participants had a negative mental model shift. The remaining 24 participants had no change in their mental model classification.

Part Type: Similar to *index*, I found a statistically significant change in the mental model shifts for *type* ($Z = -5.85, p < 0.001$). I found that most pre-test participants (n = 38) increased their mental model ranks. I observed one participant's mental model shifting from inconsistent and incorrect (II) to correct (6) (Figure 7.1c). Twenty-eight participants' mental models remained correct (C) in the post-test.

Part Elements: I did not find a statistically significant change in the mental model shifts ($Z = -1.41, p = 0.16$) for *elements*. I observed that 26 participants had no change in mental models, and 26 had a positive shift. On the other hand, 14 participants experienced a negative mental model shift (details are in Figure 7.1d).

State Change Declaration: I did not find a statistically significant change in the mental model shifts ($Z = -0.65, p = 0.51$) for the state change *declaration*. For state change *declaration*, most of the participants (n = 30) mental models did not shift. Twenty participants' mental models shifted into negative ranks (details are in Figure 7.2a). Only 14 participants' mental models shifted into positive ranks.

State Change Instantiation: I found a statistically significant change in the mental model shifts for array *instantiation* ($Z = -4.05, p < 0.001$). Most participants (n = 39) mental models shifted into positive ranks for *instantiation* (details are in

Table 7.5: Frequencies of participants across positive, negative, neutral mental model classification shift with the p value of Wilcoxon Signed-rank test.

Components P: part S: state changes	Mental Model Classification Shift				Wilcoxon Signed-Rank Test
	Positive Shift	Negative Shift	Neutral	Total	p
P:Name	9	4	53	66	n.s.
P:Index	37	5	24	66	$p < 0.001$
P:Type	38	0	28	66	$p < 0.001$
P:Elements	26	14	26	66	n.s.
S:Declaration	14	21	31	66	n.s.
S:Instantiation	39	12	15	66	$p < 0.001$
S:Assigning Elements	42	11	13	66	$p < 0.001$
S:Assignment	25	11	30	66	$p < 0.05$

Figure 7.2b). Fifteen participants' mental models remained the same. Conversely, 12 participants' mental models shifted into a negative rank.

State Change Assigning Elements: I found a statistically significant change in the mental model shifts for *assigning elements* ($Z = -3.86, p < 0.001$). Forty-two participants experienced positive shifts. Thirteen participants' mental models did not shift. Eleven participants' mental models shifted to negative ranks. Figure 7.2c shows the mental model classification shift details.

State Change Assignment: A Wilcoxon signed-rank test showed a statistically significant change in mental models ($Z = -2.27, p < 0.05$) from the pre-test to the post-test for the state change *assignment*. Twenty-five participants mental model ranks improved after classroom instruction, although scattered across various groups (see Figure 7.2d). Eleven participants' mental model ranks decreased in the post-test. For *assignment*, thirty participants' did not have a shift in their mental models.

7.4.4 Mental Model Score

I paired each participant's pre-test mental model score and post-test mental model to analyze the difference. By conducting a paired t-test, I found that participants' mental model scores in the post-test were significantly higher than those in the pre-

Table 7.6: Pre-test and Post-test Frequency Distribution across the Mental Model Categories. (N = 66)

Components P = part, S = state	II		CI		IMI		CMI		IMC		CMC		C	
	Pre-test	Post-test	Pre-test	Post-test	Pre-test	Post-test	Pre-test	Post-test	Pre-test	Post-test	Pre-test	Post-test	Pre-test	Post-test
P:Name	0.00%	0.00%	7.58%	3.03%	10.61%	7.58%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	81.82%	89.39%
P:Index	18.18%	3.23%	9.09%	0.00%	19.70%	3.23%	0.00%	0.00%	9.09%	14.52%	9.09%	9.68%	34.85%	69.35%
P:Type	1.52%	0.00%	0.00%	0.00%	6.06%	0.00%	50.00%	0.00%	0.00%	0.00%	0.00%	0.00%	42.42%	100.00%
P:Elements	1.52%	0.00%	3.03%	1.52%	3.03%	1.52%	6.06%	10.61%	1.52%	1.01%	48.48%	28.79%	36.36%	57.58%
S:Declaration	0.00%	4.69%	0.00%	1.56%	6.06%	9.38%	57.58%	45.31%	0.00%	1.56%	27.27%	28.13%	9.09%	9.38%
S:Instantiation	7.58%	3.03%	12.12%	6.06%	16.67%	10.61%	42.42%	25.76%	1.52%	3.03%	13.64%	9.09%	6.06%	42.42%
S:Assigning Elements	4.55%	1.54%	1.52%	0.00%	24.24%	13.85%	27.27%	16.92%	3.03%	4.62%	28.79%	23.08%	10.61%	40.00%
S:Assignment	0.00%	0.00%	6.06%	4.55%	46.97%	34.85%	16.67%	16.67%	27.27%	39.39%	1.52%	1.52%	1.52%	3.03%

test ($t = 10.02$, df = 59, $p < 0.001$). Furthermore, participants' mental model scores broken down to the *parts* ($t = 9.34$, df = 61, $p < 0.001$) and *state changes* ($t = 5.42$, df = 63, $p < 0.001$) components were significantly higher in the post-test than the pre-test. When I analyzed the direction of the mental model score change from pre-test to post-test, I found that 90% of the participants' mental model scores increased.

7.4.5 Previous Programming Experience and Mental Models

I analyzed participants' mental model scores and their programming experience (details are in Table 7.7). Based on the previous programming experience, I separated participants' scores into three groups. The participants in the *No prior programming* group (n = 32) reported no prior programming experience. Some participants (n = 5) learned Java before enrolling in our course and thus were placed into the *Java programming* group. The reason for enrolling in this course was either that it was a degree requirement or they were retaking this course. The remaining participants (n = 29) had various programming backgrounds (e.g., Python, JavaScript, C#, C++, SQL, HTML, Snap), grouped in the *Other programming languages* group.

In the next subsections, I present the analysis of participants' correctness and mental model scores based on their prior programming experience.

7.4.5.1 Correctness Score

A one-way ANOVA revealed a statistically significant difference at $\alpha = 0.05$ level ($F(2, 65) = 7.17, p = 0.002$) in participants' correctness scores based on the prior pro-

gramming language experience. Tukey HSD Test for multiple comparisons found that participants with experience with programming languages other than Java scored significantly higher than participants without programming experience ($p = 0.001, 95\%$ C.I. = [-8.32, -1.82]) at $\alpha = 0.05$ level. However, I found no statistically significant difference in the correctness scores between *Java programming*, *No prior programming*, and *Java programming* and the *Other programming languages* group. Before any classroom instruction, participants in the *No prior programming* group answered 50% of questions of the MMT-A correctly. The few participants in the *Java programming* group's correctness score mean was 52.79%. The participants with previous programming experience with other languages scored the highest mean (64.08%). I found a similar difference when I investigated the breakdown of the total correctness score of *parts* and *state changes*. Participants from the *Other programming languages* group scored statistically significantly higher than the *No prior programming* group in the correctness score for *parts* (One-way Anova: $F(2, 65 = 8.76), p < 0.001$; Tukey HSD Test: $p < 0.001, 95\%$ C.I. = [-4.53, -1.22]) at $\alpha = 0.001$ level and for *state changes* (One-way Anova: $F(2, 65 = 3.94), p = 0.03$; Tukey HSD Test: $p = 0.04, 95\%$ C.I. = [-4.29, -0.10]) at $\alpha = 0.05$ level.

I performed the same analysis in the post-test. In the post-test, I observed differences in the scores of those with prior programming experience and those without. The participants from the *Other programming languages* group scored significantly higher than the *No prior programming* group in the total correctness score (One-way Anova: $F(2, 65 = 3.72), p < 0.05$; Tukey HSD Test: $p = 0.03, 95\%$ C.I. = [-7.73, -0.29]) and correctness score for *parts* (One-way Anova: $F(2, 65 = 3.89), p < 0.05$; Tukey HSD Test: $p = 0.04$, 95% C.I. = [-2.86, -0.04]) at $\alpha = 0.05$ level. Unlike the pre-test, I found no statistically significant difference in the *state changes* correctness score.

7.4.5.2 Mental Model Score

Similar to the correctness score, in the pre-test, the *Other programming languages* group scored the highest (68.69%) (see Table 7.7) on their mental model score (MMS). I performed one-way ANOVA to find the various programming experience group differences in participants' mental model scores. Post hoc analysis revealed that the participants' mental model scores were statistically significantly higher in the *Other programming languages* group than the *No prior programming* group (One-way Anova: $F(2, 65 = 6.00), p < 0.05$; Tukey HSD Test: $p = 0.003$, 95% C.I. = [-8.33, -1.51]) at $\alpha = 0.05$ level. A similar difference was noticed in the mental model scores for *parts* (One-way Anova: $F(2, 65 = 6.72), p < 0.05$; Tukey HSD Test: $p = 0.002$, 95% C.I. = [-5.77, -1.11]). However, participants' mental model scores for the *state changes* were not significantly different between the *No prior programming* and the *Other programming languages* groups. The difference between the mental model scores of those who learned Java and the rest of the groups was not statistically significant.

The difference in the mental model score across the *Other programming languages* group and *No prior programming* group was also present in the post-test. Participants' mean mental model score in the post-test from the *Other programming languages* group was significantly higher than the mean mental model score of the *No prior programming* group (One-way Anova: $F(2, 65 = 3.72), p < 0.05$; Tukey HSD Test: $p = 0.04$, 95% C.I. = [-8.51, -0.18]) at $\alpha = 0.05$ level. However, when I observed the difference in mean mental model score for *parts* and *state changes*, I found no differences in prior programming experience.

7.5 Discussion

7.5.1 Mental model correctness and consistency change at the end of the course

The results indicated that the participants' mental model correctness and consistency of arrays improved at the end of the semester. Overall, the correctness and

Table 7.7: Mental model correctness score, correctness score for *parts*, *state changes*, mental model score, mental model score for *parts*, *state changes* across different programming background from the pre-test and the post-test.

	Programming Language Learned	N	Pre-test			Post-test		
			Mean (%)	SD	Tukey HSD	Mean (%)	SD	Tukey HSD
Correctness Score	(a) No Prior Programming	32	50.00	6.04	c >a	65.19	5.86	c >a
	(b) Java Programming	5	52.78	5.66	$p < 0.05$	62.22	8.88	$p < 0.05$
	(c) Other Programming Languages	29	64.08	4.22		76.33	5.74	
Correctness Score Parts	(a) No Prior Programming	32	60.50	3.09	c >a	83.48	2.40	c >a
	(b) Java Programming	5	74.29	3.51	$p < 0.001$	78.57	3.46	$p < 0.05$
	(c) Other Programming Languages	29	81.00	2.01		93.84	1.92	
Correctness Score State Changes	(a) No Prior Programming	32	43.32	3.83	c >a	53.55	4.03	n.s.
	(b) Java Programming	5	39.09	2.30	$p < 0.05$	51.82	5.77	
	(c) Other Programming Languages	29	53.27	3.01		65.20	4.51	
MMS	(a) No Prior Programming	32	58.73	6.29	c >a	74.29	6.44	c >a
	(b) Java Programming	5	64.58	5.39	$p < 0.05$	70.15	14.74	$p < 0.05$
	(c) Other Programming Languages	29	68.96	4.59		83.33	5.56	
MMS Parts	(a) No Prior Programming	32	66.42	4.38	c >a	88.98	3.30	c >a
	(b) Java Programming	5	80.00	3.42	$p < 0.05$	81.94	6.66	$p < 0.05$
	(c) Other Programming Languages	29	80.75	3.05		96.13	2.24	
MMS State Changes	(a) No Prior Programming	32	51.04	3.15	n.s.	60.00	4.00	n.s.
	(b) Java Programming	5	49.17	2.59		50.00	6.44	
	(c) Other Programming Languages	29	57.17	2.84		70.12	4.17	

mental model scores improved significantly in the post-test. I also observed improvements in the scores of *parts* and *state changes*. The participants' total correctness and mental model scores for *parts* improved significantly at the end of the semester. Similar improvements were found for the scores of *state changes*. Below, I summarize the mental model change in correctness and consistency for all the eight components of *parts* and *state changes*.

Part Name: I found most of the participants' mental models of *name* to be correct and consistent. The participants' mental model correctness and consistency for *name* were already developed before the classroom instruction on arrays. A possible explanation might be that the component *name* was already taught when they learned primitive variable *name*. Therefore, they could easily relate to array reference variable *name*.

Part Index: The results showed the mean total correctness score improved significantly (52.73% to 88.48%) for *index* at the end of the semester (see Table 7.1). Moreover, participants' mental model consistency shifted significantly from inconsistent to consistent at the end. Overall, I can conclude that the participants' mental model correctness and consistency improved at the end of the semester.

Part Type: The total correctness score improved significantly from the pre-test to the post-test for *type*. At the end of the semester, all of our participants' mental models were consistent and correct for *type*, with 38 participants making a positive significant shift in the post-test. A possible explanation might be that similar to *name*, the notion of *type* was taught and practiced for primitive variables. Therefore, at the semester's end, they are not confused about *type*. This finding, along with the findings for *name*, may support the constructivist mental model (mentioned in Section 2.1.4) where learners utilized their existing knowledge of *name* and *type* to relate in a new concept 'arrays'.

Part Elements: Similar to the other *parts* components, the mean total correctness

score for *elements* was significantly higher in the post-test. Out of the 66 participants, 61 had consistent mental models in the pre-test, and they remained consistent in the post-test (see Table 7.4). *Elements* being a new concept, I did not see as many participants in the correct mental model category compared to the other *parts* components.

State change Declaration: Surprisingly, for *declaration*, the total correctness score did not improve at the end of the semester. The mental model consistency shift and mental model classification shift were not significant. Most of the participants were found to hold consistent and mostly incorrect (CMI) mental models in the pre-test and the post-test (see Table 7.6). As Figure 7.2a shows, mental model shifts were distributed across various categories. As I included questions in the MMT-A about the hidden aspects of arrays (default value after the declaration, memory allocation), the mental model assertions may not be strongly ingrained in our participants yet. Array declaration is the same as a reference variable declaration. Previous studies have found that students have several misconceptions regarding memory management after a reference variable declaration [28, 123].

State change Instantiation: Participants' correctness scores significantly improved (36.74% to 62.50%) at the end of the semester for *instantiation*. Mental model consistency did not shift significantly. However, I observed a significant shift in mental model classification. 39 participant shifted their mental models to higher ranks. At the end of the semester, 42.42% had correct mental models, which increased from 6.06% in the pre-test.

State change Assigning Elements: I observed participants' correctness scores significantly improve at the end of the semester for *assigning elements*. Mental model consistency did not shift significantly. However, I observed a significant shift in the mental model classification. Forty-two participants' mental models shifted positively. At the end of the semester, 40% had correct mental models, which increased from

10.61% in the pre-test.

State change Assignment: Though our participants' mental model correctness score increased (44.19% to 54.29%) significantly in the post-test, the mean score of the post-test remained low for *assignment*. Mental model consistency did not shift significantly. Interestingly, unlike the other components, most participants held inconsistent mental models in the pre-test (74.24%) and the post-test (74.24%). As Gawronski et al. [42] said, mental model inconsistency serves as a cue for potential errors. When I looked at the mental model classification, I found more participants holding inconsistent and mostly correct (IMC). I observed only 3.03% of participants holding the correct (C) mental model at the end of the semester. Previous studies [17, 152] have found understanding reference variable assignments challenging for students. The findings suggest that at the end of the CS1 course, mental model correctness and consistency of an array *assignment* may not be strongly developed yet in our participants.

7.5.2 Parts vs. State Changes

I found evidence that the mental model components of *state changes* were more challenging to our CS1 students than the *parts*. I found that the participants' correctness score was significantly higher for *parts* components than *state changes*, both in the pre-test and the post-test. Similarly, I found *parts* mental model score significantly higher than *state changes* mental model score. Moreover, from Figure 7.1 and 7.2, I can notice that while the mental model classification shifts for *parts* were uniform, the mental model classification shifts for *state changes* were scattered across various kinds of shifts from the pre-test to the post-test. In particular, I observed lesser improvements in *declaration* and *assignment*. It may indicate that where mental model correctness and consistency have already developed for the *parts* components, it may not be true for the dynamic, hidden *state changes*.

7.5.3 Impact of Prior Programming Experience

I found significant differences in the participants' total correctness scores based on prior programming experience. Participants who have learned other programming languages scored significantly higher in the MMT-A, which remained significant even at the end of the semester. I found similar findings for the correctness score for *parts*, total mental model score, and mental model score for *parts* (see Table 7.7). However, I observed no difference in the total correctness and mental model scores for *state changes*. The dynamic *state changes* hidden under abstraction layers seemed equally hard for the participants with or without prior programming experience.

7.6 Summary

To summarize the answer of the **RQ7**, I can conclude that the overall correctness and consistency of novice programmers' mental models of arrays improved after classroom instruction. Even though I observed improvement in all the major components of array's *parts* and *state changes*, students' mental models of the *state change* component array *declaration* did not improve after classroom instruction. In classrooms and labs, students usually instantiate and initialize arrays right after declaration to serve the purpose of an assignment. Hence, it gives students less opportunity to understand and manipulate an array after declaration. Moreover, in most practices, array declaration and instantiation are shown with a single line (e.g., `int[] scores = new int[5];`); in these cases, the hidden state changes that are happening underneath the line of code after declaration are often not explained or visible to students. This finding and observation again strengthen the evidence that the dynamic *state changes* are harder to understand than to understand the structure (the *parts*) of a programming concept regardless of prior programming experience. The results presented in this chapter also supported the claim. Even after classroom instruction, even though improved, the mental model gap between array's *parts* and *state changes* remains

intact.

In previous chapters, I presented novice programmers' mental models of arrays before and after classroom instruction. In this chapter, I demonstrated how classroom instruction impacted their mental models. While eliciting their mental models, I also uncovered many misconceptions residing in novice programmers' mental models. In the next chapter, I present misconceptions I found in novice programmers' mental models before and after classroom instruction.

CHAPTER 8: MISCONCEPTIONS IN NOVICE PROGRAMMERS' MENTAL MODELS

A portion of the results presented in this chapter has been published in the Proceedings of 2019 IEEE Frontiers in Education Conference (FIE '23). Full citation can be found here [168].

8.1 Introduction

Misconceptions are false, persistent beliefs contradicted by established scientific evidence. [13]. Students enter into introductory programming courses with mental models filled with misconceptions, intuitive beliefs, or commonsense notions, typically generated from prior knowledge from domains such as mathematics and natural languages [8, 28, 110]. The faulty intuitive beliefs generated from pre-existing related knowledge structures often get embedded in their mental models and, if not refuted in the classroom [104], continue to remain a mistake in future courses. Hence, numerous computing educators consider research on novice programmers' misconceptions vital [8, 29, 110, 169].

My doctoral research aimed to elicit assertions of novice programmers' mental models of arrays. While analyzing the consistency of the assertions, I uncovered several misconceptions. I define a mental model as a collection of assertions, and I define a consistently chosen wrong assertion as a misconception. I report misconceptions among students from the before-instruction sample, after-instruction sample, and the change in misconceptions (paired pre-test post-test). Knowledge about incoming CS1 students' (before instruction) misconceptions can allow educators to address them with instruction or other educational interventions. Knowledge about CS1 students'

misconceptions after instruction can provide the CS1 educators the opportunity to reflect on their instructional strategy and the CS2 educators an opportunity to clarify the misconceptions. The misconception change among the paired pre-test and post-test students revealed the direction of change after classroom instruction.

I placed 30 wrong assertions multiple times as distractors in the MMT-A. In this chapter, I report and discuss the misconceptions of arrays found to be held by novice programmers before and after classroom instruction on arrays. Nine misconceptions were documented for *parts* components and *seven* for state changes before classroom instruction. Six misconceptions were documented for *parts* components and six for *state changes* after classroom instruction. Our results show that over half of our participants held at least one misconception before and after learning arrays in classrooms. Novice programmers mostly held misconceptions about the arrays' declarations (state change) both as incoming CS1 students and when they have learned arrays. After classroom instruction, the number of students holding misconceptions about the *parts* components decreased. However, for the *state changes* components, in most cases, the number of students holding misconceptions increased even after classroom instruction.

8.2 Methodology

As mentioned in Section 4.9, I defined a wrong assertion as a misconception when that assertion was presented more than once, and a participant selected that assertion every time. To present the misconceptions residing in incoming CS1 students' (before classroom instruction on arrays) mental models, I utilized **Spring 2021-Pre-test** (N = 93) data set (described in Section 4.12). To present the misconceptions residing in CS1 students after classroom instruction on arrays, I utilized **Spring 2021-Post-test**, **Spring 2023-Post-Instruction**, and **Summer 2023-Post-Instruction** data sets (described in Section 4.12) (N = 144). I report the misconceptions found in the participants before and after classroom instruction on arrays in Section 8.3.1.

Moreover, I also analyzed each participant's change in misconception by pairing their data among the **Spring 2021-Pre-test** and **Spring 2021-Post-test** data set (N = 66). I present the results of the change in misconceptions in Section 8.3.2.

8.3 Results

8.3.1 Misconceptions Before and After Classroom Insruction

From the data I collected, I found 16 misconceptions among the participants of before-classroom instruction and 12 misconceptions among the participants of after-classroom instruction. Table 8.1 shows these misconceptions and the percentage of participants that selected them. As a reminder, these wrong assertions were consistently selected when presented, and thus, I labeled them as misconceptions. In the before-instruction data set, I found nine misconceptions for *parts* and seven misconceptions for the *state changes*. In the after-instruction data set, I found six misconceptions for *parts* and six misconceptions for the *state changes*.

Table 8.1: List of Misconceptions found in the study. Each misconception is labeled with a unique identifier used as reference throughout the dissertation. The last column shows the percentage of participants holding the misconception.

Component Name	Type	Misconceptions	Before Instruction(%)	After Instruction(%)
Name	Part	MN2: Students think type is the name.	12.9%	2.1%
Index	Part	MI3: There is no indexing.	2.2%	1.4%
		MI4: Indexing starts at 1.	4.3%	0%
		MI5: Indexing ends with n.	2.2%	0%
		MI7: Students think the index is the element	1.1%	0%
Type	Part	MT3: The array name is the type of the array.	1.1%	0.7%
Element	Part	ME5: Elements and array names are semantically related.	12.9%	7.6%
		ME6: Keyword 'new' is an element in the array.	2.2%	0.7%
		ME7: No understanding about data type of elements.	1.1%	1.4%
Declaration	State Change	MD2: There is no default value (blank/no value).	21.5%	27.8%
		MD3: The default value for the array reference is the default value for type (e.g., int is 0, boolean is false).	0%	1.4%
		MD4: The default value for the array reference is stored as ?	4.3%	0%
		MD5: After declaration, memory is allocated for the elements.	4.3%	0%
		MD6: After declaration, the number of elements that can be stored is unlimited.	2.15%	5.6%
Instantiation	State Change	MIn2: After instantiation, a question mark (?) is stored as a default value.	2.2%	2.8%
Assigning Elements	State Change	MAE5: Students think primitive assignment is the same as reference assignment.	9.7%	4.2%
Assignment	State Change	MA3: Assignment copies the values.	7.5%	10.4%

The 4th column of Table 8.1 (label: Before Instruction (%)) presents the percentage of participants who selected each misconception. The three most common misconceptions from before-instruction data were: *MD2: There is no default value* (21.51%), *MN2: Students think type is the name* (12.90%), and *ME5: Elements and array names are semantically related* (12.90%). Similarly, the three most common

misconceptions from after-instruction data were: *MD2: There is no default value* (27.8%), *MA3: Assignment copies the values* (10.4%), and *ME5: Elements and array names are semantically related* (7.6%). In the following paragraphs, I describe each of the misconceptions found in our data.

8.3.1.1 *Parts*: Name, Type

Two of the *parts* components produced misconceptions in ways that were complementary to each other. For the part *name*, 12.9% participants had the misconception that an array's type was the name of the array (MN2) before classroom instruction. Conversely, 1.08% of the participants believed that the array's name was the type of the array (MT3). A similar finding was observed in the after-instruction dataset. 2.1% participants had the misconception that an array's type was the name of the array (MN2) after classroom instruction. Conversely, 0.7% of the participants believed that the array's name was the type of the array (MT3).

8.3.1.2 *Part*: Index

For the component *index*, I found four misconceptions in the before-instruction data set and one misconception in the after-instruction data set. I found that 2.15% of the participants before instruction held a misconception that indexing does not exist in arrays (MI3). Even after classroom instruction, 1.4% of participants still held the misconception. Before instruction, 4.30% of participants held the misconception that indexing starts at 1 (MI4). Similarly, 4.3% participants believed array indexing ends at n (MI5). Further analysis revealed that only one participant had the misconception that array indexing starts at one and ends with n (MI4 and MI6). Moreover, 1.08% of participants thought the element was the index (MI7) before classroom instruction.

8.3.1.3 *Part*: Elements

I asked questions about an element of integer arrays named `books` and `gadget`. The strings "Harry Potter" and "Smartwatch" were placed as options to probe the

misconception that an array name is semantically related to its element (ME5) (An exemplary question can be seen in Figure 10.3). 12.9% of the participants from the before-instruction dataset held this misconception. Even after classroom instruction, 7.6% of the participants held the misconception of ME5. Also, in the element *part*, 2.15% from the before-instruction and 0.7% from the after-instruction participants believed the keyword `new` is an element of an array (ME6). Finally, 1.08% of the participants from the before-instruction and 1.4% from the after-instruction predicted had chosen elements that can be stored in an array mismatched with their type (ME7).

8.3.1.4 *State change*: Declaration

Among the before-instruction participants, 21.51% held the misconception that after the declaration, instead of storing null, the array reference variable is blank or doesn't have any default value (MD2). Interestingly, I observed a slight increase (27.8%) in the percentage of participants holding this misconception after classroom instruction. After instruction, some of the participants also believed the default values that are stored in an array reference variable after instantiation are also stored after array declaration (MD3). However, this misconception was not found in the before-instruction dataset. In contrast, 4.3% participants from the before-instruction data set believed a question mark (?) is the default value of an array reference variable when declared (MD4). Moreover, 4.3% of the participants believed an unlimited number of elements could be stored in the array after declaration (MD6). These two wrong assertions (MD4 and MD5) were not found in the after-instruction sample. Lastly, the percentage of participants believing that after the declaration, the number of elements that can be stored is unlimited (MD6) experienced an increase from before-instruction (2.15%) to after-instruction (5.6%).

8.3.1.5 *State change*: Instantiation

I have found only one misconception on the state change of *instantiation*. 2.23% of the participants thought a question mark (?) is stored as the elements of an array after *instantiation* (MIn2). A similar percentage (2.8%) of participants held this misconception after classroom instruction.

8.3.1.6 *State change*: Assigning Elements

I found one misconception related to *assigning elements*. 9.68% of the participants had a misconception that assigning another variable to an indexed location of an array (e.g., `numbers[1] = count;`) acts as a reference assignment, meaning they share the space, not just the value (MAE5). After instruction, the percentage of the participants holding this misconception decreased (4.2%).

8.3.1.7 *State change*: Array Assignment

Similar to the previous component, only one misconception was found in the component of reference *assignment*. When an array reference variable is assigned to another array reference variable (e.g., `salary = newSalary`), the reference gets copied over, not the values. 7.53% of the participants believed array *assignment* copies the values of the arrays before classroom instruction. Surprisingly, after classroom instruction, the percentage of the participants holding this misconception increased (10.4%)

8.3.1.8 Parts and State Changes

Figure 8.1 shows the frequency distribution of participants holding at least one misconception across all the *part* and *state change* components. Array declaration was the most common type of misconception among participants (both before instruction and after instruction), holding at least one misconception. After classroom instruction, the number of participants holding at least one misconception on *declaration* did not decrease but slightly increased (31.9% from 31.18%). From Figure 8.1, I can observe that the frequency of participants holding at least one misconception for the

parts components decreased in the after-instruction; for *name* and *index* decreased significantly ($p < 0.05$). However, for the *state changes* components, I observed a slight decrease (for *instantiation* and *assigning elements*) or increase (for *declaration* and *assignment*).

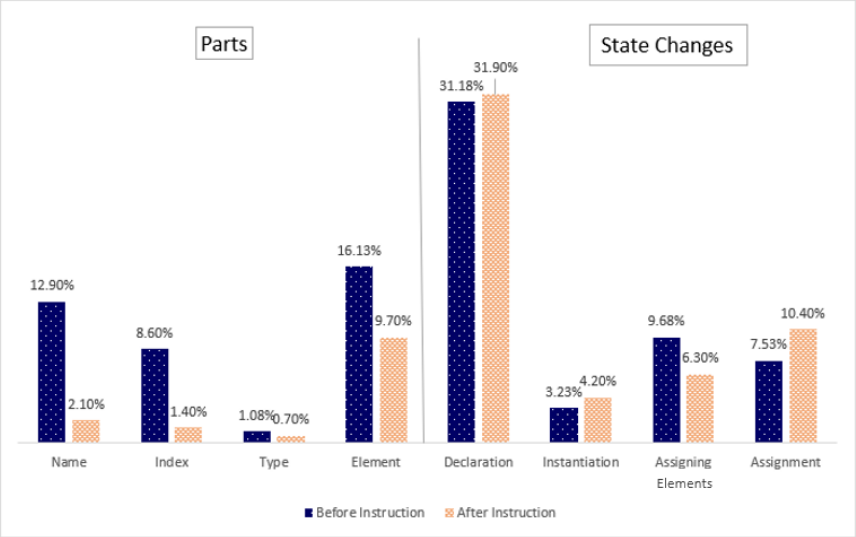


Figure 8.1: Frequency distribution of the participants holding at least one misconception across the *parts* and *state changes* components before and after classroom instruction.

As shown in Figure 8.2, before instruction 13 (14%) of the participants held misconceptions about only *parts* components. Moreover, 25 (26.9%) participants held misconceptions about only *state changes* components. Additionally, 15 (16.1%) participants held misconceptions for both *parts* and *state changes* components. Forty participants had no misconceptions (note they had mistakes but were not consistently selected to label them as misconceptions).

Figure 8.3 shows 9 (6.25%) of the participants held misconceptions about only *parts* components after classroom instruction. Moreover, 55 (38.19%) participants held misconceptions about only *state changes* components. Additionally, 9 (6.25%) participants held misconceptions for both *parts* and *state changes* components. Seventy-one

(49.31%) participants had no misconceptions.

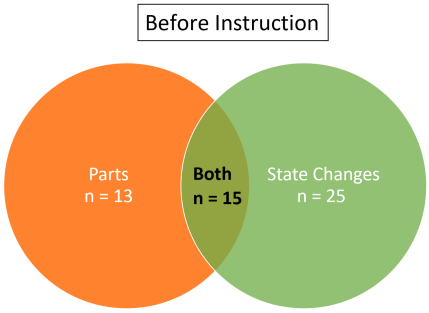


Figure 8.2: Venn diagram showing 13 participants (14%) had misconceptions on only *parts* components, 25 (26.9%) on only *state changes* components, and 15 (16.1%) on both the components. 40 (43%) participants did not have any misconceptions.

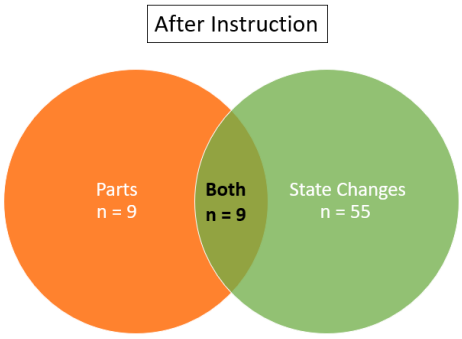


Figure 8.3: Venn diagram showing 9 participants (6.25%) had misconceptions on only *parts* components, 55 (38.19%) on only *state changes* components, and 9 (6.25%) on both the components. 71 (49.31%) participants did not have any misconceptions.

Over half of our participants held at least one misconception before classroom instruction (57%) and after (50.7%). Among the before-instruction participants, for *part* components, I found 65 participants (69.89%) with no misconception, 20 (21.51%) had one, 7 (7.53%) had two, and 1 (1.08%) with three misconceptions (see chart (a) in Figure 8.4). For *state changes* components, I found 35 (37.63%) with no misconceptions, 38 (40.86%) with one, 14 (15.05%) with two, and 6 (6.45%) with three misconceptions (see chart (b) in Figure 8.4).

Among the after-instruction participants, for *part* components, I found 126 participants (87.5%) with no misconception, 16 (11.1%) had one, 2 (1.4%) had two (see chart (c) in Figure 8.4). For *state changes* components, I found 80 (55.6%) with no misconceptions, 49 (34%) with one, 14 (9.7%) with two, and 1 (0.7%) with three misconceptions (see chart (d) in Figure 8.4).

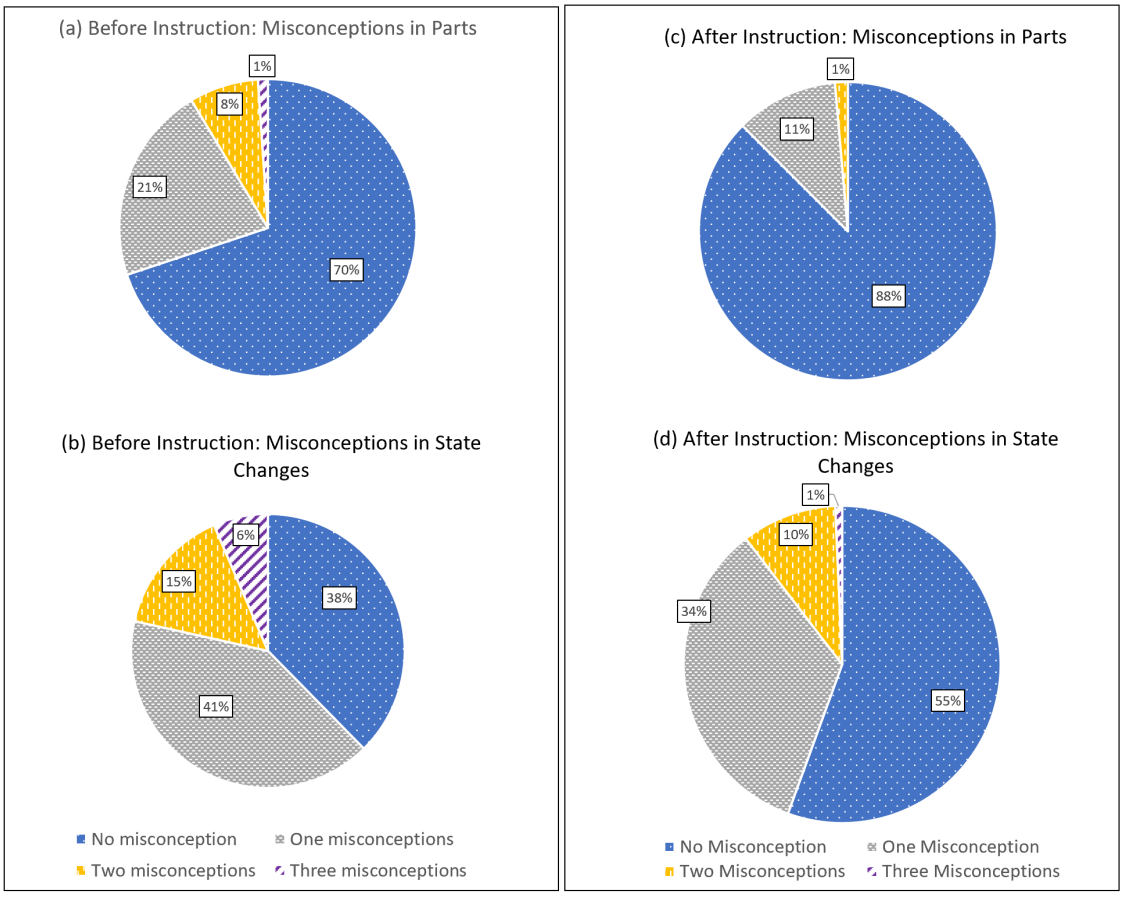


Figure 8.4: Frequency distribution of the participants across the number of misconceptions (zero, one, two, or three).

8.3.1.9 Prior Programming Experience

I analyzed if prior programming experience affects the misconceptions found in students' mental models. I found no significant correlation between participants' prior programming experience and their number of misconceptions. I also explored the correlation between each misconception listed in Table 8.1 and the participants' prior

programming experience. I found several weak correlations between a specific misconception and participants' prior programming experience from the before-instruction sample. However, no such correlation was found among the participants from the after-instruction sample. I found that the misconception *MN2: Students think type is the name* had a weak negative correlation with participants who had programming experience with programming languages other than Java (*phi* coefficient: -0.30, $p < 0.05$). I also found a positive weak correlation between *MT3: The array name is the type of the array* and participants who had experience with Java (*phi* coefficient: 0.27, $p < 0.05$), and *MA3: Assignment copies the values* and participants who had learned programming languages other than Java (*phi* coefficient: 0.21, $p < 0.05$).

8.3.2 Change in Misconception from Pre-test to Post-test

Below, I describe the participants' changes in misconceptions from the pre-test to the post-test based on each component of arrays.

8.3.2.1 *Parts*: Name, Type

For the component *name*, I found that in the pre-test, five participants held the misconception 'MN2: Array type is the name of the array' before instruction on arrays. After classroom instruction on arrays, five participants' misconceptions cleared up. However, one participant's misconception remained even after instruction. One participant did not have the misconception (MN2) in the pre-test, but this misconception appeared in the post-test. Figure 8.5 shows the changes in misconceptions for the component *name*.

For the component *type*, I could not find any misconception in any of the participants.

8.3.2.2 *Part*: Index

For the *part* component *index*, eight participants had the following misconceptions in the pre-test: '*MI3: There is no indexing into the array*' (2), '*MI4: Array index*

starts with 1' (3), and 'MI5: Array index ends with n (index of last element)' (2). After instruction, all of their misconceptions were absent (portrayed in Figure 8.5).

8.3.2.3 *Part*: Element

In total, nine participants held misconceptions regarding *elements* of an array in the pre-test. The misconceptions are: 'ME5: Element values and array names are related (e.g., books and "Harry Potter")' (8) and 'ME7: Type of values stored do not match the type of array' (1). These participants' misconceptions were eliminated after instruction. However, in the post-test, I identified the rise of the misconception ME5 in three participants and ME7 in one participant. I found another misconception in the post-test, which was not identified in the pre-test. One participant from the post-test believed '*ME6: Keyword 'new' is an element in the array*'. In total, I found five participants whose misconceptions arose after instruction, eight participants' misconceptions diminished, and one participant's misconception still remained after instruction (ME5) (illustrated in Figure 8.5).

8.3.2.4 *State change*: Declaration

I found twenty participants holding misconceptions on array *declaration* in the pre-test. Fifteen of them had a misconception that '*MD2: There is no default value for the elements of the array (blank/no value)*' after declaration. I observed the clarification of this misconception among 12 participants in the post-test. However, this misconception remained intact in three participants in the post-test. An additional nine participants were found to hold this misconception in the post-test. These nine participants did not have this misconception in the pre-test.

Similarly, nine participants were identified to have the misconception '*MD4: The default value for the array reference is stored as '?''* in the post-test, which did not exist in the pre-test. In the pre-test, two participants held the misconception of MD4. This misconception still remained in one participant in the post-test, and for

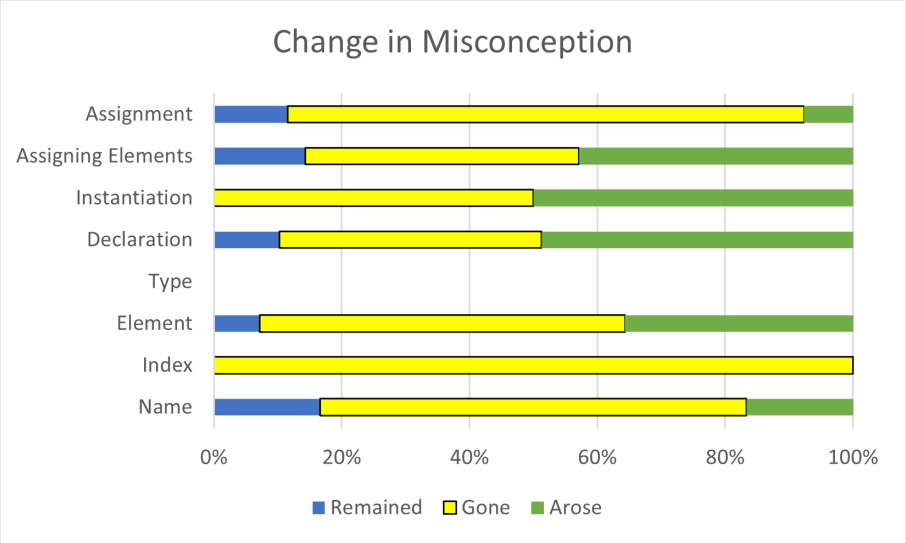


Figure 8.5: Participants' change in misconception from the pre-test to the post-test. The changes are labeled as remained (misconceptions present in the pre-test and remained in the post-test), gone (misconception present in the pre-test but diminished in the post-test), and arose (misconception was not present in the pre-test but arose in the post-test) (N = 66).

one participant, it was clarified.

In the pre-test, three participants held the misconception '*MD5: After declaration, memory is allocated for the elements*'. All of their misconceptions were clarified after instruction; thus, they were absent in the post-test. This misconception arose for one participant in the post-test.

In summary, I observed misconceptions rising among 19 new participants in the post-test about array *declaration*. Sixteen participants' misconceptions were clarified in the post-test, and four participants' misconceptions remained intact in the post-test. Figure 8.5 shows this frequency distribution.

8.3.2.5 *State change*: Instantiation

In the pre-test, two participants had a misconception that '*MIn2: After instantiation, '?' is stored as a default value*'. After classroom instruction, this misconception was clarified. However, this misconception arose among two new participants in the

post-test.

8.3.2.6 *State change*: Assigning Elements

In the pre-test, four participants had a misconception that '*MAE5: Primitive assignment is the same as reference assignment.*'. After classroom instruction, this misconception was clarified for three students. However, this misconception remained among one participant and arose among three new participants in the post-test.

8.3.2.7 *State change*: Assignment

Before classroom instruction twenty-four participants believed '*MA4: Array assignment transfers (cuts) values.*' After classroom instruction, this misconception was clarified among twenty-one participants and remained for three. After instruction, two new participants began to believe *'MA4: Array assignment transfers (cuts) values.'*

8.4 Discussion

8.4.1 Misconceptions Identified

Misconceptions are the incorrect assertions residing in humans' mental models that do not correspond to reality. Pea [170] termed it as a conceptual "superbug". Unlike actual bugs in computing, they are hidden inside a learner's mental model. According to mental model theories [40], misconceptions implanted in a mental model cannot be changed easily. Thus, identifying misconceptions is crucial for effective learning. In a study of numerous physics teachers, Sadler et al. [102] found that teachers who could identify students' misconceptions at the beginning of instruction were more effective in teaching than the teachers who could not.

I found several misconceptions in my data that were also commonly found in the literature and often obtained through more time-consuming qualitative methods (e.g., interviews). I am confident that our questionnaire (MMT-A) is a suitable proxy to be used as a probe to elicit students' misconceptions effectively.

A common misconception is that an array's *index* goes from index 1 to the number of elements inclusively [7,111]. In my study, I found evidence of this misconception in the form of the index starting at 1 (MI4), indexing ending at n (MI5), and, in some cases, both misconceptions at the same time (MI4 and MI5). These misconceptions were among the prior-instruction sample. I did not find misconceptions about MI4 and MI5 in the after-instruction sample.

Progmiscon.org [112] listed in their list of common misconceptions that students think elements are untyped. I found evidence of a similar misconception (ME7) in the before and after instruction samples. The data shows that students believed the array's name and its elements are semantically related (ME5), confirming a similar finding in the literature, where Kaczmarczyk et al. [7] found that students assumed a connection between a variable's name and the element value. Moreover, in both the physics and mathematics misconceptions literature, students' misconceptions appear when they treat abstract concepts as physical objects or they focus too much on the context of a problem [171].

Kaczmarczyk et al. also found a misconception in their qualitative study that "students think memory is allocated for Objects which have been declared, but not instantiated" [7, p.110]. I found evidence to support this finding. 4.3% participants from our prior-instruction sample believed this notion (Table 8.1 MD5). Kaczmarczyk et al. [7] found that students believed there is no default value for primitive variables in Java; I found a similar misconception but for reference variables (MD2). Surprisingly, when I compared each individual student's misconception change, I found that nine students started to believe that after declaration there was no default value stored in array reference variables (MD2) after classroom instruction. As opposed to storing `null` as a default value, many students (before instruction: 21.51%, after instruction: 27.8%) believed there is no default value for reference variables (MD2) or that reference variables store a question mark (?) as default value (before instruction: 4.3%)

(MD4). Some CS1 textbooks use a "?" as a default value in their diagrams [159]; thus, it is unsurprising that some students also have this misconception. Similarly, 2.23% participants from the prior-instruction sample believed question marks (?) are stored as elements of the array when it is instantiated. After classroom instruction, I found a misconception among students that the default value for the array reference variable when it is declared is the same default value when it is instantiated (e.g., int is 0, boolean is false). Perhaps they transferred the knowledge of instantiation to answer the questions regarding declaration.

When a primitive variable is assigned to an array's element, I found our participants believe it works the same as the reference assignment (MAE5). The same misconception was found in a qualitative study by Sorva [28].

Assigning an array to another is the same as a reference assignment. A prevalent misconception in reference assignments is that students think values get copied over instead of references similar to primitive assignments [17,28,152]. In my data, I found 7.53% participants holding this misconception (MA3) among the prior-instruction sample. Surprisingly, I saw an increase in the percentage in the after-instruction sample. This may indicate the instruction is not helping clarify this misconception among students.

8.4.2 Misconceptions: Parts and State Changes

This work identified CS1 students' mental model misconceptions of arrays.The most prevalent misconception was in the *state change* array declaration. Even after classroom instruction, I observed more students holding misconceptions on array *declaration*. From Figure 8.5, I can interpret that there is a visible decrease in the number of participants having misconceptions about the *parts* components. However, surprisingly, in the *state changes* components, in most cases (three out of four components), I noticed an increase among the students holding misconceptions after classroom instruction.

Prior studies have found both the novices and upper-level students struggle to understand the transitions of *state changes* [26, 80, 152]. Krishnamurthi et al. [30] describe that comparative studies between stateful and non-stateful programming concepts are understudied in computing education. He remarked, "state is a powerful tool that must be introduced with responsibilities" [30, p.385]. Psychologist Mayer [5, 160] showed through several studies that illustrations of a scientific concept's *parts* and *state changes* improved students' mental models.

8.5 Summary and Limitations

In Section 2.3.1, I have described how, in other domains, researchers elicited misconceptions with a questionnaire. In the CS domain, few researchers have followed a quantitative approach to uncover misconceptions (mentioned in Section 2.3.2). Among the few studies conducted in the domain of CS, most of the researchers used a qualitative approach to identify misconceptions. As CS classrooms are becoming larger, it seemed unfeasible to identify CS1 students' misconceptions and address them by using a qualitative approach. The questionnaire MMT-A developed based on mental model assertions and consistency can act as a diagnostic tool to identify misconceptions in a classroom. If included in an adaptive learning system, the identified misconceptions can be used to provide adaptive learning content to students.

My results are susceptible to a construction threat to validity, resulting from using multiple choice questions as some participants may guess or unintentionally choose an answer. In this case, a wrong answer may occur from a slip or misconception. As Lewis & Norman [172, p. 414] describe: "A person establishes an intention to act. If the intention is not appropriate, this is a mistake. If the action is not what was intended, this is a slip." I reduced this threat by including a wrong assertion multiple times in the questionnaire and identifying a participant's wrong assertion as a misconception only when they chose it 100% of the times it appeared.

To further explore this validity threat, next chapter, I describe a think-aloud study with MMT-A to observe whether the findings from the qualitative think-aloud data support the findings of the qualitative data mentioned in this chapter.

CHAPTER 9: EXPLORING VALIDITY AND RELIABILITY OF THE MENTAL MODEL TEST

9.1 Introduction

The previous chapters presented the results obtained from administering my instrument, the Mental Model Test of Arrays (MMT-A). Tony Albano, in his book on educational and psychological measurement, quoted, "A good test purpose articulates key information about the test, including what it measures (the construct), for whom (the intended population), and why (for what reason)" [173, sec.9.1.3]. The purpose of the MMT-A is to elicit mental models of novice programmers, not to utilize it as a validated psychometric test or a knowledge assessment instrument (e.g., concept inventory). Nevertheless, I performed several statistical analyses to explore how my collected responses fit into the validity claim by following Tony Albano's definition- "validity refers to the degree to which evidence and theory support the interpretations of test scores entailed by the proposed uses of a test" [173, sec.4.1]. By following a common procedure to provide a validity argument in the CS education domain, I performed a correlation with the scores obtained from the MMT-A and participants' CS1 course scores. Moreover, I conducted a statistical analysis with Rasch analysis (initial dataset) and Item Response Theory (IRT) (complete dataset) analysis.

I found a moderate positive correlation between scores obtained from the MMT-A and participants' CS1 exam scores (details are in Section 9.3.1). Rasch analysis with the initial dataset (Spring 2021-Pre-test Data-set) revealed an acceptable set of items (details are in 9.3.3). The measure of internal consistency reliability from Cronbach's alpha revealed good internal consistency both for the initial and complete dataset. However, factor analysis with a complete dataset revealed multiple factors

present in the participants' responses which violated the unidimensional assumption for conducting Item Response Theory. While the violation does not revoke the results rather it suggests that researchers should carefully consider the implications of this violation and explore potential reasons for it. In my dissertation's context, the violation of unidimensionality is due to the decomposition of the concept of arrays into sub-components which was necessary and applicable for the purpose of eliciting novice programmers' mental models based on the mental model theories. Below I describe the detailed results. Nonetheless, the graphical representations obtained from the Rasch Analysis (Wright Map) and Item Characteristic Curve (ICC) from IRT analysis further strengthen the claim that arrays state changes are harder for novice programmers.

9.2 Method

As discussed in Section 2.2.2, Allison Tew [83] claimed validity by providing two pieces of evidence: 1) content-related evidence and 2) construct-related evidence. I provide content-related evidence by analyzing common CS1 textbooks to identify each component of *parts* and *state changes* of arrays. Later, I developed questions or items for the Mental Model Test of Arrays (MMT-A) by utilizing the components of *parts* and *state changes*.

Tew [83] provided construct-related evidence with a three-pronged data analysis: 1) think-aloud interview data, 2) statistical analysis using the Item Response Theory of participants' responses, and 3) correlation with participants' scores from Tew's assessment instrument and their exam scores. I followed Tew's approach to explore construct-related evidence. Chapter 10 presents the data and results of a think-aloud interview data with MMT-A. In Section 9.3.1, I present the results of the correlation between scores obtained from MMT-A and participants' scores obtained from their CS1 course. After the first round of data collection, with the initial Spring 2021-Pre-test Data-set, I explored the quality of the questions of my instrument, MMT-

A. By conducting statistical analysis with Item Response Theory (IRT) [174], one can analyze the quality of a set of questions. IRT is a statistical framework used in psychometric and educational measurement to model the relationship between individuals' latent traits (unobservable characteristics, such as ability or proficiency) and their responses to test items. IRT provides a way to analyze and understand how individuals with different levels of the latent trait respond to different items in a test or questionnaire. Different IRT models exist, each with its own set of assumptions and parameterizations [175]. Common models include the Rasch model (one-parameter logistic model), the 2-parameter logistic model (2PL), and the 3-parameter logistic model (3PL). Unidimensionality is a key assumption in IRT models, implying that the measured construct is adequately represented by a single latent trait [176]. When unidimensionality is violated, there are multiple underlying factors influencing the observed responses. While a violation of unidimensionality does not necessarily revoke the results, it does raise concerns about the validity of the measurement [177]. It is suggested that researchers should carefully consider the implications of this violation and explore potential reasons for it.

With the initial smaller set of data collected during the first phase of my data collection (Spring 2021-Pre-test Data-set, N = 91), the participants' responses were analyzed using the Rasch modeling [177] with Winsteps software [178]. Rasch measurement constructs a unidimensional scale that tests how well the data fits the model [179]. The scale also helps us to determine item fit, item difficulty, and person reliability to design a high-quality test. Rasch measurement model is suitable for lower sample size requirements [173]. Rasch analysis provides an item difficulty map that allows us to understand which items are most and least difficult. It also provides a person reliability index to determine the replicability of person ordering [180]. In Section 9.3.3, I discuss the findings of performing Rasch analysis on my initial data set.

Later, I included all the datasets collected during Spring 2021, Spring 2023, and Summer 2023 to perform statistical analysis. After removing duplicate and blank data, I performed the statistical analysis on 282 responses. Participants' responses were coded as dichotomous data for each item, scoring 1 for correct answers and 0 for incorrect ones. Before performing IRT on the dataset, I wanted to investigate if the data met the unidimensional assumption. For this purpose, I administered exploratory factor analysis to ensure that there is more than one latent trait MMT-A is measuring. Section 9.3.4.1 presents the result of the factor analysis. Then, Section 9.3.4.2 presents the results of performing the 2-parameter logistic IRT model (2PL), which includes item-specific parameters for discrimination and difficulty.

9.3 Results

9.3.1 Correlation with Course Scores

I conducted a correlation analysis among participants' CS1 course scores and scores obtained from the MMT-A on two data sets: Spring 2021-Pre-test Data-set and Spring 2021-Post-test Data-set. For both the datasets, I found moderate positive correlations with scores obtained from MMT-A and CS1 course scores. Below, I present the detailed results.

9.3.1.1 Spring 2021-Pre-test Data-set

I used two scores from the participants' course: 1) the total score at the end of the semester (total CS1 score) and 2) the exam 3 scores. Exam 3 was conducted right after the topic of arrays was covered in class. I compared the relationship of their course scores with two scores from the MMT-A: 1) their correctness score (see Section 4.6) and 2) their total mental model score (see Section 4.8).

Correlation between MMT-A generated scores and Exam 3 score

I found a moderate positive correlation between the exam 3 scores and the mental model score, Pearson's $r = .475, N = 91, (p <= 0.001)$. Figure 9.2 shows the scatter-

plot of the results. Similar results were found when compared with the correctness score, Pearson's $r = .480, N = 91, (p <= 0.001)$. Figure 9.1 shows the scatterplot of the results.

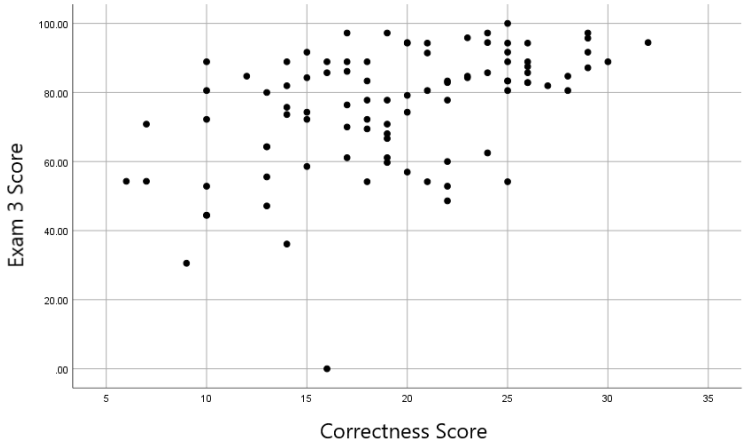


Figure 9.1: Scatterplot showing the correlation between participants' Exam 3 score and the correctness score $r = .475, N = 91, p <= 0.001$.

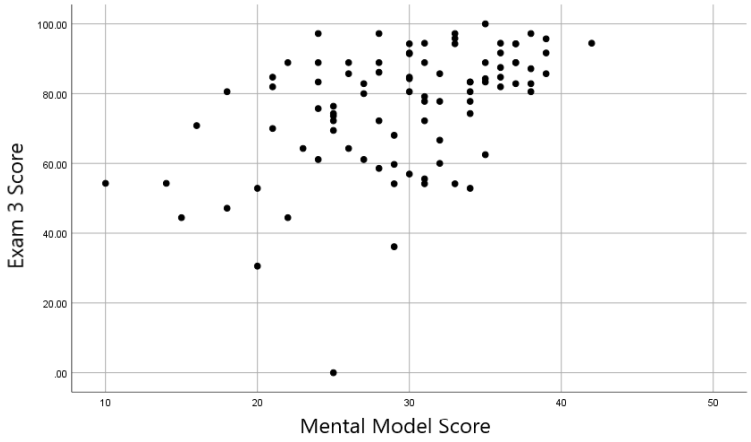


Figure 9.2: Scatterplot showing the correlation between participants' Exam 3 score and the mental model score ($r = .480, N = 91, p <= 0.001$).

Correlation between MMT-A generated scores and total CS1 score

A Pearson product-moment correlation coefficient was computed to examine the relationship between participants' total end-of-semester scores in their CS1 course

(total CS1 course) and the correctness score from our instrument. I found a moderate positive correlation between the two variables, Pearson's $r = .427, N = 91, (p <= 0.001)$. Figure 9.3 shows the scatterplot of the results.

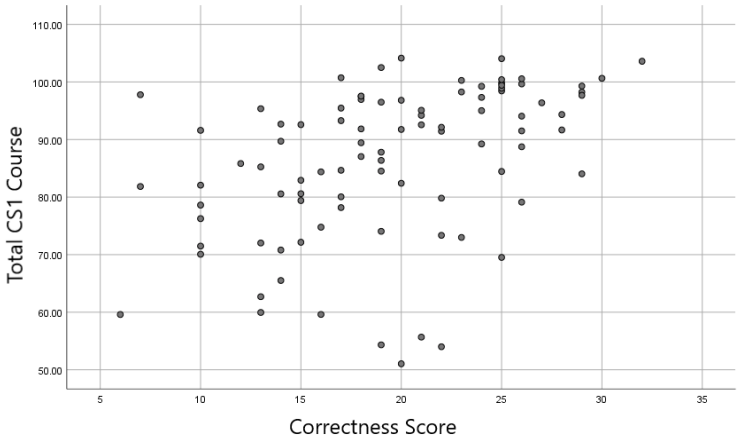


Figure 9.3: Scatterplot showing the correlation between participants' total CS1 score and the correctness score ($r = .427, N = 91, p <= 0.001$).

Similarly, I found a moderate positive correlation between our participants' total CS1 score and the mental model score, Pearson's $r = .481, N = 91, (p <= .001)$. The scatterplot is shown in Figure 9.4.

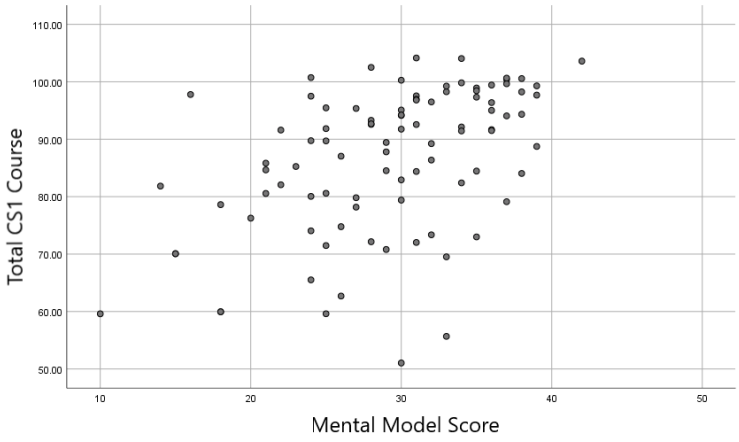


Figure 9.4: Scatterplot showing the correlation between participants' total CS1 score and the mental model score ($r = .481, N = 91, p <= .001$).

Overall, I found statistically moderate strong correlations between the measures

from our instruments and students' overall performance in the CS1 course.

9.3.2 Spring 2021-Post-test Data-set

Here, I present the results utilizing the Spring 2021-Post-test data set with N = 101. I performed a similar analysis, as mentioned in the previous Section.

Correlation between MMT-A generated scores and Exam 3 score

I found a moderate positive correlation between the exam 3 scores and the mental model score, Pearson's $r = .405, N = 101, (p < 0.001)$. Figure 9.6 shows the scatterplot of the results. Similar results were found when compared with the correctness score, Pearson's $r = .422, N = 101, (p < 0.001)$. Figure 9.5 shows the scatterplot of the results.

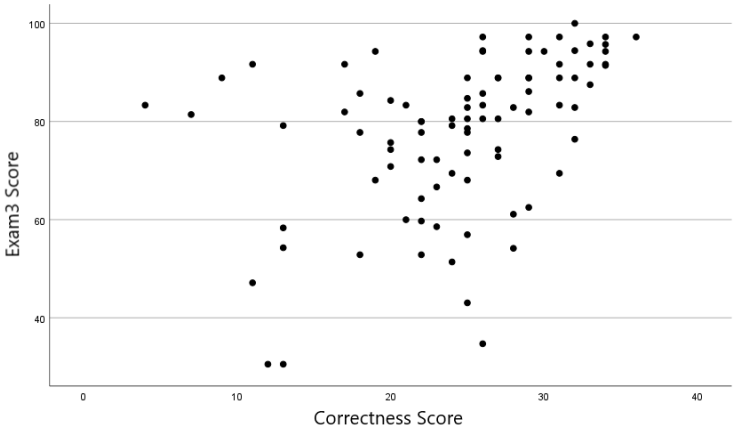


Figure 9.5: Scatterplot showing the correlation between participants' Exam 3 score and the correctness score $r = .422, N = 101, p < 0.001$.

Correlation between MMT-A generated scores and total CS1 score

A Pearson product-moment correlation coefficient was computed to examine the relationship between participants' total end-of-semester scores in their CS1 course (total CS1 course) and the correctness score from our instrument. I found a moderate positive correlation between the two variables, Pearson's $r = .468, N = 101, (p < 0.001)$. Figure 9.7 shows the scatterplot of the results.

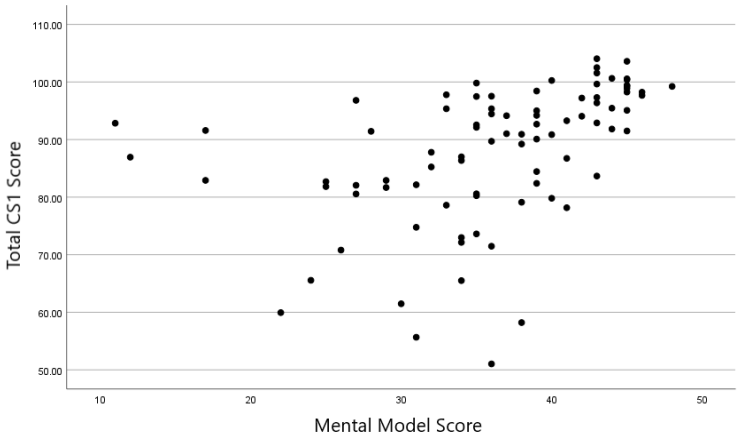


Figure 9.6: Scatterplot showing the correlation between participants' Exam 3 score and the mental model score ($r = .405, N = 101, p < 0.001$).

Similarly, I found a moderate positive correlation between our participants' total CS1 score and the mental model score, Pearson's $r = .446, N = 101, (p < .001)$. The scatterplot is shown in Figure 9.8.

I found statistically moderate positive correlations between the measures from our instruments and students' overall performance in the CS1 course in both data sets.

9.3.3 Rasch Analysis with Initial Dataset

9.3.3.1 Unidimensionality

I measured unidimensionality via mean-square (MNSQ) fit statistics and principal component analysis of Rasch residuals (PCAR). When I loaded all the items with dichotomous data in the analysis, the PCAR revealed that the Rasch dimension explained only 26.6% variance in the data with an eigenvalue of 13.03. A variance of greater than 50% explained by measures provides support for scale unidimensionality [181]. Furthermore, I found poor evidence for unidimensionality as the first contrast eigenvalue was 4.76 (9.7%), which is above the commonly acceptable threshold of 2-3 and higher than 5% [182].

Additionally, I measured MNSQ fit statistics for all the items. MNSQ fit values between 0.6 and 1.4 are considered reasonable with the measurement model [182].

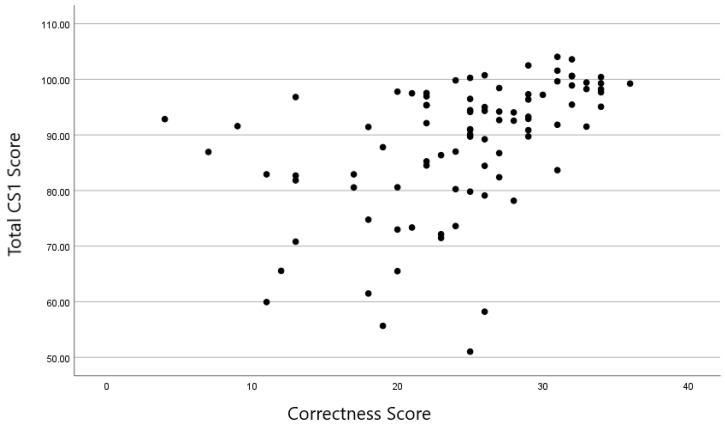


Figure 9.7: Scatterplot showing the correlation between participants' total CS1 score and the correctness score ($r = .468, N = 101, p < 0.001$).

As defined by Lambert [181], infit statistics indicate the fit of individual item response patterns to the measurement model, underlying construct, and the possibility of secondary dimensions. Outfit statistics are sensitive to outliers, in other words, responses that show great differences between person responses and item difficulties [181]. Underfit or Infit/Outfit MNSQ values exceeding a cut-off (e.g., >1.4) are usually considered a greater flaw than overfit (MNSQ < 0.6). Overfit was not used as exclusion criteria in this study [182]. Four items, including 2 items from *element* (E1, E3 listed in the MMT-A) and 2 items from *instantiation* (In1, In2 listed in the MMT-A), were identified as having infit or outfit mean statistics greater than 1.4 (underfitting items).

As I observed poor measure of unidimensionality and poor fitting for four items, I reran the Rasch analysis after removing the poorly fitting items (E1, E3, In1, In2). After removing the poorly fitting items, the dimensionality was reexamined with the remaining 32 items. There was still poor evidence to support unidimensionality as the unexplained variance in the first contrast was still above the threshold of 2–3 eigenvalues (3.5) and higher than the recommended 5% (7.4%) suggesting there was still a noticeable secondary dimension in the items.

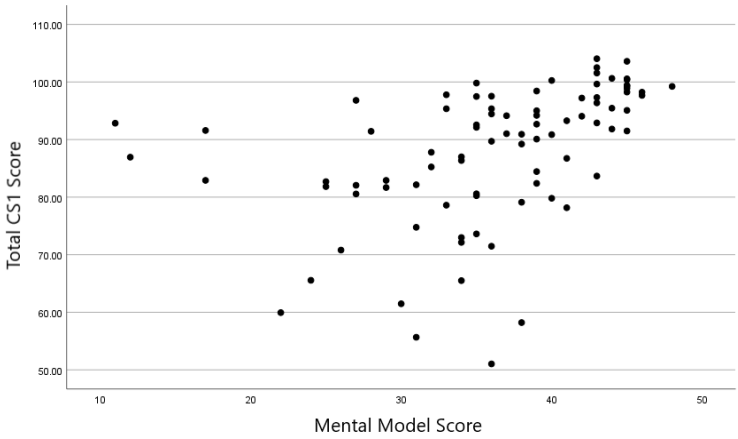


Figure 9.8: Scatterplot showing the correlation between participants' total CS1 score and the mental model score ($r = .446, N = 101, p < .001$).

As I organized the mental model of arrays into two components, *parts* and *state changes*, I wanted to further explore how the items in each set can construct a latent variable. The results revealed poor measure of unidimensionality and item fit. However, the statistical output in the Rasch modeling suggested that a subset of items worked better together. Seventeen items found fit to be in the subset (N1, I1, I2, I3, I4, I5, T1, T2, E4, E5, AE1, AE2, AE4, AE5, AE6, A2, A4). The PCAR revealed that the Rasch dimension explained 43.5% variance in the data with an eigenvalue of 13.07. The first contrast (the largest secondary dimension) had an eigenvalue of 2.95 and accounted for 9.8% of the unexplained variance. All fit statistics for all of the subset items were within acceptable limits except AE1. Item AE1 had an outfit statistic slightly outside the optimal range (1.56), suggesting the presence of outliers or unusual response patterns. The subset of items I received was at least acceptable, although not an ideal, scale.

9.3.3.2 Item Difficulty Measures

Rasch analysis also provides a graphical description of the knowledge of the test-takers and the difficulty of items onto the same item map, known as an Item map or Wright Map [183]. As Rasch model creates a scale, there is an expected distribu-

tion of item difficulty and person ability. These distributions should follow a normal curve. Figure 9.9 shows the wright map that I obtained from the analysis. It shows two vertical histograms where items (right histogram) are arranged from the easiest on the bottom and the most difficult on the top. The left vertical histogram shows the distribution of persons (denoted by #) arranged by their ability reassured by the correctness of the items (bottom: lowest ability, top: highest ability). Here, the middle vertical line that separates the two histograms represents the log-odd units (logit) scores. Here, M denotes the mean score, S denotes one standard deviation, and T denotes two standard deviations. Therefore, from Figure 9.9, I can say that the most difficult item was A1 from the *state changes* component *assignment* and the easiest item was from the *part* component *type* T2. Interestingly, 11 items from the *part* components were found to be easier items showing logit scores below the means, and only five items were from the *state changes*. All the items showing a sign of difficulty (logit scores above the mean) were from the *state changes* (A1_CORR, A6_CORR: *assignment*; IN3_corr, IN4_CORR: *instantiation*; D6_CORR, D3_CORR, D4_CORR, D1_CORR, D5_CORR: *declaration*; AE3_CORR, AE1_CORR, AE5_CORR: *assigning elements*). The histograms are shown in Figure 9.9.

9.3.3.3 Reliability

Reliability measures ensure the quality of a test. Oftentimes, the notion of accuracy is related to it [184]. Reliability refers to how consistent an instrument is in measuring a construct. I measured reliability using the following Rasch indices: the person separation index, item separation index, person reliability, and item reliability. A good test can give more precise locations of item difficulty and person ability [185]. Person separation and item separation indices generated by the Rasch measurement model help the investigator to determine whether there are enough items and persons spread along the continuum. A low person separation index (<2.0) means the test is unable to differentiate between a person's ability. Similarly, a low item separation

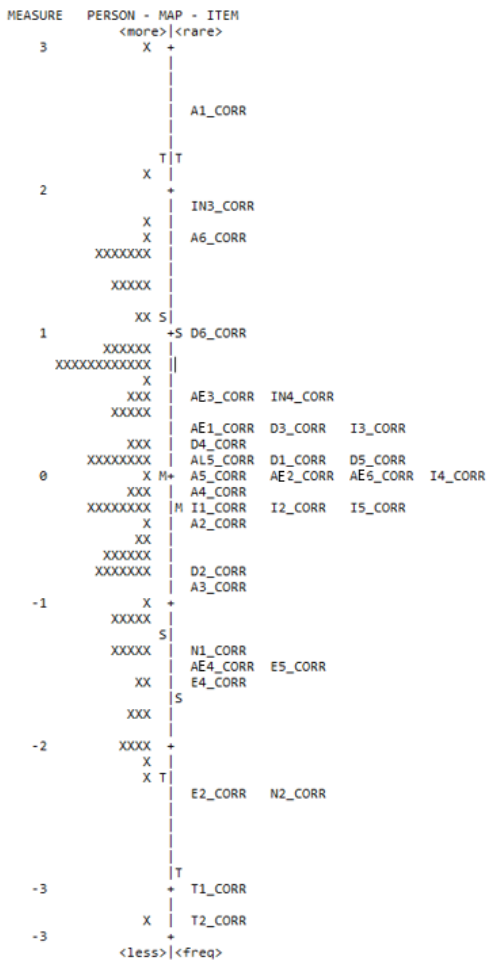


Figure 9.9: Item-person map (Wright map) portraying person ability distribution with the item difficulty distribution. The letters prefixed to the item numbers denote: N - *part: name*, T - *part: type*, I - *part: index*, E - *part: element*, D - *state change: declaration*, IN - *state change: instantiation*, AE - *state change: assigning elements*, A - state change: assignment.

index (<2.0) means items cannot be separated as easily or difficult. Additionally, the person reliability index ensures the replicability of the person ordering if the same person were given a similar set of items measuring the same construct [185]. Alongside that, the item reliability index ensures the replicability of an item's placement in terms of difficulty if the same item were given to a sample with the same size and attributes [185]. Item and person separation indices greater than 2.0 are considered

adequate [182]. Person and item reliability scores above 0.8 are considered a good indication. I also used Cronbach's alpha to measure internal consistency.

When I analyzed all 36 items using Rasch measurement, I found the sample-based person separation index to be 1.86 (< 2.00) and model-based 1.96 (< 2.00). Here, the sample-based separation index is a lower bound, and the model-based separation index is an upper bound to this value. In addition, I found the sample-based item separation index to be 4.44 (good separation: Item Separation Index ≥ 3.0) and the model-based item separation index to be 4.60 (good separation: Item Separation Index ≥ 3.0). The sample-based (.95) and model-based (.95) item reliability were high. However, the sample-based (.78) and model-based (.79) person reliability did not exceed 0.8.

When I reran the Rasch analysis with 32 items, I found higher sample-based (2.16) (moderate separation: $2.0 \leq$ Item Separation Index < 3.0) and model-based (2.27) person separation index (moderate separation: $2.0 \leq$ Person Separation Index < 3.0) as well as person reliability (sample-based: 0.82, model-based: .84). Similarly, I found higher sample-based (4.73) and model-based (4.90) item separation index and person reliability (sample-based: 0.96, model-based: .96). For these 32 items Cronbach's alpha value was .85.

I analyzed the higher correlated subset of 17 items, and I found the following results: sample-based (2.03) and model-based (2.16) person separation index and person reliability (sample-based: 0.80, model-based: .82). Similarly, I found the following for items: sample-based (4.30) and model-based (4.44) item separation index and item reliability (sample-based: 0.95, model-based: .95). For this subset the Cronbach's alpha value was .89, indicating high internal consistency reliability.

Though not all the items of our MMT-A were measured to be fit to create a unidimensional measurement scale with appropriate reliability, a subset of the items were found to have a good fit. In the end, with the initial set of data, the Rasch

analysis results suggested I needed more data to validate the complete instrument.

9.3.4 Analysis with Full Dataset

9.3.4.1 Exploratory Factor Analysis

An exploratory factor analysis was conducted using Principle Component Analysis (PCA) to assess the factor structure underlying the items in the MMT-A. PCA is a statistical method used for dimensionality reduction and data exploration. Varimax rotation technique was used in factor analysis to simplify the interpretation of factors by maximizing the variance of the squared loadings. Exploratory factor models were evaluated based on extracted communalities and factor loadings. Items that had extracted communalities greater than 0.30 within the same factor were considered sufficiently related to one another [186]. Similarly, items that loaded on one factor and had a factor loading greater than 0.30 were strongly correlated to the factor and remained in the final scale. Table 9.1 and 9.2 show the presence of multiple factors underlying the participants' responses. Therefore, I conclude that my dataset does not meet the assumption of unidimensionality. The reason behind this phenomenon is maybe because of the division of the concept of arrays into multiple *parts* and *state changes* components. Even though my data violates the unidimensionality assumption, I present the result of the IRT model analysis below.

Table 9.1: Factor Laodings and Item-Total Correlations for each item of the *parts* components.

Component	Items	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6	Factor 7	Factor 8	Factor 9	Factor 10	Factor 11
P: Name	N1	0.48										
	N2	0.41										
P: Index	I1	0.80										
	I2	0.79										
	I3	0.51										
	I4	0.73										
	I5	0.75										
P: Elements	E4	0.60										
	E5	0.57										
P: Type	T1	0.33										
	T2											
P: Elements	E1				0.92							
	E2				0.34							
	E3				0.94							

Table 9.2: Factor Laodings and Item-Total Correlations for each item for the *state changes* components.

Component	Items	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6	Factor 7	Factor 8	Factor 9	Factor 10	Factor 11
S: Assigning Elements	AE1	0.41						0.63				
	AE2	0.58						0.46				
	AE4	0.52										
S: Instantiation	In1				0.53							
S: Declaration	D1						0.81					
	D2						0.80					
	D3					0.88						
	D4					0.90						
	D5								0.89			
	D6											0.814
S: Instantiation	In2		0.81									
	In3		0.83									
	In4		0.78									
S: Assigning Elements	AE3		0.67									
	AE5			0.78								
	AE6			0.78								
S: Assignment	A2			0.54								
	A3			0.37								
	A4			0.54								
	A1									0.46		
	A6									0.45		
	A5										0.92	

9.3.4.2 Item Response Theory Analysis

In Item Response Theory (IRT), the 2PL (Two-Parameter Logistic) model is used to model the probability of a correct response to an item as a function of an individual's latent trait (ability) and two item parameters: the discrimination parameter (a) and the difficulty parameter (b). The 2PL model assumes that the probability of a correct response increases with the latent trait and that each item has a unique difficulty level.

Item Difficulty

The difficulty parameter (often denoted as b) is a parameter associated with each item in a test or assessment. The difficulty parameter represents the level of the latent trait at which there is a 50% probability of a correct response to the item. In simpler terms, it indicates the point on the latent trait continuum where an individual has an equal likelihood of succeeding or failing on the item.

If the difficulty parameter is positive, the item is located to the right of the latent trait distribution. This means that individuals with higher levels of the latent trait are required to have a reasonable chance of answering the item correctly. A higher positive b indicates a more difficult item.

If the difficulty parameter is negative, the item is located to the left of the latent trait distribution. This suggests that individuals with lower levels of the latent trait are more likely to answer the item correctly. A lower negative b indicates an easier item.

The magnitude of the difficulty parameter reflects the degree of difficulty or ease of an item. Larger positive values of b indicate greater difficulty, while larger negative values indicate greater ease.

Comparing the difficulty parameters of different items within the same test can provide insights into the relative difficulty levels of those items. Items with higher

positive b values are generally more challenging than items with lower positive b values.

Item Discrimination

The discrimination parameter (often denoted as a) is a parameter associated with each item in a test or assessment. The discrimination parameter measures how effectively an item differentiates between individuals with different levels of the latent trait being measured. For example, in a test of math ability, the discrimination parameter indicates how well an item distinguishes between individuals with high and low math abilities.

A positive discrimination parameter ($a > 0$) indicates that the item effectively discriminates between individuals with different levels of the latent trait. Higher values of (a) suggest stronger discrimination.

On the other hand, a negative discrimination parameter ($a < 0$) is a less common scenario and is theoretically problematic. A negative value for the discrimination parameter implies that individuals with higher levels of the latent trait are more likely to provide incorrect responses, while individuals with lower levels of the latent trait are more likely to provide correct responses. This goes against the fundamental assumption of IRT, where higher ability should be associated with a higher probability of a correct response.

Item Analysis

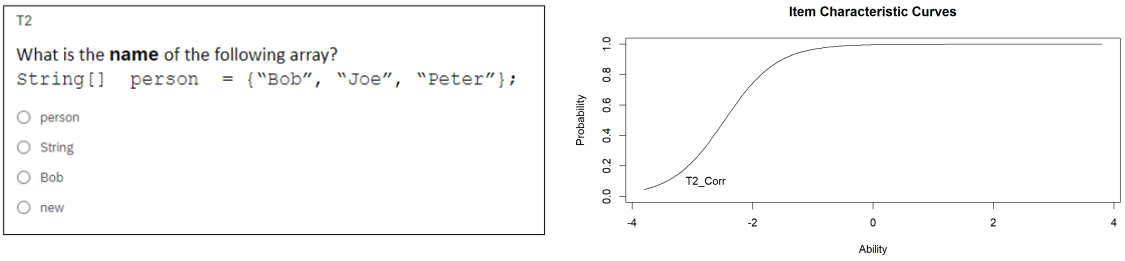
Table 9.3 shows the percentage of the correct response, point biserial correlation coefficient with total correctness score, and item difficulty (b) and discrimination (a) parameters obtained from the IRT for each item. Point biserial correlation was used to assess the relationship between the correct response of an item with the total correctness score. Items A5 and A6 seemed problematic due to insignificant correlation coefficients and low or negative discrimination parameters. Apart from

Table 9.3: Results of IRT analysis (N = 282) and Point Beserial Correlation for each item in the MMT-A. The third column (% correct) denotes the frequency of participants in the percentage who answered the item correctly. Point Biserial coefficients marked with an asterisk (*) denote statistically insignificant.

				IRT parameters	
Component	Items	% correct	Point Biserial	Difficulty (b)	Discimination (a)
P: Name	N1	87.6	0.49	-1.59	1.88
	N2	93.6	0.34	-2.08	1.85
P: Index	I1	83.3	0.59	-1.21	2.61
	I2	84.4	0.60	-1.15	3.73
	I3	70.2	0.59	-0.83	1.67
	I4	83.3	0.62	-1.13	3.45
	I5	85.1	0.61	-1.18	4.04
P: Element	E1	64.9	0.35	-1.05	0.68
	E2	87.1	0.43	-1.88	1.33
	E3	64.2	0.49	-0.81	0.94
	E4	80.9	0.66	-1.09	3.44
	E5	82.6	0.58	-1.16	2.96
P: Type	T1	96.1	0.36	-1.95	2.98
	T2	97.9	0.22	-2.47	2.30
S: Declaration	D1	46.8	0.33	0.07	0.63
	D2	61.3	0.34	-0.95	0.59
	D3	41.1	0.29	0.96	0.35
	D4	42.9	0.18	2.18	0.13
	D5	59.6	0.26	-0.77	0.56
	D6	36.3	0.40	0.76	0.79
S: Instantiation	In1	64.0	0.45	-0.64	1.07
	In2	36.0	0.34	0.72	0.86
	In3	34.9	0.47	0.58	1.27
	In4	48.2	0.42	0.03	0.93
S: Assigning Elements	AE1	67.0	0.59	-0.66	2.11
	AE2	72.3	0.65	-0.79	2.60
	AE3	49.6	0.36	-0.03	0.84
	AE4	79.8	0.60	-1.04	2.74
	AE5	69.8	0.53	-0.78	1.87
	AE6	65.9	0.65	-0.78	1.87
S: Assignment	A1	17.1	0.33	2.36	0.73
	A2	68.8	0.65	-0.67	2.79
	A3	69.1	0.52	-0.78	1.74
	A4	70.2	0.55	-0.79	1.97
	A5	**56.9**	**0.12***	**-2.84**	**0.09**
	A6	**23.1**	**0.11***	**-4.61**	**-0.27**

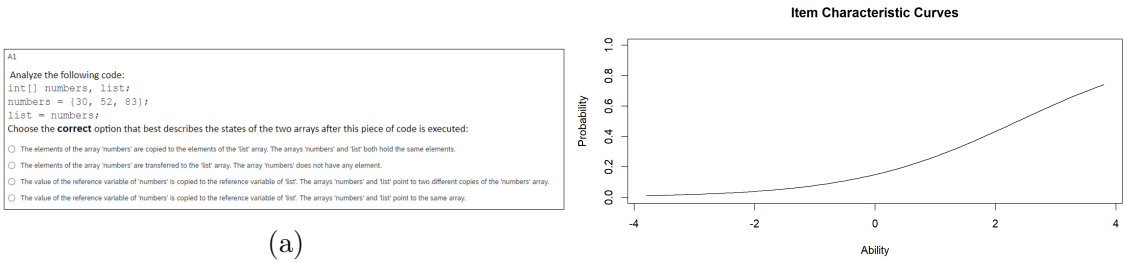
these two items, item difficulty ranged from -2.47 (item T2 from the *part* component *type*) to +2.36 (item A1 from the *state change* component *assignment*).

Figure 9.10 and 9.11 shows the Items T2 and A1 with their corresponding Item Characteristic Curve (ICC). The Item Characteristic Curve (ICC) is a graphical representation of the relationship between an examinee's level of the latent trait (ability or proficiency) and the probability of responding correctly to a particular item in Item Response Theory (IRT). The ICC is a fundamental concept in IRT, providing insights into how an item behaves across different levels of the latent trait. The typical ICC has a sigmoid (S-shaped) curve. The curve starts at a low probability of success for individuals with low ability, rises sharply through the region where most individuals are likely to answer correctly, and then levels off at a high probability for individuals with high ability. Appendix 11 includes the ICC curve for all the 36 items in the MMT-A.



(a) (b)

Figure 9.10: (a) Question T2 and (b) its Item Characteristic Curve.

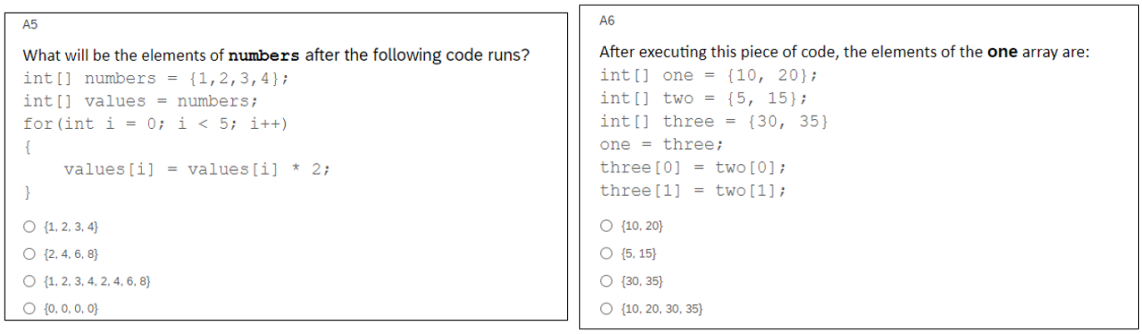


(a) (b)

Figure 9.11: (a) Question A1 and (b) its Item Characteristic Curve.

9.3.4.3 Reliability

A Cronbach evaluation indicated that the MMT-A had good internal consistency. Cronbach's alpha is a measure of internal consistency reliability. It assesses the extent to which items in a test are correlated with each other, providing an overall estimate of how well the items measure the same underlying construct. Cronbach's alpha of .90 and above indicates very high internal consistency. That means items in the test are highly correlated, suggesting a strong and reliable measure of the underlying construct. Cronbach's alpha in the range of 0.80 to 0.89 indicates good internal consistency. This means the test is considered reliable for most research and practical purposes. With N = 282, for all 36 items, the Cronbach's alpha value was 0.876, indicating good internal consistency.



(a) (b)

Figure 9.12: (a) Question A5 and (b) Question A6 used in the Mental Model Test appeared to be problematic in the item analysis.

9.4 Discussion

The statistical analysis showed that my data did not meet the assumption of unidimensionality for Item Response Theory (IRT) both in preliminary analysis (Section 9.3.3) and final analysis (Section 9.3.4). This is due to the items loading up to multiple constructs. This result is expected, as to elicit novice programmers' mental models, I decomposed the concept array into *part* and *state changes* components with four subcomponents each. For assessing the validity of a test, IRT is the common

statistical analysis approach in the CS education domain [83, 85, 87]. Therefore, I proceeded with this approach. To completely validate the MMT-A as a standardized test in the future, Multidimensional Item Response Theory (MIRT) [187] may seem appropriate. Due to the purpose of my instrument being to elicit mental models, not a standardized test, I did not proceed to conduct further statistical analysis with MIRT.

However, the statistically significant correlation with scores obtained from MMT-A and participants' CS1 course score (mentioned in Section 9.3.1) provides credibility to my instrument. In addition, the Rasch and IRT statistical analysis provided evidence to support further the claim that students found the questions of *state changes* difficult. The Wright map (Figure 9.9) from the initial dataset (N = 91) shows the most difficult question is A1 from the *state changes* component- *assignment* and the easiest question being T2 from the *part* component *type*. Moreover, all the questions with difficulty measured above 0 are the questions on arrays *state changes* (see Figure 9.9). The IRT analysis with the full dataset (N = 282) also refers to the same. By considering items A5 and A6 problematic and excluding them, we can see the items that have item difficulty above zero are D1, D3, D4, In1, In3, In4, and A1, which are questions on arrays *state changes*. The ICC curves (see Appndix 11) provide additional insight into it. The ICC curves of the *part* component *name* started rising in between -4 to -2 logit ability demonstrating that lower-ability students have a higher probability of answering correctly. A similar phenomenon was noticed for the items related to the *part* components *type* and *elements*. For the *part* component *index* the curve started to rise in between -2 to 0 logit ability indicating the items of *index* are harder than the items of the rest of *parts* components. However the ICC curves for the *state changes* components *declaration* and *instantiation* showed a steep upward curve rather than a sigmoid shaped curve indicating that as the ability increases so as the probability of getting the items correct. The average ability test-takes (0 logit) have

around or below the probability of 60% to answer the items correctly. On the other hand, for the *parts* components, the probability for an average ability test-taker was higher. The ICC curves again strengthen the finding that the array's *state changes* are harder.

The items A5 and A6 (see Figure 9.12) were found problematic based on the IRT analysis. Question A5 is the only question that students had to answer by utilizing the knowledge of the loop. In question A6, students had to consider three arrays, which can be challenging. Moreover, question A6 was the last question participants had to answer. Lesser knowledge of loop and cognitive load to handle three arrays and cognitive tiredness may have impacted the response to questions A5 and A6. Future administration of MMT-A excluding these two items can provide additional insight into the findings.

9.5 Conclusion

The purpose of my instrument, the Mental Model Test (MMT-A), is not to be a standardized test. Nonetheless, I explored the validity and reliability of MMT-A to assess the quality and consistency of my items. In this Chapter, I presented the findings by utilizing different statistical measures to provide a conclusion. The statistically significant correlation between scores obtained from MMT-A and CS1 course score suggests the construct I am hoping to measure with my test correlated well with what other tests are measuring. Even though my dataset did not meet the unidimensional assumption of measuring a single latent trait, the results from Rasch and Item Response Theory further strengthen the claim that array's *state changes* are harder for novice programmers than the *parts*. The statistical analysis to measure the reliability of the test provided good measures for MMT-A. In the next Chapter, I further provide evidence for the results found from MMT-A responses by conducting a think-aloud study with a subset of questions from the MMT-A.

CHAPTER 10: EXPLORING MENTAL MODELS WITH THINK-ALOUD

10.1 Introduction

In the previous chapters, I presented a quantitative approach to elicit novice programmers' mental models and their misconceptions with the questionnaire MMT-A. I conducted a qualitative study (semi-structured interview with think-aloud protocol) with a sub-set questionnaire of MMT-A. I began with three motivations- 1) to understand novice programmers' mental models of arrays *parts* and *state changes* components with MMT-A in a qualitative way, 2) to learn if the findings of qualitative data analysis support the findings of the quantitative data analysis, and 3) to understand if the participants' interpretation to the choices of the MMT-A is in alignment with the mental model assertion mapped from the MMT-A's choices. I presented eleven questions to ten programming students from the College of Computing and Informatics at UNC Charlotte. The key finding was participants had more inaccuracy and confusion regarding the array's *state changes* than the *parts* components. This finding also emerged in the quantitative data among different participants mentioned in the previous chapters (Chapters 5 and 6). In addition, I found incorrect assertions in the qualitative data, which were also present in the quantitative data (mentioned in Chapter 8). However, qualitative exploration revealed additional assertions from participants' mental models, which I did not consider in my list of assertions. This suggests revising the list of assertions of novice programmers' mental models of arrays.

10.2 Methodology

10.2.1 Participants

I recruited novice programmers from undergraduate students at the University of North Carolina at Charlotte. Students could participate in this study if they had recently completed or were currently enrolled in ITSC: 1212, ITSC: 1213, or ITSC: 2214 courses. Participants were recruited during the 1st, 2nd, 3rd, and 4th weeks of the Fall 2023 semester from the College of Computing and Informatics (CCI). I recruited participants by including flyers placed across the CCI building, announcements sent to the programming courses, and classroom visits to CCI programming courses. Participants received $25 for participating in the study. The study was approved by the Institutional Review Board (IRBIS-21-0067). I conducted the study with 10 participants. Table 10.1 shows details of participants' programming education, previous programming experience, and Java learning sources.

10.2.2 Task

I asked participants to answer eleven multiple-choice questions from the MMT-A using the think-aloud protocol. Participants were shown one question at a time on a screen. In the MMT-A, assertions were placed in multiple questions to measure consistency. For this qualitative study, I selected a subset of questions from MMT-A from the array's each *parts* and *state changes* components. I chose the questions that could provide the context to elicit the set of correct assertions from the participants. For example, the *part* component *name* has only one correct assertion- MN1 (listed in Table 5.1). Therefore, I included only one question (see Figure 10.1) for the *name* where MN1 is present. I conducted semi-structured interviews to elicit the details of participants' mental model assertions. Figures 10.1, 10.2, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 10.9, 10.10, and 10.11 show the eleven questions used in this study. Participants were asked to read each question out loud and then answer while thinking out loud. I

asked further related questions to understand the participants' mental models.

10.2.3 Interview Protocol

Before the interview, students' consent to audio and video record the semi-structured interview with the think-aloud protocol was taken. Students were notified that no prior preparation was needed for the interview. The interviews were conducted using the video conference platform Zoom. I presented each question to the participant by sharing my screen. The interview lasted for 25-40 minutes. The participants were asked to determine the correct answer to each question by thinking out loud. Based on the participants' responses, I asked more questions to understand the rationale behind their responses.

10.2.4 Analysis

The interviews were audio recorded and transcribed. I recorded each participant's answers to each question. I recorded their explanation of their chosen answers. I used In Vivo Coding [188] to preserve participants' tone, voice, and support for their mental model assertions. In In Vivo coding, findings are presented from participants' own language and terminology rather than alternative methods where codes are researcher-derived. Participants' inconsistent answers were discussed with another research personnel to remove bias from a single coder.

10.3 Results

Table 10.2 summarizes the correctness of all the participants' responses to the eleven questions. Below I describe each participants' responses to the questions in details.

Table 10.1: Details of each participant's (n = 10) programming exposure, previous programming experience, and source where they learned Java.

Participants	Prior Programming Experience	Java learning Source	CCI passed computing courses with semesters
P1	None	UNCC course, Youtube	ITSC: 1212: Spring 2022, ITSC: 1213: Fall 2022, ITSC: 2214: Spring 2023
P2	Java, C++	UNCC course, Youtube, Online tutorials	ITSC 1212: Fall 2021, ITSC: 1213: Spring 2022, ITSC: 2214: Fall 2022
P3	C, C++	UNCC course	ITSC: 1212: Fall 2021, ITSC: 1213: Spring 2022, ITSC: 2214: Fall 2022
P4	None	Textbook	ITSC: 1212 equivalent at a community college: Summer 2023
P5	None	UNCC course, Online tutorials	ITSC: 1212: Fall 2021, ITSC: 1213: Spring 2022
P6	AP CS Principles	AP CS course materials	AP CS Principles in High school.
P7	Markup language (HTML, CSS), JavaScript	UNCC Course	ITSC: 1212: Spring 2023
P8	Block based programming in high school, Java at previous University.	UNCC course, Textbooks, Online tutorials	ITSC: 1212: Credit by exam
P9	AP CS Principles (Python) in high school	UNCC Course	ITSC: 1212: Spring 2023
P10	No	UNCC Course	ITSC: 1212: Summer 2023

Table 10.2: Partcipants' response to each question (questions are on leftmost column; P: *parts* components, S: *state changes* components). Participants' correct response is marked with a check mark (✓), incorrect response with a cross (X), and confusion with (C). The highest course completion for P1, P2, and P3 is ITSC: 1212, ITSC: 1213, and ITSC: 2214 (yellow columns). The highest course completion for P5 is ITSC: 1212 and ITSC: 1213 (green column). The highest course completion for P4 and P6-P10 is ITSC: 1212 (purple columns).

Components	Questions	P1	P2	P3	P5	P4	P6	P7	P8	P9	P10
P: Name and Type	Q1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P: Index	Q2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P: Element	Q3	✓	✓	✓	✓	✓	✓	C	✓	✓	✓
S: Declaration	Q4	✓	✓	✓	✓	X	✓	X	X	X	X
S: Declaration	Q5	✓	✓	✓	✓	✓	✓	X	✓	C	C
S: Instantiation	Q6	✓	✓	✓	X	C	C	C	C	C	X
S: Instantiation	Q7	X	X	X	X	✓	X	X	X	X	X
S: Assigning Elements	Q8	✓	✓	X	✓	✓	✓	✓	✓	✓	✓
S: Assignment	Q9	✓	✓	✓	✓	✓	✓	✓	✓	✓	X
S: Assignment	Q10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S: Assignment	Q11	X	X	✓	X	X	X	X	X	X	X
Score (out of 11)		9	9	9	8	8	8	5	7	6	5

10.3.1 *Parts* Components

10.3.1.1 Name & Type

Figure 10.1 (correct answer marked with X) shows one of the questions (*Name & Type: Q1*) from the MMT-A presented to all the participants. Everyone responded to the question with the correct answer. P1 also identified the name of the array to be `scores` in the next question (Index Q2: Figure 10.2). P5 answered the question with more details. P5, while identifying the name of the array, remarked "*It's behind the parentheses, where you declare the name after the parentheses are where you define what the array or object is going to be. char is the type.*" P7 identifies `letters` as the name as it appears on the left of the assignment operator (=). However, P7 seemed unsure about what `char` means, saying, *"I think like if I were to code it, I know what you mean, but I don't know how to explain what* `char` *specifically means in this situation."* The rest of the nine participants identified `char` as the data type.

They identified the array's name by seeing the word that is next to the array's data type.

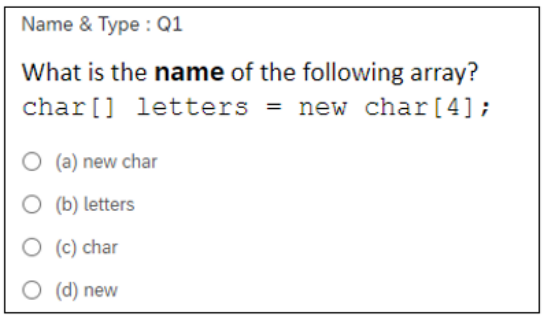


Figure 10.1: Think-aloud question probe for *part: name*. The correct answer is option (b).

10.3.1.2 Index

Figure 10.2 shows one of the questions (*Index: Q2*) from the MMT-A presented to all the participants. Everyone responded to the question with the correct answer. All the participants responded confidently that array indexing starts with zero. They mentioned the indices of the `scores` array are 0, 1, 2.

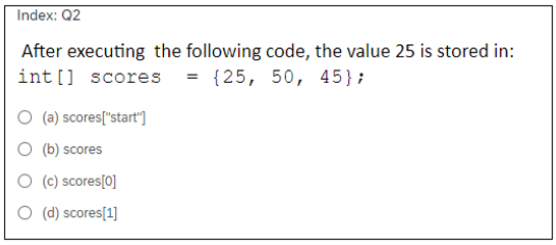


Figure 10.2: Think-aloud question probe for *part: index*. The correct answer is option (c).

10.3.1.3 Elements

Figure 10.3 shows one of the questions (*Elements: Q3*) from the MMT-A presented to all the participants. Eight out of the ten participants confidently responded with the correct answer. They said because `books` is an integer array, it will only store integer numbers. They did not choose the other option because they thought they

were either decimals, characters, or Strings. P7 was uncertain between the correct option and option (c). P7 thought because the array's name was `books`, it could store the String element 'Harry Potter'. P7 remarked, "*I feel like because the name is books, I thought of Harry Potter first, but because it says* `int` *what my guess would be 10.*" P7 was "*51%*" confident for the correct answer and "*49%*" confident for option (c): 'Harry Potter'. P10, while responding to the *Elements: E1* question at first,

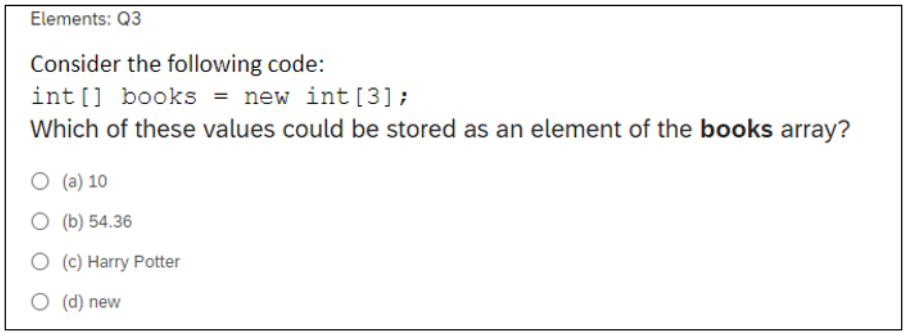


Figure 10.3: Think-aloud question probe for *part: elements*. The correct answer is option (a).

expressed confusion on what the term *element* means. P10 remarked, "*I actually don't know what an element is.*" Then P10 realized an element might be what it is called a value in the array. P10 then answered the question with 10 (option (a)) explaining 10 to be the only integer here that could be technically stored in the `int` array.

10.3.2 *State changes* Components

10.3.2.1 Declaration

Two questions from the MMT-A were presented to our participants on the *state change* component *declaration* (see Figure 10.4). In Java, when an array is declared, an array reference variable is created, and by default, it stores *null* in it. *null* is a special keyword in Java stored in the reference variables to show that the reference variable refers to nothing.

Declaration: Q4

Five (P1, P2, P3, P5, and P6) among the ten participants answered the first question (Declaration: Q4 see Figure 10.4) correctly. These five participants used the process of elimination to conclude with the right answer. P1 discarded option (b) as the code in the question `char[] productCode` did not mention any capacity (size). P2 discarded option (b) as he believes `productCode` array cannot store an unlimited number of elements as it is an array, not an ArrayList. P1 was also found to have this view. P3, P4, and P5 said `productCode` cannot store unlimited elements because it does not have a size right now. In this context, referring to array instantiation, P6 remarked *"I think you need another line of code that says like, how, how many spaces the array has."* In addition, P7 labeled this question as "*tircky*". This participant was confused between the correct option and option (b). However, this participant was "*65%*" confident that option (b) was the correct answer. However, this participant was quite confident that there was no limit to how many things can be stored in an array.

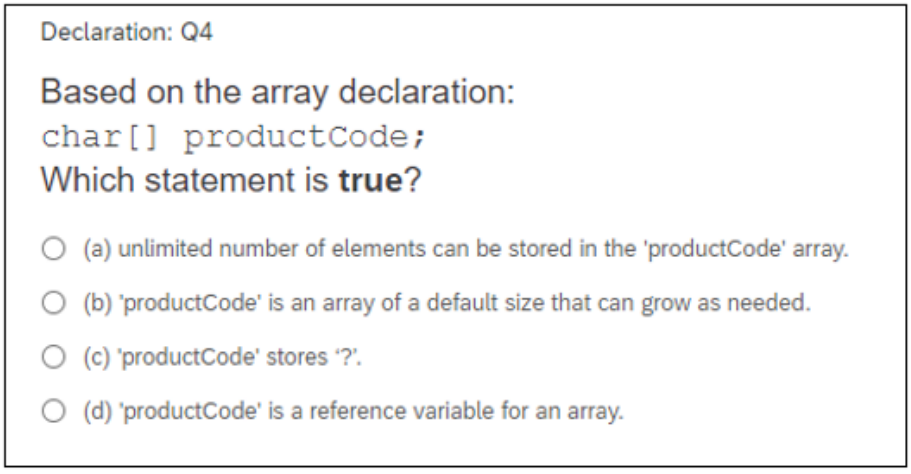


Figure 10.4: Think-aloud question probes for *state change: declaration*. The correct answer is option (d).

P1 and P2 discarded option (c) as a correct choice, mentioning `productCode` is an array, not an ArrayList. These five participants discarded option (d) because there

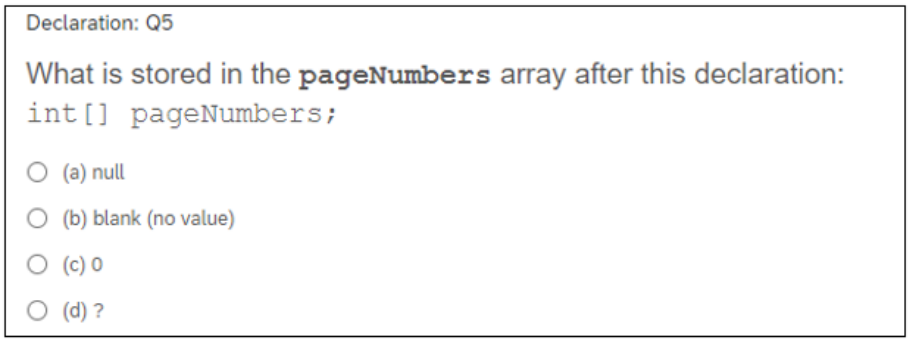


Figure 10.5: Think-aloud question probes for *state change: declaration*. The correct answer is option (a).

was no line of code that assigned question mark (?) to the `productCode` array. P2 believed no elements can be stored in the `productCode` array right now because it is not instantiated. P3 seemed confused about the concept of default values in Java. P3 mentioned that an array does not auto-store anything when it is declared.

These five participants chose option (a), which says `productCode` is a reference variable for an array to be true. In the follow-up, I asked them what is a reference variable. P1 mentioned the line of code `char[] productCode;` is not actually creating an array with space (memory). This participant thinks we can refer to the array with the reference variable `productCode`, but we cannot actually store elements in it. P2 had difficulty explaining what is an array reference variable. P2 remarked, "*It's stuff that makes sense in my head, but I've never had to put it to words. When you refer to an object, it's typically you are using its name. So, like, the reference variable is, like a pointer to the object.*" P3 also mentioned the same difficulty in explaining what an array reference variable is. According to this participant, an array "*reference variable stores the array itself that it's referencing.*" P5 mentions an array reference variable as a pointer. This participant explained that by executing the line of array declaration, `productCode` is not pointing anywhere. P6 stated the difference between a reference variable and a primitive variable while defining it. P6 stated that a primitive variable is like a value. Because an array is an object, an array reference variable stores a

location "*like it's pointing to a specific place in the memory.*" Despite providing a clear explanation of a reference variable, P6 felt "*70%*" confident with the answer. Moreover, four participants (P4, P7, P9, P10) said they do not remember what a reference variable is. Therefore, they did not consider option (a) as a correct answer. On this note, P4 remarked, "*I'm not good with my jargon. So I just don't really remember, what a reference variable is.*"

The next popular choice was option (c). Three participants (P4, P9, P10) chose the option (c) to be true. All of them said that there is no mention of a specific size in the declaration; right now, it is an empty array (default size is zero), but it can be grown as needed by specifying its size later by instantiation. On this remark, P4 used an analogy of an empty house with no rooms saying, "*It'd (`productCode` array) be like an empty house with no rooms. And then time we declare a `productCode` with a new character. We would add it to the to the thing and it would create a new room.*"

All of the participants believed option (d) could not be a choice as a question mark (?) was not assigned to the `productCode` array.

Declaration: Q5

After declaration, `null` is stored in an array reference variable in Java. The question **Q5** (see Figure 10.5) aimed to elicit our participants' knowledge in this regard.

Seven (P1-P6, P8) out of the ten participants responded with the correct answer (option (a): `null`). Among them, three participants (P3, P4, and P8) used the process of elimination, and four participants (P1, P2, P5, P6) directly concluded the answer. Although P1 answered correctly when explaining the answer, a potential misconception came out. This participant remarked, "*if you tried to reference this, like, `pageNumber[1]`, you would get `null`.*" According to Java semantics, `pageNumber[1]` does not exist right now as the array reference variable is only declared in this line of declaration. P2 answered correctly but used the wrong terminology. P2 said the answer is `null` because "*it hasn't been initialized yet.*" The terminology here should

be instantiated, not initialized. P5 answered this question with `null`; however, was a bit confused with option (b) blank/no value. P9 and P10 were also confused between `null` and blank/no value. P9 was "*50%*" sure, and P10 was "*60%*" for the correct answer to be blank/no value (option (b)). P9 tried to differentiate between `null` and blank/no value as "*Blank is just, you know, blank and there's nothing in it. But null, it's like, like it's holding null.*" P10 said, *"I just feel blank because there are no values yet. But that can change. And I feel like if it was null, means it's void, so it wouldn't technically be void. So I would say blank, I think".* P10 thinks `null` is void and blank means empty. Although this participant interpreted `null` and blank no value in a similar way, this participant thinks they are different. P10 explained null with a mathematical term '∅' saying *"it can't return a value or it can't hold a value may be one of the two."*

P2 discarded option (b) by saying 'blank' is not a computer term. P2 further explained to consider 'blank' as an empty quotation mark (' '), but P2 would term that as `null`. P6 further mentions never seeing anything return blank. Therefore, P6 also does not consider 'blank' to be a computer term.

All participants discarded options (c) and (d), explaining because a zero or question mark (?) was not assigned to the `pageNumbers` array, it can not store them. However, P4 showed a little bit of support for option (c): 0. P4 thought that as `pageNumbers` array reference variable is an integer type, it may auto-store zero. I put zero as a viable option because some students might think the same default value that is stored in the elements of arrays after instantiation (i.e., 0 in `int` array, 0.0 in `float` or `double` array) are stored in an array reference variable when it is declared.

As a follow-up question, I asked all the participants what is `null`. Table 10.3 summarizes the key findings of the responses. Most of the participants (n = 6) responded `null` as nothing or no value. P1 in this context remarked, *"Null, I think to me, it's saying like, there's nothing like there's no value of any type, like, you're not*

Table 10.3: Themes Emerging from the question 'what is `null`?'

What is `null`?	Partcipants
1. `null` is kind of an error	P1, P7.
2. `null` is absence of anything, lack of content, void, or empty set.	P2, P3, P5, P10.
3. `null` is nothing or no value.	P1, P3, P5, P8, P9, P10.
4. `null` is a standard form or programmer term of nothing	P3, P8.
5. `null` means the variable is not pointing to anything.	P2, P6.

getting an integer, you're getting nothing." P3 described `null` as *"it's kind of a telling you as the programmer, hey there's nothing in here. You haven't assigned anything."* P5 views `null` as nothing, which later can be filled with stuff.

Some of the participants (n = 4) defined `null` as the absence of anything, void, or an empty set. On this note, P2 remarked, *"null is the absence of anything, it's not zero, it's not. It's just the absence of having any sort of data that it's pointing to."*

P3 and P8 mentioned `null` as the programming standard term of nothing. P8 remarked, *"`null` is the computer science-like coding version of nothing."* While defining `null`, P2 and P6 explicitly said a variable stores `null` when it is not pointing to anything. P6 quoted, *"if it's a reference variable, it means it's not actually pointing to any data. It's just empty."* Though uncertain P1 and P7 hinted `null` to be an error. P7 remarked, *"I don't remember exactly what `null` is, it's, but it's not an error. I know that there's a difference between a `null` and an error, but it's like, kind of an error. `Null` is a kind of error."*

10.3.2.2 Instantiation

When an array is instantiated, memory is allocated for the array to store elements with the size n, and the reference pointing to the array gets assigned to the array reference variable. Appropriate default values (i.e., 0 in `int` array) are stored in each of the elements of the array in Java. Question Q6 gauged participants' knowledge of the memory allocation of the array after instantiation, and Q7 assessed participants' knowledge about the default value stored in array elements after instantiation (questions presented in Figure 10.6 and 10.7).

Instantiation: Q6

Only three participants (P1, P2, P3) answered this question correctly (option (a) in Figure 10.6). It is worth mentioning that these three participants had passed the three core programming courses in our college: ITSC 1212: Introduction to Computer Science I, ITSC 1213: Introduction to Computer Science II, and ITSC 2214: Data Structures and Algorithm. However, P1 and P3 used the process of elimination to arrive at this conclusion. P2 concluded with the correct answer, saying, "*it's (`goals` array) going to be a length of four. So, however much memory that takes up, it would be allocated.*" P1 defined instantiation as the creation of a new array. P1 reiterated that the array's size is four with indices 0, 1, 2, and 3. While explaining what is happening after an array instantiation P3 provided a clear difference between array declaration and instantiation. P3 remarked, *"Okay, then I'm going to make a new one. variable goals, this is going to point to this... So whenever I call on goals, it's going to say that, Oh, you mean this one over here! The ones in the past (pointing to the declaration question) would have been okay, I'm going to make a new integer array; I'm going to call it goals. So I said, Okay, goals. When I call goals, it's going to be like, Okay, what are you talking about? What? Where's it? "*

Four participants seemed unsure about the accurate answer to the question Q6. P4 stated to be equally unsure about the correct option (a) and incorrect option (c). This participant claimed to be out of practice with arrays. However, at the moment of the interview, P4 was currently enrolled in the ITSC: 1213 course, which begins with the review of arrays. P4 thought the array's size was five (option (c)) and thought the indices were 0, 1, 2, 3, and 4. The same confusion was held by P9. P9 also stated that the indices of the `goals` array are 0, 1, 2, 3, and 4. It is worth mentioning that similar to P4, P9 was enrolled in the ITSC: 1213 course at the moment of the interview. P10 chose option (c) as the correct answer without any confusion. P10 also stated the indices be 0, 1, 2, 3, and 4.

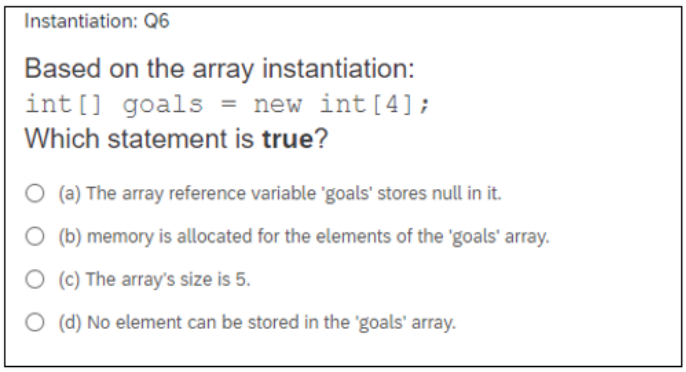


Figure 10.6: Think-aloud question probes for *state change: instantiation*. The correct answer is option (b).

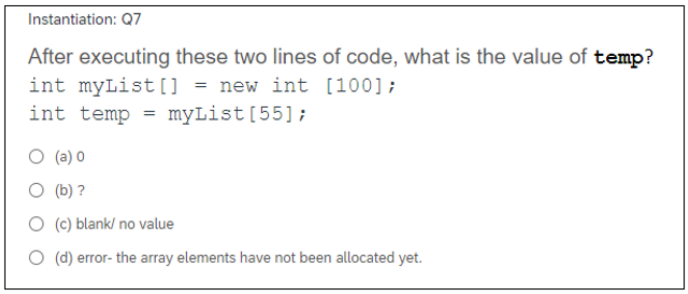


Figure 10.7: Think-aloud question probes for *state change: instantiation*. The correct answer is option (a).

Participants P6 and P8 were confused between the correct option (a) and incorrect option (b). However, P6 believes `goals` array has a certain amount of memory. P8 noted option (b) as the correct option as this participant thought each element of the `goals` array stores `null` as a default value after instantiation. Both P6 and P8 believe the size of `goals` array is four, not five.

P10 seemed confused about the correct answer for question Q6. This participant has a different view about this line of instantiation. P10 is certain that the array's size is not five (option (c)). According to this participant, the line of code in the question `int[] goals = new int[4];` means we are assigning the fifth element of the array to the `goals` variable. On this remark, P10 said, "*I think this (the line of code in the question) is calling on the fifth element, so there can potentially be more*" (referring to the size of the `goals` array). P10 mentioned being only enrolled in theITSC: 1212

course in Spring 2023.

Instantiation: Q7

Figure 10.7 shows the question Instantiation: Q7. This question aims to elicit participants' mental models of default values stored in each element of an array when it is instantiated. Only one participant (P4) answered this question correctly. P4 is well aware that the default value (P4 called it '*base value*')(which is 0 in this case) got assigned to the `temp` array *"because that's typically what Java does."* However, this participant did not seem 100% certain about the answer. Most of the participants (n = 9) either chose option (c) or (d).

P1 thought there was a syntax error in the code because the square bracket ([]) is placed after the name of the array reference variable, not before. This participant is not aware of this alternate way to declare an array. However, P1 did not choose option (d) because the error mentioned in option (d) did not match what this participant believed. When prompted to consider this a correct way to declare an array, this participant chose option (c) as the correct answer. P1 believes that because we did not assign any value to the elements, it cannot store 0; therefore, the elements should be blank. P2 had the same belief as P1. P6, P7, and P10 seemed unsure between the incorrect options (c) and (d). P10 mentioned that options (c) and (d) are essentially the same. P10 thought as the elements have not been allocated yet, each element of the `myList` array has no value. P6 thought the execution of the code would give an error, but the error would not be given because the array elements have not been allocated yet (option (d)). P6 believes after instantiation, memory is allocated, and this participant expressed this belief in the previous question (Q6). P6 believes an error might occur because there is nothing in index 55 of the array because it was never set to any value. P6 is "*50%*" confident between options (c) and (d). While thinking about the answer, P6 remarked, "*I never learned how the memory works for arrays.*"

Similar to P6, P3, P5, and P8 also believe in error but think the reason for the error stated in option (d) is wrong. P3 explains the answer as *"you made a new integer that can hold 100 different elements. But you haven't put anything in any of those 100 different elements. It's a bunch of boxes with nothing in them."* P3 believes accessing the elements of the `myList` array by assigning the element of 55 in `temp` will cause `nullPointerException`.

Similar to P1, P5 thought the alternate way to declare an array by placing the square brackets after the name of the array (`myList []`) is an error. P5 explains the error as "*I don't believe Java allows that. I don't believe Java allows square brackets to be in the name of an object.*" On the same note, P8 believes the program will generate an error: element not found.

Most of the participants did not consider option (a) (n = 9) and option (b) (n = 10) as a correct choice because they think an element can only store a zero or a question mark if it is assigned to it.

10.3.2.3 Assigning Elements

Figure 10.8 shows the question from the MMT-A presented to the participants on the *state change* component *assigning elements*. All of the participants except P3 answered this question correctly. These nine participants stated even though the value of `velocity[2]` has changed in the last line, it did not affect the value of `velocity[1]`. The older value, which `velocity[2]` stored at the moment of the assignment operation (=), gets stored in `velocity[1]`. In this context, P4 mentioned coding as a linear process. All the participants immediately discarded option (c) by saying an array element cannot store two values at the same time. P3 answered this question with option (d). P3 thinks assignment operation with the elements of an array works similarly to a reference assignment. This participant mentioned `velocity[1]` pointing to `velocity[2]`.

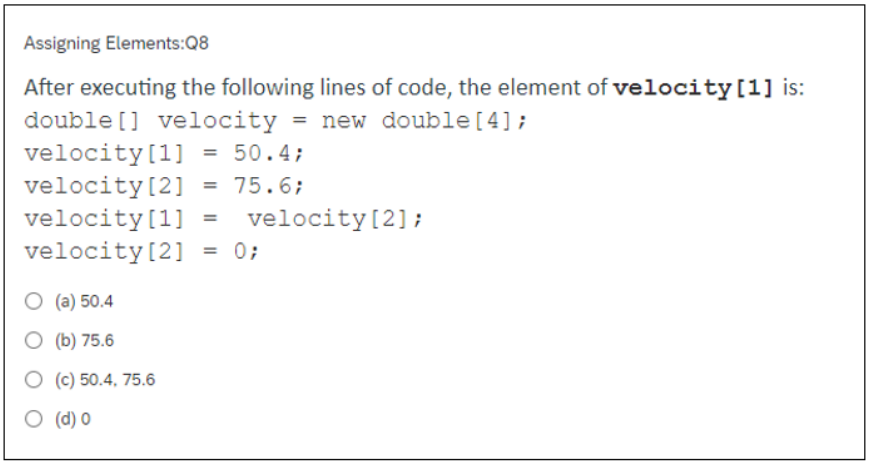


Figure 10.8: Think-aloud question probes for *state change: assigning elements*. The correct answer is option (b).

10.3.2.4 Assignment

Figures 10.9, 10.10, and 10.11 show the three assignment questions from the MMT-A presented to the participants. When an array is assigned to another array, two things happen: 1) the array reference gets assigned from right to left, and 2) the reference of the right side does not change. With these two questions, I aim to unfold participants' mental model of array assignment.

Assignment: Q9

All participants except P10 answered this question correctly. All nine participants used the same explanation to conclude with their answer as they did in the *assigning literal* question (Q8). P5 and P7 explicitly mentioned the values of the *largeNumbers* array will get copied over to `smallNumbers`. P1 was unsure whether the assignment operation changes the values or makes it point to the `largeNumbers` array. Whereas, P2 and P3 mentioned after the assignment operation, the `smallNumbers` array is pointing to the `largeNumbers` array.

P10 is the only participant who chose the incorrect choice: option (c). According to this participant, the `smallNumbers` and `largeNumbers` variables used in the as-

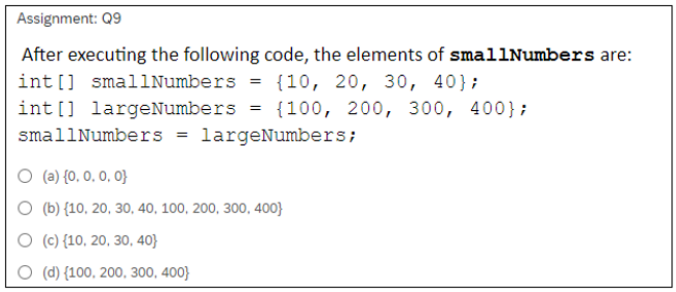


Figure 10.9: Think-aloud question probes for *state change: assignment*. The correct answer is option (d).

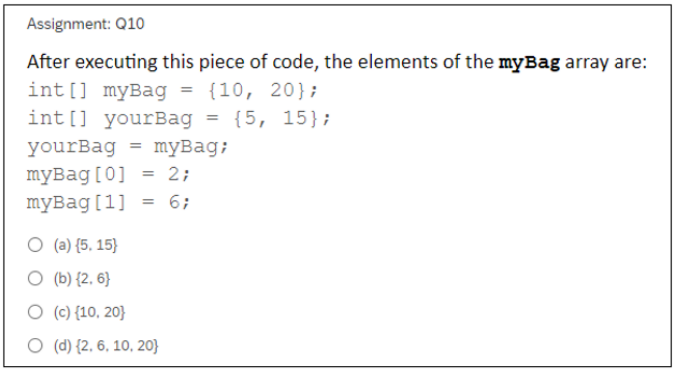


Figure 10.10: Think-aloud question probes for *state change: assignment*. The correct answer is option (b).

signment operation are not the arrays created in the first two lines. These are two different primitive variables. Therefore, the `smallNumbers` array will be unchanged. This participant explains *"it's (`smallNumbers` array) still gonna be 10, 20, 30, 40. Because on the third line and code, there's no brackets (i.e., []), so I don't think it would recognize that it's, it's small numbers array, that array equals large numbers array, because I think it's just gonna, like it could be any variable. So I don't think it would know if it's like the array."*

Assignment: Q10, Q11

All the participants used the same mental model assertion that the assignment copies values from right to left to answer the question Q9, which they used to answer the previous questions. However, when asked about the values of *yourBag* array (question Q11), everyone except P3 answered incorrectly. Most participants (n =

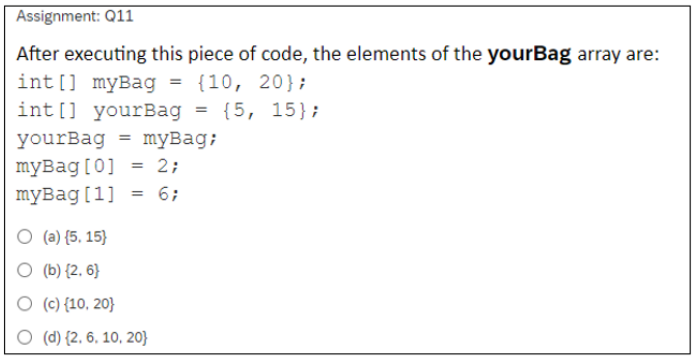


Figure 10.11: Think-aloud question probes for *state change: assignment*. The correct answer is option (b).

8) believed the array assignment is only copying values; therefore, the changes of the elements of `myBag` in the last two lines of code should not effect the `yourBag` array. However, P1 was unsure and was expressing this uncertainty repeatedly. P1 later concluded by saying, "*I'm thinking one option is now* ***`yourBag`*** *is like pointing at* ***`myBag`****, if that makes sense. And the other one is the values are literally being reassigned. I think I think it's more likely that it's reassigning.*" P3 held the correct mental model assertion of array assignment. However, P3 also believed primitive assignment works as reference assignment (mentioned in Section 10.3.2.3).

On the other hand, P10 again applied the mental model assertion used in Q9 to answer Q11. P10 believes that because there are no square brackets in the third line of code (`yourBag = myBag`), `yourBag` array will hold the initial values of 5, 15.

10.3.3 Incorrect Assertions

Throughout the think-aloud semistructured interview, I found participants stating assertions that are incorrect based on the semantics of Java. I present these assertions as incorrect assertions, which have the potential to be possible misconceptions. Table 10.4 lists the incorrect assertion found in the data. Three participants (P4, P9, P10) mentioned during the interview that an array's index ends with n (array's size). In Java, array indexing begins with 0 and ends with n-1. All three participants described the indices of the `goals` array from question Q6 (see Figure 10.6) to be

0, 1, 2, 3, 4. Meanwhile, the accurate indices are 0, 1, 2, and 3. It is worth noting that these three participants had the experience of completing only the ITSC: 1212 course. Moreover, in the context of the question, Q6, P1, and P3 mentioned arrays can hold size+1 elements. P7 appeared believing the array's name and the values it can hold are semantically related. This participant thought the String "Harry Potter" could be an element of the `books` array even though `books` array is an integer array.

Three participants (P4, P9, P10) appeared to believe after the declaration, the default size of an array is 0, and later on, during instantiation, it can grow as needed. These three participants chose option (c) as the correct choice in the question Declaration: Q4 (see Figure 10.4). However, in the MMT-A, I mapped the choice (c) with the following assertion '*MD5: After declaration, memory is allocated for the elements'*. The participants did not seem to imply that option (c) meant MD5. These participants appeared to believe as in the declaration, there is no mention of the array's size; the default size of the array is zero, which can be changed when an array is instantiated. On the other hand, P7 appeared to believe that because there is no specific mention of the array's size in the declaration, there will be no limit on the stored elements in the array. P7 mentioned while choosing option (b) in question Q4 (see Figure 10.4) as the correct choice, *"in every array, I'm pretty sure we were taught that there's no limit of how many things can be stored"*. I mapped option (d) of the questions Q4 and Q5 to the following assertion: '*MD4: The default value for the array reference is stored as '?'*. However, in the interview process, I realized one participant (P8) considered a question mark (?) as the default value of the array reference variable after declaration only when the array reference variable is of character (`char`) type. P8 answered with a question mark in question Q4 when the given array (`productCode`) was character typed. However, P8 did not choose a question mark (?) in the following question Q5 when the given array (`pageNumbers`) was integer (`int`) typed. P8 discarded option (d) in the question Q5 by saying, "*So first off question*

Table 10.4: Incorrect assertions found in the think-aloud semi-structured interview data with participants (n = 10) separated by the highest course completion.

Misconceptions or Incorrect Assertions	Highest Course Completion		
	ITSC: 1212	ITSC: 1213	ITSC: 2214
1. MI5: Indexing ends with n (array's size).	P4, P9, P10	-	-
2. ME5: Elements and array names are semantically related (e.g., books and "Harry Potter").	P7	-	-
3. After declaration, the default size of an array is 0 which can be grown as needed later.	P4, P9, P10	-	-
4. MD6: After declaration, the number of elements that can be stored is unlimited.	P7	-	-
5. After declaration, question mark (?) is stored in the char typed \\array reference variable as a default value.	P8	-	-
6. MD2. After declaration, there is no default value for the elements of the array (blank/no value).	P6, P7, P10	-	P1, P2
7. After instantiation, null is stored in each of the indices as the default elements.	P8	P5	-
8. MAE5. Primitive assignment is the same as reference assignment.	-	-	P3
9. MA3. Array assignment copies the values.	P4, P6, P7, P8, P9, P10	P5	P1, P2

mark is not in there because question mark is a character". Half of the participants (see Table 10.4) appeared to believe that after the declaration, the array reference variable stores no value/blank.

Two participants (P5, P8) mentioned they believe after instantiation null is stored in each of the elements as a default value. P5 while answering question Q6 mentioned, *"Currently it would store four* `null` *values.* `Null` *in index 1, 2, 3 and zero. And then memory is allocated for the elements of* `goals` *array. I'm gonna say no, not for the elements because it's* `null`*. So I don't think it would allocate memory to it.* `goals` *store the whole group of these* `null` *objects."*

P3 appeared to believe the assignment operation between two array elements works the same as the reference assignment. P3 was the only one who answered correctly about the array assignment (question Assignment: Q11). The rest of the nine participants appeared to believe that the array assignment only copies the values, not references.

10.3.4 Miscellaneous Findings

The previous section mentioned wrong assertions or misconceptions found in the data. During the interview, I observed some findings that I had not anticipated. Below, I summarize the findings:

"Java does not auto-store"- No knowledge of default values: I found many participants in many contexts consistently claiming that Java does not have a concept of default values. In the context of question Declaration: Q4 (see Figure 10.4), nine participants rejected question mark (?) being the default value of an array reference variable when declared. Their reason was since we did not assign a question mark (?) to the *productCode* array, it cannot store a question mark. A similar explanation was found for discarding a zero as a default value stored in an array reference variable when declared. After instantiation, zero is stored in the elements of an integer array. In the context of question Q7, nine participants believed zero could not be stored in the elements because we had not assigned zero to each element. They consistently mentioned Java cannot have a default value unless assigned.

"I'm not good with jargon."- Lack of knowledge of programming terminologies: Four participants (P4, P7, P9, P10) stated they did not remember what the programming terminology meant. These four participants stated they don't remember what a reference variable is. In addition, P7 does not remember what `null` is. Also, P10 could not remember what an element was. It is worth noting that all these four participants had passed only the first introductory computing course,ITSC: 1212. Moreover, I noticed inconsistent use of terminology. For example, P4 repeatedly termed the process of instantiation as the declaration.

"There are no brackets"- Syntactical confusion: Throughout the interview session, some participants appeared to have syntactical confusion. P1 and P5 recognized an alternate way of declaring array (`int myList[]`) to be an error as they had not seen this way of declaring an array. On the other note, P10 appeared to believe

you need to include a square bracket([ ]) if you want to assign one array to another. For example, in the context of the question Assignment: Q9, according to P10, the correct way of an array assignment is `smallNumbers[] = largeNumbers[];`. These two variables will not be considered arrays without the square brackets.

10.4 Discussion

Based on the above results, I discuss several key findings and their connection to the prior work mentioned in this dissertation.

10.4.1 Parts vs. State changes

Throughout all the responses collected from the MMT-A, one finding emerged repeatedly: students struggle more with array's *state changes* components than the *parts* components. This qualitative analysis strengthens this finding. Almost all the students accurately answered the questions related to the *parts* components (see Figure 10.2). At the same time, inaccuracy and confusion were seen in the answers to the questions related to the *state changes* components. Moreover, while generating answers, I observed students quickly narrowing down an answer for the *parts* components. On the other hand, for answering the questions of *state changes*, students mostly used the process of elimination without full confidence. I also found more incorrect assertions about the components of *state changes*. Incorrect assertions were found among the students who passed all three core computing courses in CCI related to array's *state change*: *declaration* and *instantiation*.

State changes are dynamic and invisible although crucial to learn [25, 26]. State changes are difficult to understand as the mechanism is hidden from the perceptual view [19, 20]. The textual representation of a program has little or no connection to the state change it provokes [25, 27]. Mapping syntax properties to concrete state changes create obstacles not only for novices but also for advanced programmers [25].

10.4.2 Incorrect Assertions

I found the evidence of six incorrect mental model assertions in the qualitative data included in the MMT-A. These six assertions were also found in the quantitative data (mentioned in Chapter 8). The incorrect assertions regarding the end index of an array (MI5 in Table 10.4) is a well-known misconception of arrays [7]. The quantitative and quantitative responses provided evidence for this misconception present in our students. Interestingly, this misconception (MI5) was found to be present in the before-instruction (ITSC: 1212) data set, not the after-instruction (see Table 8.1). The participants holding this wrong assertion in the think-aloud interview had completed ITSC: 1212.

The next wrong assertion in Table 10.4, ME5, was found in the quantitative data both within before and after instruction participants. I included this misconception as an option of MMT-A by reviewing Kaczmarczyk et al.'s work [7]. In their semi-structured think-aloud study, they found participants assuming a semantic connection between values and the variable names regardless of the variable type. They found their participant remarking, *"And so because there's two arrays, cheese and meats, uh, all those turkey and ham and roast beef are gonna be sorted into the meats array"* [7, p.109]. I also found a participant assuming the String "Harry Potter" as an element of the integer `books` array because "Harry Potter" is the name of a book.

I found several misconceptions regarding array's *state change*: *declaration*. In the quantitative analysis (mentioned in Chapters 5, 6 and 8), I found the component *declaration* to be the most challenging to the students. The component *declaration* deals with underlying memory usage, which is hidden from the programmer. Kaczmarczyk et al. [7] also found many misconceptions regarding underlying memory usage in their work. The participants of my study stated that they did not properly understand the concept of memory usage in Java. On this context P6 mentioned, *"I never learned how the memory works for arrays."* P2 mentioned lack of practice in dealing with

underlying memory usage by saying, *"I would say I don't have a ton of experience messing with uninitialized arrays like this, very quickly when you start programming Java, you learn to initialize things because you'll get errors and so I don't really play around with uninitialized."*

In previous work, researchers have reported confusion on primitive and reference variable assignments [17, 28, 74]. Sorva [28] found that students think assigning primitives is equivalent to assigning references. In the interview, I also encountered this misconception (listed in Table 10.4). Moreover, a common misconception among students is that an array assignment only replaces values in the elements. I found almost all the participants holding this misconception. These two misconceptions regarding assignment operation were also found in the quantitative data (mentioned in Table 8.1) by using the MMT-A.

I found a misconception among several students that variables do not hold default values unless assigned in Java. A participant from Kaczmarczyk et al. [7] work also mentioned a similar belief. One of their participants said, *"I don't think any value is being created for them because there's no assignment there. You know, it's just being declared as a variable"* [7, p.110]. My study participants mentioned an array reference variable holding nothing or blank. A similar belief was found among the participants of Kaczmarczyk et al. [7].

Incorrect assertions 3, 5, and 7 (see Table 10.4) emerged as new misconceptions from our data that I did not include in my incorrect assertions list nor I have seen in the literature. The emergence of these three misconceptions informs us more about the incorrectness of novice programmers' mental models. Moreover, the mutual misconceptions found from the quantitative and qualitative data can suggest that MMT-A can be used as a diagnostic tool to uncover misconceptions.

10.4.3 Exploration of mental model consistency

Throughout the interview process, I observed the evidence that points to the application of consistent mental models in expressing similar knowledge. P1, while answering question Q2 (see Figure 10.2), mentioned indexing starting from 0. P1 then reiterated that while answering question Q6 (see Figure 10.6). I observed students use the same mental model consistency to answer multiple times that a zero or question mark (?) cannot be stored in an array unless assigned. I observed evidence that they were consistently applying the correct mental model that the assignment operator copied from right to left in answering all the questions related to the assignment. Moreover, I noticed P10 consistently applying the incorrect mental model assertion whenever an assignment operation between two arrays was noticed. P10 responded with the incorrect mental model assertion that there should be square brackets on each operand in an assignment operation if we want to specify the variables as arrays.

10.4.4 Impact of Programming Exposure

I observed the impact of programming exposure on the responses of my participants. From Table 10.2, we can notice the participants who had the most programming exposure (completed ITSC: 1212, 1213, and 2214 courses) more accurately answered the questions. From Table 10.2, we can observe the difference in mental model correctness across the participants with different programming exposure. However, for the *parts* components, similar accuracy was found regardless of the difference between programming exposure. The main difference can be noticed in the components of *state changes*. A similar finding was found in the quantitative data. In previous chapters, I mentioned participants with prior programming experience before joining our college scored higher in the *state changes* than participants with no programming experience.

Moreover, I found a prevalence of misconceptions among the participants who had

completed only ITSC: 1212. In Table 10.4, we can notice misconceptions are more prevalent among the participants who had passed only ITSC: 1212.

10.4.5 Implications to the MMT-A

One of the purposes of this semi-structured think-aloud interview study with a subset of questionnaires from the MMT-A was to understand if the students reveal more mental model assertions which are not included in our pre-defined list of assertions. Although most of the assertions I defined and mapped to each choice of the MMT-A matched with the participants' mental model assertion, I found three additional assertions not present in my list. In the context of question Q4 (see in Figure 10.4), participants responded with option (c) (Misc3 in Table 10.4), saying the default size of the array is 0, which can be grown by specifying the size later (maybe hinting at instantiation). However, I mapped option (c) with the assertion 'MD5: Students think memory is allocated'. Since the option mentions a default size, I interpreted the corresponding assertion as MD5. However, participants did not perceive option (c) to be MD5. They said the array's default size is 0, which did not mean that they believed memory was allocated after declaration.

Moreover, I placed a question mark (?) as a distractor in the questions of declarations by reviewing a textbook portraying a question mark as a default value of an array after declaration and instantiation (mentioned in Chapter 3). Therefore, I mapped the choice of question mark (?) with the following assertion: 'MD4: Students think ? is stored as a default value'. MD4 asserts a question mark to be the default value without specifying the array type. However, in the interview session, I found that a participant advocated for the question mark (?) as a default value of an array reference variable only when the array reference variable is character (`char`) typed.

Additionally, in the context of the question Q6 (see Figure 10.6), participants responded with option (b) by stating each of the elements stored `null`. However, I mapped the option (b) with the following assertion: MIn4: After instantiation, no

memory is allocated. P5 appeared to believe memory is not allocated for the array, which matches my mapped assertion. However, P8 believed memory is allocated by saying "*Though null is stored in each element. array instantiation was given for four spaces for the four integers.*" Given this evidence, it suggests we have more possible mental model assertions which need to be considered for analyzing the responses from MMT-A.

10.5 Summary

In this Chapter, I presented the details of the qualitative study I performed. From the findings, I presented a detailed mental model of programming learners of arrays *parts* and *state changes*. The key results of the qualitative study are similar to the quantitative study: students' mental model of array's *state changes* is less accurate and inconsistent than the *parts* components. I also found common misconceptions identified in the quantitative data to be present in the qualitative data. This suggests that students' interpretation of the subset of questions of MMT-A in large are aligned with our sets of assertions. Moreover, from the qualitative findings, I discovered new incorrect assertions that I did not include in my set of incorrect assertions. This finding implies that qualitative data can provide more insight into the improvement of MMT-A.

CHAPTER 11: DISCUSSION AND CONCLUSION

In this Chapter, I discuss the overall findings of my dissertation work. I discuss the implication of my findings to future research and teaching practices. I outline a summary of the contribution resulting from this work. Lastly, I conclude with the limitations of my research studies.

11.1 Overall Summary of Findings

In this dissertation work, I introduced a questionnaire-based instrument, '*the Mental Model Test of Arrays (MMT-A)*' (details are in Chapter 4), to elicit novice programmers' mental models of arrays through the lens of mental model theories based on correctness and consistency. The data revealed several insightful findings aligned with the previous literature.

Students' mental model correctness and consistency for array's *state changes* are lower than the *parts* components. I began with analyzing introductory programming textbooks. With this analysis, I found the portrayal of array's *state changes* had less emphasis. Moreover, the connection with underlying memory with the array's structure and *state changes* was absent in most of the textbooks (discussed in Chapter 3). This finding motivated me to understand novice programmers' mental models of array's *parts* and *state changes*. Students held more incorrect and inconsistent mental models for array's *state changes* than the *parts* components. This finding was common in every phase of data collection.

- Students' mental model correctness and consistency was lower for array's *state changes* than the *parts* components before their classroom instruction of arrays in our CS1 course

- Students who successfully completed the CS1 course, even though their mental model correctness and consistency improved after classroom instruction, the knowledge gap between the *parts* and *state changes* persists.

- Students' persisting misconceptions were mostly seen about the array's *state changes* components.

- Students' qualitative responses revealed confusion in the questions of array's *state changes*.

It is worth noting that understanding *state changes* of arrays requires a well-developed mental model of reference variables and underlying memory usage. A reference variable is a *part* of the structure of an array that is not visible in the code snippet, unlike other *part* such as *name* and *type*. This finding broadly supports the work of other studies in this area. We have seen a plethora of misconceptions related to reference assignments [17, 28], object declaration, and instantiation [7]. Moreover, in my textbook analysis, I found only one textbook explaining array *declaration* with diagrams and only a handful of textbooks depicting how reference assignment works. Perhaps CS1 students could easily connect the *parts* components (i.e., *name, type*) from the code syntax. However, perceiving the underlying steps of dynamic *state changes* is difficult by just perceiving the code syntax. These findings further solidify the claim that the *state changes* components are threshold concept [20–22, 189]. Threshold concepts are potentially very troublesome for students, and mastering them significantly transforms their learning trajectory in an irreversible way so that the transformation is unlikely to be forgotten [189]. Being a troublesome concept, novice programmers are known for holding incorrect or incomplete mental models of *state changes* [17, 29]. From the mental model perspective, my dissertation findings further provide the evidence to solidify the findings from the literature that CS1 students struggle to develop accurate and consistent mental models for arrays *state changes*.

Prior knowledge affects initial mental models. Constructivist belief articulates that learners use their existing knowledge to develop a mental model of a new system. In my work, this phenomenon was found when students entered into a CS1 course. Most students were found to hold correct and consistent mental models for arrays *name* and *type* as, at the time of data collection, they had already learned primitive variables and data types. Moreover, students who had experience with other programming languages had better mental models before classroom instruction (see Chapter 5). This phenomenon was also found when I paired participants' data before and after the CS1 instruction of arrays only for the *parts* components. (see Chapter 7). Whereas prior programming knowledge did not make any impact on the mental models of the *state changes* components. Moreover, in the larger set of post-instruction data, I found no effect of prior knowledge on their mental models (see Chapter 6) on either *parts* or *state changes* components.

Gender difference diminished after instruction. Before classroom instruction on arrays, I found a gender difference in mental models. Male students' mental models were more correct and consistent than the female students. However, this gender gap diminished as the students received instruction on arrays. This may suggest no additional intervention is needed to close the gender gap in terms of mental model development among CS1 students.

11.2 Implications

***State changes* require more attention.** Findings from my work are well aligned with previous literature which elucidates that inaccuracy and inconsistency exist in novice programmers' mental models of array's *state changes*. In Java, similar to arrays, objects are also manipulated with reference variables having the analogous *state changes* (e.g., declaration, instantiation). Therefore, it may be a case that the same difficulties and misconceptions can be found in novice programmers' mental models of objects, which may further hinder their success in more advanced courses such as

Data Structures. The dynamic *state changes* are hidden under abstraction layers. Understanding the dynamic *state changes* and its connection to code is crucial for a programmer to succeed. The ability to shift between levels of abstraction (i.e., concrete code to underlying *state changes*) determines the expertise of a programmer [190].

The purpose of abstraction that makes the *state changes* hidden is not to confuse programmers but rather to provide a new semantic level in which programmers can be absolutely precise. Our job as educators is to uncover the abstraction layers to the programming learner so that they can shift between the abstraction ladder. Moreover, teachers should make the students aware that *state changes* are troublesome, and students need to pay more attention and practice. Despite this importance, Schulte and Bennedsen [191] found programming teachers do not consider providing an overall picture of how programs' dynamic aspects get executed by the notional machine as important. Instructors and educators need to explain more and provide more specific materials for practice that deliberately introduce all the factual assertions of each component of arrays *state changes*. Several visualization tools (e.g., [192, 193], physical memory models [194], object and memory diagrams [195, 196] are proposed to be effective in explaining *state changes*. However, their impact on mental models is less studied. Instructional design strategies that are proven to improve mental models need to be introduced and intervened. The impact of *explanative diagrams* (described in Appendix 11) designed using the empirical multimedia-learning principles based on mental model theories can be assessed in the future step. Moreover, misconception-driven feedback [197] seems promising for addressing misconceptions in mental models.

We need more research on notional machines. A notional machine is a pedagogical tool to explain how programs execute in a given language with an appropriate learner-oriented abstraction [2, 26, 30, 34]. Researchers argue notional machines

can be an appropriate tool to present the hidden layers of programming, hence *state changes*. However, Krishnamurthi et al. [30] argue there is scant research to identify which design and representation of a notional machine is appropriate for the mental model development of different levels of students. My research findings can provide future research directions on the design of notional machines, which can be evaluated to address the mental model gaps (i.e., incorrect assertions) mentioned in my dissertation.

Students and instructors need to be aware of mental models. Mental models govern actions and represent a person's belief system. However, they are individualistic, and often, the learners are unaware of the gaps in the mental model. Moreover, if instructors teaching a concept are unaware of the gaps in students' mental models, there will be no effort to address them. Hence, the flaws in the mental model become fixated, making it harder to change. Therefore, it is very crucial for the learner and the instructor to be aware of the mental models to prevent the fixation of flawed mental models.

CS1 course should introduce a separate section based on prior programming experience. I found the impact of prior programming experience in the initial mental models of novice programmers' when they enter into a CS1 course. This finding suggests there needs to be a separate section based on prior programming experience where instructors can address students based on their initial knowledge.

11.3 Summary of Contributions

This thesis presents the following contributions to the body of research in CS education:

1. A summary of interdisciplinary literature connecting mental models and CS education.

2. A novel instrument to elicit mental models of arrays in a scalable way based on

the theories of mental models- *the Mental Model Test of Arrays(MMT-A).*

3. An analytical framework to analyze and categorize mental models of arrays based on correctness and consistency.

4. An analysis approach to identify misconceptions in mental models based on consistency.

5. A decomposition of the programming concept *arrays* into its *parts* and *state changes.*

6. An understanding of novice programmers' mental models before and after the formal classroom instruction of arrays.

7. An inventory of misconceptions a novice programmer can hold before and after the formal classroom instruction of arrays.

11.4 Limitations

I claim MMT-A as an instrument to analyze novice programmers' mental models of arrays. The results obtained from the MMT-A correlated with a subset of the participants' course performance. Furthermore, the results aligned with previous studies' findings related to misconceptions. However, the generalizability of the results should be made with caution. More empirical studies are needed to generalize the findings. Moreover, the exploration of the validity of the instrument- MMT-A revealed that the data collected for my dissertation could not meet the unidimensionality assumption for conducting 1-PL Item Response Theory (Rashch analysis) or 2-PL Item Response Theory analysis. These analysis results were acceptable, given that MMT-A is primarily used to elicit mental models and is not a high-stakes test. One possible reason for this could be based on the theories of mental models, I have dissected the concept of arrays in *parts* and *state changes* where each part and state change has more components. The mental models for each component are elicited

with a collection of assertions. The result is that I could measure the knowledge of one single concept, 'arrays,' but using more than one dimension. A replication of this study by collecting more data is needed to perform a more reliable validation analysis of our instrument. More insight from additional qualitative data on the use of the instrument can also be added to refine the MMT-A further.

It was noteworthy to find the significant differences in students' mental models due to different factors, such as gender, and prior programming experience. However, as discussed earlier, these trends need further investigation as the sample size for each group for a factor was small and not proportional. In this case, the study can be repeated with a bigger and more proportional sample size to evaluate whether the results are consistent.

In my dissertation work, I elicited CS1 students' mental model assertions. However, I did not investigate how these mental model assertions originated or were adopted. It is possible that a different method of eliciting the mental models (e.g., interviews) might generate different results. Most data for this study was collected during the academic terms impacted by the global pandemic (COVID-19). It is possible that this limited participation from students might have impacted the results or that the results were influenced by the impact that the pandemic had on the entire educational system.

11.5 Future Directions

Here, I share several ideas and directions that can be explored to further advance research in this area. The future directions are not limited to:

Developing Mental Model Tests for Other Programming Concepts. The process to develop the questionnaire of MMT-A by utilizing factual assertions from Java semantics and incorrect assertions from well-known misconceptions can be followed to develop a mental model test for other programming concepts such as *objects*. Moreover, the classification of mental models based on correctness and consistency

can also be applied to classify mental models of other programming concepts.

Improvising to be a Concept Inventory. My purpose for the approach to developing a multiple-choice-based questionnaire was to elicit mental model assertions. Though the development of *MMT-A* and concept inventory is similar, there are key differences in the purpose and design approach (discussed in Section 2.2). However, the process to develop a concept inventory can be followed to improvise the *MMT-A* and validate it with Multi-dimensional Item Response Theory to utilize it as a concept inventory for arrays.

Assessing Educational Intervention. There has been a lack of tests that measure mental models of arrays except the MMT-A. The next step to improve novice programmers' mental models of arrays is to design and assess interventions. The educational interventions can be assessed with MMT-A as a pre-test and post-test to ensure the intervention's impact on mental models. Moreover, techniques such as misconception-driven feedback [197] or refutational lectures [103] can be evaluated by identifying misconceptions using the MMT-A.

Providing automated feedback. There has been a proliferation of the development and usage of automated assessment tools [198, 199] and personalized adaptive learning platforms [200]. The nature of MMT-A being a multiple-choice-based test makes it an appropriate tool to provide automated personalized feedback to students. The integration of MMT-A as an automatic feedback tool in the future can help instructors to assess students' mental models in their learning trajectories. Moreover, personalized learning materials or practice modules can be presented to students based on the gap in their mental models.

Identifying Liminal Space for *State Changes* of Arrays. Liminal space is a transitional period between beginning to learn a concept to fully mastering it [201]. Future research can utilize MMT-A across different course levels to identify the liminal space of array's *state changes*.

11.6 Conclusion

In this dissertation, I presented my definition of a mental model along with a test (MMT-A) to elicit CS1 students' mental models of arrays. From the test, I elicited assertions of CS1 students' mental models at different points (i.e., before instruction and after instruction), allowing us to analyze their mental models' consistency and correctness. CS1 students' mental model correctness and consistency improved from before arrays classroom instruction to the end of the semester. However, even though we saw improvements in the *parts* and *state changes* components, our results revealed that mental models of *state changes* components were less accurate and inconsistent for novice programmers than *parts*. Students with prior programming experience and students without programming experience had similar incorrect and inconsistent mental models for the components of *state changes*. This may suggest that educators need to put more effort into teaching the dynamic invisible aspects of *state changes*. The finding can also suggest that the assertions of those dynamic *state changes* need more time, experience, and practice to be ingrained in novices' mental models. The findings from my dissertation also revealed that prior programming knowledge impacts CS1 students' initial mental model correctness and consistency. This finding supports the belief of constructivism [11] that learners are not clean slates. In the CS education context, the lack of prior programming experience puts students at a disadvantage when compared to other students. This disadvantage persists after a semester of the CS1 course.

In closing, I argue that because mental models resemble a system's *parts* and dynamic behavior (e.g., *state changes*), it gives us an opportunity to capture a mental model and to measure its correctness and consistency. Our findings align with previous literature that shows that *state changes* are difficult to grasp for CS1 students regardless of prior programming experience. The findings from my dissertation strengthen the argument that *state changes* need more attention in future research.

REFERENCES

[1] W. B. Rouse and N. M. Morris, "On looking into the black box: Prospects and limits in the search for mental models.," *Psychological bulletin*, vol. 100, no. 3, p. 349, 1986.

[2] S. Fincher, J. Jeuring, C. S. Miller, P. Donaldson, B. du Boulay, M. Hauswirth, A. Hellas, F. Hermans, C. Lewis, A. Mühling, *et al.*, "Notional machines in computing education: The education of attention," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 21–50, 2020.

[3] P. E. Dickson, N. C. Brown, and B. A. Becker, "Engage against the machine: Rise of the notional machines as effective pedagogical devices," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 159–165, 2020.

[4] R. E. Mayer, "Cognitive theory of multimedia learning," *The Cambridge handbook of multimedia learning*, vol. 41, pp. 31–48, 2005.

[5] R. Mayer and J. K. Gallini, "When is an illustration worth ten thousand words?," *Journal of Educational Psychology*, vol. 82, pp. 715–726, 12 1990.

[6] D. A. Norman, "Some observations on mental models," in *Mental models*, pp. 15–22, Psychology Press, 1983.

[7] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proceedings of the 41st ACM technical symposium on Computer science education*, pp. 107–111, ACM, 2010.

[8] A. Swidan, F. Hermans, and M. Smit, "Programming misconceptions for school students," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pp. 151–159, 2018.

[9] N. J. Nersessian, "The cognitive basis of model-based reasoning in science," *The cognitive basis of science*, pp. 133–153, 2002.

[10] J. Hammarbäck, "Finding paths or getting lost?: Examining the mental model construct and mental model methodology," 2017.

[11] J. Piaget, *Construction of Reality in the Child: Translated by Margaret Cook*. Basic Books, 1954.

[12] J. Proulx, "Constructivism: A re-equilibration and clarification of the concepts, and some potential implications for teaching and pedagogy," *Radical pedagogy*, vol. 8, no. 1, pp. 65–85, 2006.

[13] A. K. Taylor and P. Kowalski, "Naïve psychological science: The prevalence, strength, and sources of misconceptions," *The Psychological Record*, vol. 54, pp. 15–25, 2004.

[14] R. Hanson, A. Sam, and V. Antwi, "Misconceptions of undergraduate chemistry teachers about hybridisation," *African Journal of Educational Studies in Mathematics and Sciences*, vol. 10, pp. 45–54, 2012.

[15] J. Otero, "Influence of knowledge activation and context on comprehension monitoring of science texts," *Metacognition in educational theory and practice*, pp. 145–164, 1998.

[16] S. Vosniadou, "What can persuasion research tell us about conceptual change that we did not already know?," *International Journal of Educational Research*, vol. 35, no. 7-8, pp. 731–737, 2001.

[17] L. Ma, *Investigating and improving novice programmers' mental models of programming concepts.* PhD thesis, University of Strathclyde, 2007.

[18] S. Dehnadi, *A cognitive study of learning to program in introductory programming courses.* PhD thesis, Middlesex University, 2009.

[19] J. Sorva and T. Sirkiä, "Uuhistle: A software tool for visual program simulation," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, 2010.

[20] D. Shinners-Kennedy, "The everydayness of threshold concepts: State as an example from computer science," in *Threshold concepts within the disciplines*, pp. 119–128, Brill, 2008.

[21] J. Rountree and N. Rountree, "Issues regarding threshold concepts in computer science," in *Proceedings of the Eleventh Australasian Conf. on Computing Education-Vol. 95*, pp. 139–146, 2009.

[22] K. Sanders and R. McCartney, "Threshold concepts in computing: past, present, and future," in *Proceedings of the 16th Koli Calling international conference on computing education research*, pp. 91–100, 2016.

[23] K. Cunningham, "Purpose-first programming: A programming learning approach for learners who care most about what code achieves," in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pp. 348–349, 2020.

[24] E. Vagianou, "Program working storage: a beginner's model," in *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, pp. 69–76, 2006.

[25] T. B. Weidmann, S. Thorgeirsson, and Z. Su, "Bridging the syntax-semantics gap of programming," in *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 80–94, 2022.

[26] B. Du Boulay, "Some difficulties of learning to program," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 57–73, 1986.

[27] V. Lonati, A. Brodnik, T. Bell, A. P. Csizmadia, L. De Mol, H. Hickman, T. Keane, C. Mirolo, and M. Monga, "What we talk about when we talk about programs," in *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 117–164, 2022.

[28] J. Sorva, "The same but different students' understandings of primitive and object variables," in *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, (New York, NY, USA), pp. 5–15, ACM, 2008.

[29] J. Sorva, "Misconceptions and the beginner programmer," *Computer science education: Perspectives on teaching and learning in school*, vol. 171, 2018.

[30] S. Krishnamurthi and K. Fisler, "Programming paradigms and beyond," *The Cambridge Handbook of Computing Education Research*, vol. 37, 2019.

[31] F. G. Halasz and T. P. Moran, "Mental models and problem solving in using a calculator," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 212–216, 1983.

[32] D. E. Kieras and S. Bovair, "The role of a mental model in learning to operate a device," *Cognitive science*, vol. 8, no. 3, pp. 255–273, 1984.

[33] J. M. Carroll and J. R. Olson, "Mental models in human-computer interaction," *Handbook of human-computer interaction*, pp. 45–65, 1988.

[34] J. Sorva, "Notional machines and introductory programming education," *Trans. Comput. Educ.*, vol. 13, July 2013.

[35] P. Johnson-Laird, B. Gawronski, and F. Strack, "Mental models and consistency," *Cognitive consistency: A fundamental principle in social cognition*, pp. 225–243, 2012.

[36] J. De Kleer and J. S. Brown, "Mental models of physical mechanisms and their acquisition," *Cognitive skills and their acquisition*, pp. 285–309, 1981.

[37] J. De Kleer and J. S. Brown, "Assumptions and ambiguities in mechanistic mental models," in *Mental models*, pp. 163–198, Psychology Press, 2014.

[38] T. M. Haladyna, *Developing and validating multiple-choice test items*. Routledge, 2004.

[39] K. Craik, "The nature of explanation cambridge university press: Cambridge," 1943.

[40] J. K. Doyle, D. N. Ford, M. J. Radzicki, and W. S. Trees, "Mental models of dynamic systems," *Encyclopedia of Life Support Systems*, 2001.

[41] N. A. Jones, H. Ross, T. Lynam, P. Perez, and A. Leitch, "Mental models: an interdisciplinary synthesis of theory and methods," *Ecology and Society*, vol. 16, no. 1, 2011.

[42] B. Gawronski and S. M. Brannon, "What is cognitive consistency, and why does it matter?," 2019.

[43] B. Gawronski and G. V. Bodenhausen, "Self-insight from a dual-process perspective," *Handbook of self-knowledge*, pp. 22–38, 2012.

[44] S. M. Glynn, B. K. Britton, and R. H. Yeany, *The psychology of learning science*. Routledge, 2012.

[45] R. B. Davis *et al.*, *Constructivist Views on the Teaching and Learning of Mathematics. Journal for Research in Mathematics Education: Monograph No. 4.* ERIC, 1990.

[46] M. Ben-Ari, "Constructivism in computer science education," *Journal of Computers in Mathematics and Science Teaching*, vol. 20, no. 1, pp. 45–73, 2001.

[47] M. Ben-Ari, "Constructivism in computer science education," *Acm sigcse bulletin*, vol. 30, no. 1, pp. 257–261, 1998.

[48] J. Rasmussen, *On the structure of knowledge-a morphology of metal models in a man-machine system context*. Risø National Laboratory, 1979.

[49] W. Veldhuyzen and H. G. Stassen, "The internal model concept: An application to modeling human control of large ships," *Human Factors*, vol. 19, no. 4, pp. 367–380, 1977.

[50] C. D. Wickens, J. G. Hollands, S. Banbury, and R. Parasuraman, *Engineering psychology and human performance*. Psychology Press, 2015.

[51] M. D. Williams, J. D. Hollan, and A. L. Stevens, "Human reasoning about a simple physical system," *Mental models*, pp. 131–154, 1983.

[52] R. M. Young, "Surrogates and mappings: Two kinds of conceptual models for interactive devices," in *Mental models*, pp. 43–60, Psychology Press, 2014.

[53] D. Gentner and D. R. Gentner, "Flowing waters or teeming crowds: Mental models of electricity," in *Mental models*, pp. 107–138, Psychology Press, 2014.

[54] M. Hegarty, M. A. Just, and I. R. Morrison, "Mental models of mechanical systems: Individual differences in qualitative and quantitative reasoning," *Cognitive Psychology*, vol. 20, no. 2, pp. 191–236, 1988.

[55] J. de Kleer and J. S. Brown, "Qualitative reasoning about physical systems, chapter qualitative physics based on confluences," 1985.

[56] D. Gentner and A. L. Stevens, "Mental models," 1983.

[57] J. H. Larkin and H. A. Simon, "Why a diagram is (sometimes) worth ten thousand words," *Cognitive science*, vol. 11, no. 1, pp. 65–100, 1987.

[58] B. Y. White and J. R. Frederiksen, "Qualitative models and intelligent learning," *Artificial Intelligence and Education: Learning environments and tutoring systems*, vol. 1, p. 281, 1987.

[59] J. M. Clark and A. Paivio, "Dual coding theory and education," *Educational Psychology Review*, vol. 3, pp. 149–210, Sep 1991.

[60] W. Schnotz, "Commentary: Towards an integrated view of learning from text and visual displays," *Educational psychology review*, vol. 14, no. 1, pp. 101–120, 2002.

[61] R. N. Carney and J. R. Levin, "Pictorial illustrations still improve students' learning from text," *Educational psychology review*, vol. 14, no. 1, pp. 5–26, 2002.

[62] S. Ainsworth, "Deft: A conceptual framework for considering learning with multiple representations," *Learning and instruction*, vol. 16, no. 3, pp. 183–198, 2006.

[63] B. Eilam and Y. Poyas, "External visual representations in science learning: The case of relations among system components," *International Journal of Science Education*, vol. 32, no. 17, pp. 2335–2366, 2010.

[64] L. Leivas Pozzer and W.-M. Roth, "Prevalence, function, and structure of photographs in high school biology textbooks," *Journal of Research in Science Teaching*, vol. 40, no. 10, pp. 1089–1114, 2003.

[65] K. R. Butcher, "Learning from text with diagrams: Promoting mental model development and inference generation.," *Journal of Educational Psychology*, vol. 98, no. 1, p. 182, 2006.

[66] S. Byram, B. Fischhoff, M. Embrey, W. Bruine de Bruin, and S. Thorne, "Mental models of women with breast implants: Local complications," *Behavioral Medicine*, vol. 27, no. 1, pp. 4–14, 2001.

[67] S. J. Hysong, R. G. Best, J. A. Pugh, and F. I. Moore, "Not of one mind: mental models of clinical practice guidelines in the veterans health administration," *Health services research*, vol. 40, no. 3, pp. 829–848, 2005.

[68] T. Darisi, S. Thorne, and C. Iacobelli, "Influences on decision-making for undergoing plastic surgery: a mental models and quantitative assessment," *Plastic and Reconstructive Surgery*, vol. 116, no. 3, pp. 907–916, 2005.

[69] M. Van Someren, Y. F. Barnard, and J. Sandberg, "The think aloud method: a practical approach to modelling cognitive," *London: AcademicPress*, vol. 11, pp. 29–41, 1994.

[70] G. Pask and B. C. Scott, "Learning strategies and individual competence," *International Journal of Man-Machine Studies*, vol. 4, no. 3, pp. 217–253, 1972.

[71] L. A. Freeman and L. M. Jessup, "The power and benefits of concept mapping: measuring use, usefulness, ease of use, and satisfaction," *International Journal of Science Education*, vol. 26, no. 2, pp. 151–169, 2004.

[72] W. C. McGaghie, D. R. McCrimmon, G. Mitchell, and J. A. Thompson, "Concept mapping in pulmonary physiology using pathfinder scaling," *Advances in Health Sciences Education*, vol. 9, no. 3, pp. 225–240, 2004.

[73] C. Wang and M. A. Burris, "Photovoice: Concept, methodology, and use for participatory needs assessment," *Health education & behavior*, vol. 24, no. 3, pp. 369–387, 1997.

[74] S. Dehnadi, R. Bornat, *et al.*, "The camel has two humps (working title)," *Middlesex University, UK*, pp. 1–21, 2006.

[75] A. Radermacher, G. Walia, and R. Rummelt, "Improving student learning outcomes with pair programming," in *Proceedings of the ninth annual international conference on International computing education research*, pp. 87–92, 2012.

[76] V. Ramalingam, D. LaBelle, and S. Wiedenbeck, "Self-efficacy and mental models in learning to program," in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pp. 171–175, 2004.

[77] H. Julie and D. Bruno, "Approach to develop a concept inventory informing teachers of novice programmers' mental models," in *2020 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9, IEEE, 2020.

[78] L. Wittie, A. Kurdia, and M. Huggard, "Developing a concept inventory for computer science 2," in *2017 IEEE Frontiers in Education Conference (FIE)*, pp. 1–4, IEEE, 2017.

[79] A. K. Taylor and P. Kowalski, "Student misconceptions: Where do they come from and what can we do?," *Scholarship of Teaching and Learning in Psychology*, 2014.

[80] K. Goldman, P. Gross, C. Heeren, G. L. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles, "Setting the scope of concept inventories for introductory computing subjects," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 2, pp. 1–29, 2010.

[81] I. A. Halloun and D. Hestenes, "The initial knowledge state of college physics students," *American journal of Physics*, vol. 53, no. 11, pp. 1043–1055, 1985.

[82] D. Hestenes, M. Wells, and G. Swackhamer, "Force concept inventory," *The physics teacher*, vol. 30, no. 3, pp. 141–158, 1992.

[83] A. E. Tew, *Assessing fundamental introductory computing concept knowledge in a language independent manner*. Georgia Institute of Technology, 2010.

[84] A. E. Tew and M. Guzdial, "The fcs1: a language independent assessment of cs1 knowledge," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, pp. 111–116, 2011.

[85] M. C. Parker, M. Guzdial, and A. E. Tew, "Uses, revisions, and the future of validated assessments in computing education: A case study of the fcs1 and scs1," in *Proceedings of the 17th ACM Conference on International Computing Education Research*, pp. 60–68, 2021.

[86] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, validation, and use of a language independent cs1 knowledge assessment," in *Proceedings of the 2016 ACM conference on international computing education research*, pp. 93–101, 2016.

[87] B. Xie, M. J. Davidson, M. Li, and A. J. Ko, "An item response theory evaluation of a language-independent cs1 knowledge assessment," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 699–705, 2019.

[88] A. E. Tew and B. Dorn, "The case for validated tools in computer science education research," *Computer*, vol. 46, no. 9, pp. 60–66, 2013.

[89] M. J. Allen and W. M. Yen, *Introduction to measurement theory*. Waveland Press, 2001.

[90] T. L. Kelley, "Interpretation of educational measurements.," 1927.

[91] J. D. Brown, "What is construct validity? what is construct validity?,"

[92] L. J. Cronbach and P. E. Meehl, "Construct validity in psychological tests.," *Psychological bulletin*, vol. 52, no. 4, p. 281, 1955.

[93] D. F. Polit and C. T. Beck, *Nursing research: Generating and assessing evidence for nursing practice*. Lippincott Williams & Wilkins, 2008.

[94] D. Gurel, A. Eryilmaz, and L. McDermott, "A review and comparison of diagnostic instruments to identify students' misconceptions in science," *Eurasia Journal of Mathematics Science and Technology Education*, vol. 11, no. 5, 2015.

[95] B. J. Guzzetti, T. E. Snyder, G. V. Glass, and W. S. Gamas, "Promoting conceptual change in science: A comparative meta-analysis of instructional interventions from reading education and science education," *Reading Research Quarterly*, pp. 117–159, 1993.

[96] A. A. diSessa, "A history of conceptual change research," in *The Cambridge Handbook of the Learning Sciences*, pp. 88–108, Cambridge University Press, Sept. 2014.

[97] E. D. Vaughan, "Misconceptions about psychology among introductory psychology students," *Teaching of psychology*, vol. 4, no. 3, pp. 138–141, 1977.

[98] A. Eryilmaz, "Effects of conceptual assignments and conceptual change discussions on students' misconceptions and achievement regarding force and motion," *Journal of research in science teaching*, vol. 39, no. 10, pp. 1001–1015, 2002.

[99] E. Boyes and M. Stanisstreet, "Misconceptions in first-year undergraduate science students about energy sources for living organisms," *Journal of Biological Education*, vol. 25, no. 3, pp. 209–213, 1991.

[100] S. Vosniadou, "16 universal and culture-specific properties of," *Mapping the mind: Domain specificity in cognition and culture*, p. 412, 1994.

[101] A. K. Taylor and P. Kowalski, "Naïve psychological science: The prevalence, strength, and sources of misconceptions," *The Psychological Record*, vol. 54, pp. 15–25, 2004.

[102] P. M. Sadler, G. Sonnert, H. P. Coyle, N. Cook-Smith, and J. L. Miller, "The influence of teachers' knowledge on student learning in middle school physical science classrooms," *American Educational Research Journal*, vol. 50, no. 5, pp. 1020–1049, 2013.

[103] P. Kowalski and A. K. Taylor, "The effect of refuting misconceptions in the introductory psychology class," *Teaching of Psychology*, vol. 36, no. 3, pp. 153–159, 2009.

[104] P. Kowalski and A. K. Taylor, "Reducing students' misconceptions with refutational teaching: For long-term retention, comprehension matters.," *Scholarship of Teaching and Learning in Psychology*, vol. 3, no. 2, p. 90, 2017.

[105] D. A. Bensley and S. O. Lilienfeld, "What is a psychological misconception? moving toward an empirical answer," *Teaching of Psychology*, vol. 42, no. 4, pp. 282–292, 2015.

[106] D. E. Trowbridge and L. C. McDermott, "Investigation of student understanding of the concept of velocity in one dimension," *American journal of Physics*, vol. 48, no. 12, pp. 1020–1028, 1980.

[107] A. Caramazza, M. McCloskey, and B. Green, "Naive beliefs in "sophisticated" subjects: Misconceptions about trajectories of objects," *Cognition*, vol. 9, no. 2, pp. 117–123, 1981.

[108] D. R. Mulford and W. R. Robinson, "An inventory for alternate conceptions among first-semester general chemistry students," *Journal of chemical education*, vol. 79, no. 6, p. 739, 2002.

[109] P. M. Sadler, "Psychometric models of student conceptions in science: Reconciling qualitative studies and distractor-driven assessment instruments," *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching*, vol. 35, no. 3, pp. 265–296, 1998.

[110] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, p. 1, 2017.

[111] E. Kurvinen, N. Hellgren, E. Kaila, M.-J. Laakso, and T. Salakoski, "Programming misconceptions in an introductory level programming course exam," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 308–313, 2016.

[112] L. Chiodini, I. Moreno Santos, A. Gallidabino, A. Tafliovich, A. L. Santos, and M. Hauswirth, "A curated inventory of programming language misconceptions," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pp. 380–386, 2021.

[113] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo, "Identifying and validating java misconceptions toward a cs1 concept inventory," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 23–29, 2019.

[114] J. Sorva *et al.*, *Visual program simulation in introductory programming education*. Aalto University, 2012.

[115] M. Teif and O. Hazzan, "Partonomy and taxonomy in object-oriented thinking: Junior high school students' perceptions of object-oriented basic concepts," in *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '06, (New York, NY, USA), pp. 55–60, ACM, 2006.

[116] N. Ragonis and M. Ben-Ari, "A long-term investigation of the comprehension of oop concepts by novices," *Computer Science Education*, vol. 15, no. 3, pp. 203–221, 2005.

[117] E. Albrecht and J. Grabowski, "Sometimes it's just sloppiness-studying students' programming errors and misconceptions," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pp. 340–345, 2020.

[118] S. Holland, R. Griffiths, and M. Woodman, "Avoiding object misconceptions," in *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '97, (New York, NY, USA), pp. 131–134, ACM, 1997.

[119] J. Sorva, "Students' understandings of storing objects," Koli Calling '07, (AUS), p. 127–135, Australian Computer Society, Inc., 2007.

[120] T. Sirkiä and J. Sorva, "Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises," in *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, pp. 19–28, 2012.

[121] S. F. Mazumder, C. Latulipe, and M. A. Pérez-Quiñones, "Are variable, array and object diagrams in java textbooks explanative?," in *Proceedings of the 2020 ACM conference on innovation and technology in computer science education*, pp. 425–431, 2020.

[122] R. McFall, H. Dershem, and D. Davis, "Experiences using a collaborative electronic textbook: Bringing the "guide on the side" home with you," in *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, (New York, NY, USA), pp. 339–343, ACM, 2006.

[123] L. W. Foderaro, "In a digital age, students still cling to paper textbooks.." https://www.nytimes.com/2010/10/20/nyregion/20textbooks.html, 2010. Accessed: 2019-03-26.

[124] G. A. Valverde, L. J. Bianchi, R. G. Wolfe, W. H. Schmidt, and R. T. Houang, *According to the book: Using TIMSS to investigate the translation of policy into practice through the world of textbooks*. Springer Science & Business Media, 2002.

[125] M. Berges and P. Hubwieser, "Concept specification maps: Displaying content structures," in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, 2013.

[126] K. McMaster, B. Rague, S. Sambasivam, and S. Wolthuis, "Coverage of cs1 programming concepts in c++ and java textbooks," in *2016 IEEE Frontiers in Education Conference (FIE)*, 2016.

[127] J. Börstler, M. E. Caspersen, and M. Nordström, "Beauty and the beast: on the readability of object-oriented example programs," *Software Quality Journal*, 2016.

[128] "Amazon's best sellers.." https://www.amazon.com/Best-Sellers-Books-Java-Programming/zgbs/books/3608, 2016. Accessed: 2018-06-28.

[129] "Object-oriented programming textbooks.." https://www.barnesandnoble.com/b/textbooks/computer-programming/object-oriented-programming/_/N-8q9Zvok, 2015. Accessed: 2019-07-02.

[130] K. B. Bruce, A. P. Danyluk, and T. P. Murtagh, *Java: An eventful approach*. Pearson Prentice Hall, 2006.

[131] A. L. Roman Lysecky, "Java early objects." https://www.zybooks.com/catalog/java-early-objects/. Accessed: 2019-07-12.

[132] P. S. Nair, *Java programming fundamentals: problem solving through object oriented analysis and design*. CRC press, 2008.

[133] D. M. Arnow and G. Weiss, *Introduction to programming using java: an object-oriented approach*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[134] J. C. Adams, L. R. Nyhoff, and J. L. Nyhoff, *Java: An Introduction to Computing*. Prentice-Hall, Inc., 2001.

[135] T. Gaddis, S. Mukherjee, and A. K. Bhattacherjee, *Starting out with Java: From control structures through objects*. Pearson, 2013.

[136] P. Deitel and H. Deitel, *Java How to program*. Prentice Hall Press, 2011.

[137] P. Naughton and H. Schildt, *Java: The complete reference*. McGraw-Hill, Inc., 1996.

[138] W. Savitch, *Java: An Introduction to Problem Solving and Programming, Student Value Edition Plus MyProgrammingLab with Pearson eText-Access Card Package*. Pearson, 2017.

[139] Y. D. Liang, *Introduction to Java programmng: comprehensive version*. Pearson Education, 2011.

[140] D. S. Malik, *JavaTM Programming: From Problem Analysis to Program Design*. Cengage learning, 2011.

[141] C. S. Horstmann, *Java Concepts*. New York, NY, USA: John Wiley & Sons, Inc., 4th ed., 2007.

[142] S. Reges and M. Stepp, "Building java programs: A back to basics approach plus myprogramminglab with pearson etext–access card package," 2016.

[143] M. Guzdial and B. Ericson, *Introduction to computing & programming in Java: a multimedia approach*. Pearson Prentice Hall, 2007.

[144] C. T. Wu, *An Introduction to Object-Oriented Programming with Java*. McGraw-Hill Pub. Co., 6th ed., 2006.

[145] A. L. Santos and H. Sousa, "An exploratory study of how programming instructors illustrate variables and control flow," in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, Koli Calling '17, (New York, NY, USA), pp. 173–177, ACM, 2017.

[146] B. Tversky, "Spatial mental models," *Psychology of Learning and Motivation*, vol. 27, pp. 109–145, 1991.

[147] P. N. Johnson-Laird, *How we reason*. Oxford University Press, USA, 2006.

[148] T. M. Haladyna and S. M. Downing, "A taxonomy of multiple-choice item-writing rules," *Applied measurement in education*, vol. 2, no. 1, pp. 37–50, 1989.

[149] B. Bettin, "Toward understanding and enhancing novice students' mental models in computer science," in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pp. 323–324, 2019.

[150] E. Albrecht and J. Grabowski, "Sometimes it's just sloppiness - studying students' programming errors and misconceptions," SIGCSE '20, (New York, NY, USA), p. 340–345, Association for Computing Machinery, 2020.

[151] M. Hristova, A. Misra, M. Rutter, and R. Mercuri, "Identifying and correcting java programming errors for introductory computer science students," *ACM SIGCSE Bulletin*, vol. 35, no. 1, pp. 153–156, 2003.

[152] T. Sirkiä *et al.*, "Recognizing programming misconceptions-an analysis of the data collected from the uuhistle program simulation tool," Master's thesis, 2012.

[153] D. Doukakis, M. Grigoriadou, and G. Tsaganou, "Understanding the programming variable concept with animated interactive analogies," in *Proceedings of the The 8th Hellenic European Research on Computer Mathematics & Its Applications Conference (HERCMA'07)*, 2007.

[154] P. Bayman and R. E. Mayer, "Using conceptual models to teach basic computer programming.," *Journal of Educational Psychology*, vol. 80, no. 3, p. 291, 1988.

[155] R. T. Putnam, D. Sleeman, J. A. Baxter, and L. K. Kuspa, "A summary of misconceptions of high school basic programmers," *Journal of Educational Computing Research*, vol. 2, no. 4, pp. 459–472, 1986.

[156] B. Ericson, B. Hoffman, and J. Rosato, "Csawesome: Ap csa curriculum and professional development (practical report)," in *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, WiPSCE '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[157] E. Charters, "The use of think-aloud methods in qualitative research an introduction to think-aloud methods," *Brock Education Journal*, vol. 12, no. 2, 2003.

[158] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating the viability of mental models held by novice programmers," in *Proceedings of the 38th SIGCSE technical symposium on computer science education*, pp. 499–503, 2007.

[159] M.-Q. Syeda Fatema Mazumder, Celine Latulipe, "Are variable, array and object diagrams in java textbooks explanative?," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE 2020, in press.

[160] R. E. Mayer and V. K. Sims, "For whom is a picture worth a thousand words? extensions of a dual-coding theory of multimedia learning.," *Journal of educational psychology*, vol. 86, no. 3, p. 389, 1994.

[161] L. Murphy and L. Thomas, "Dangers of a fixed mindset: implications of self-theories research for computer science education," in *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pp. 271–275, 2008.

[162] A. Tafliovich, J. Campbell, and A. Petersen, "A student perspective on prior experience in cs1," in *Proceeding of the 44th ACM technical symposium on Computer science education*, pp. 239–244, 2013.

[163] L. J. Barker, C. McDowell, and K. Kalahar, "Exploring factors that influence computer science introductory course students to persist in the major," *ACM Sigcse Bulletin*, vol. 41, no. 1, pp. 153–157, 2009.

[164] L. K. Alford, M. L. Dorf, and V. Bertacco, "Student perceptions of their abilities and learning environment in large introductory computer programming courses," in *2017 ASEE Annual Conference & Exposition*, 2017.

[165] L. L. Beck, A. W. Chizhik, and A. C. McElroy, "Cooperative learning techniques in cs1: design and experimental evaluation," *ACM SIGCSE Bulletin*, vol. 37, no. 1, pp. 470–474, 2005.

[166] A. Lishinski, A. Yadav, J. Good, and R. Enbody, "Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, pp. 211–220, 2016.

[167] A. Settle, J. Lalor, and T. Steinbach, "Reconsidering the impact of cs1 on novice attitudes," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pp. 229–234, 2015.

[168] S. F. Mazumder and M. A. Pérez-Quiñones, "Incoming cs1 students' misconceptions on arrays," in *2023 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9, IEEE, 2023.

[169] M. Clancy, J. Stasko, M. Guzdial, S. Fincher, and N. Dale, "Models and areas for cs education research," *Computer Science Education*, vol. 11, no. 4, pp. 323–341, 2001.

[170] R. D. Pea, "Language-independent conceptual "bugs" in novice programming," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 25–36, 1986.

[171] G. L. Herman, *The development of a digital logic concept inventory*. University of Illinois at Urbana-Champaign, 2011.

[172] C. Lewis and D. A. Norman, "Designing for error," in *Readings in human–computer interaction*, pp. 686–697, Elsevier, 1995.

[173] T. Albano, "Introduction to educational and psychological measurement using R," 2018.

[174] R. K. Hambleton and R. W. Jones, "Comparison of classical test theory and item response theory and their applications to test development," *Educational measurement: issues and practice*, vol. 12, no. 3, pp. 38–47, 1993.

[175] S. E. Stemler and A. Naples, "Rasch measurement v. item response theory: Knowing when to cross the line.," *Practical Assessment, Research & Evaluation*, vol. 26, p. 11, 2021.

[176] K. B. Christensen, S. Kreiner, and M. Mesbah, *Rasch models in health*. John Wiley & Sons, 2013.

[177] W. J. Boone, J. R. Staver, and M. S. Yale, *Rasch analysis in the human sciences*. Springer, 2013.

[178] J. Linacre, "A user's guide to winsteps ministeps," *Rasch model computer programs manual*, vol. 3, no. 0, 2011.

[179] S. E. Stemler and A. Naples, "Rasch measurement v. item response theory: Knowing when to cross the line.," *Practical Assessment, Research & Evaluation*, vol. 26, p. 11, 2021.

[180] B. D. Wright and G. N. Masters, *Rating scale analysis*. MESA press, 1982.

[181] R. G. Lambert, "Technical manual for the teaching strategies gold®," 2020.

[182] T. G. Bond, C. M. Fox, and H. Lacey, "Applying the rasch model: Fundamental measurement," in *in the social sciences (2nd*, Citeseer, 2007.

[183] M. Wilson, *Constructing Measures: An Item Response Modeling Approach: An Item Response Modeling Approach*. Routledge, 2004.

[184] M. Wu and R. Adams, *Applying the Rasch model to psycho-social measurement: A practical approach*. Educational Measurement Solutions Melbourne, 2007.

[185] T. G. Bond and C. M. Fox, *Applying the Rasch model: Fundamental measurement in the human sciences*. Psychology Press, 2013.

[186] J. W. Osborne, *Best practices in exploratory factor analysis*. CreateSpace Independent Publishing Platform, 2014.

[187] M. D. Reckase, *Multidimensional Item Response Theory*. Springer, 2009.

[188] K. Charmaz, *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.

[189] J. Sorva, "Reflections on threshold concepts in computer programming and beyond," in *Proceedings of the 10th Koli calling intl. conf. on computing education research*, pp. 21–30, 2010.

[190] J. Hartmanis, "Turing award lecture on computational complexity and the nature of computer science," *Commun. ACM*, vol. 37, p. 37–43, oct 1994.

[191] C. Schulte and J. Bennedsen, "What do teachers teach in introductory programming?," in *Proceedings of the second international workshop on Computing education research*, pp. 17–28, 2006.

[192] P. J. Guo, "Online python tutor: Embeddable web-based program visualization for cs education," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, 2013.

[193] J. H. Cross, D. Hendrix, and D. A. Umphress, "Jgrasp: an integrated development environment with visualizations for teaching java in cs1, cs2, and beyond," in *34th Annual Frontiers in Education, 2004. FIE 2004.*, pp. 1466–1467, 2004.

[194] C. M. Lewis, "Physical java memory models: A notional machine," SIGCSE '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[195] L. Thomas, M. Ratcliffe, and B. Thomasson, "Scaffolding with object diagrams in first year programming classes: Some unexpected results," 03 2004.

[196] M. Holliday and D. Luginbuhl, "Using memory diagrams when teaching a java-based cs1," 04 2018.

[197] L. Gusukuma, A. C. Bart, D. Kafura, and J. Ernst, "Misconception-driven feedback: Results from an experimental study," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pp. 160–168, 2018.

[198] S. H. Edwards and K. P. Murali, "Codeworkout: Short programming exercises with built-in data collection," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, (New York, NY, USA), p. 188–193, Association for Computing Machinery, 2017.

[199] "Gradescope." https://www.gradescope.com/, 2020. Accessed: 2024-01-18.

[200] S. Marwan, G. Gao, S. Fisk, T. W. Price, and T. Barnes, "Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science," in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ICER '20, (New York, NY, USA), p. 194–203, Association for Computing Machinery, 2020.

[201] R. McCartney, J. Boustedt, A. Eckerdal, J. E. Moström, K. Sanders, L. Thomas, and C. Zander, "Liminal spaces and learning computing," *European Journal of Engineering Education*, vol. 34, no. 4, pp. 383–391, 2009.

[202] B. du Boulay, T. O'Shea, and J. Monk, "The black box inside the glass box: presenting computing concepts to novices," *International Journal of Man-Machine Studies*, vol. 14, no. 3, pp. 237 – 249, 1981.

[203] J. S. Bruner *et al.*, *Toward a theory of instruction*, vol. 59. Harvard University Press, 1966.

[204] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer science education*, vol. 13, no. 2, pp. 137–172, 2003.

[205] M. Guzdial, S. Krishnamurthi, J. Sorva, and J. Vahrenhold, "Notional machines and programming language semantics in education (dagstuhl seminar 19281)," in *Dagstuhl Reports*, vol. 9, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[206] H. Kragh, *Niels Bohr and the quantum atom: The Bohr model of atomic structure 1913-1925*. OUP Oxford, 2012.

[207] I. Newton, *Philosophiae naturalis principia mathematica*, vol. 2. typis A. et JM Duncan, 1833.

[208] P. E. Smaldino, "Models are stupid, and we need more of them," *Computational social psychology*, pp. 311–331, 2017.

[209] R. Mayer and R. E. Mayer, *The Cambridge handbook of multimedia learning*. Cambridge university press, 2005.

[210] R. E. Mayer, "Multimedia learning," in *Psychology of learning and motivation*, vol. 41, pp. 85–139, Elsevier, 2002.

[211] R. E. Mayer, "The promise of multimedia learning: using the same instructional design methods across different media," *Learning and instruction*, vol. 13, no. 2, pp. 125–139, 2003.

[212] R. E. Mayer and R. Moreno, "A split-attention effect in multimedia learning: Evidence for dual processing systems in working memory.," *Journal of educational psychology*, vol. 90, no. 2, p. 312, 1998.

[213] M. S. Donovan, J. D. Bransford, and J. W. Pellegrino, "How people learn," *Retrieved March*, vol. 8, p. 2006, 1999.

[214] N. M. Lambert and B. L. McCombs, *How students learn: Reforming schools through learner-centered education.* American Psychological Association, 1998.

[215] A. Paivio, *Mental representations: A dual coding approach.* Oxford University Press, 1990.

[216] A. Baddeley, "Working memory oxford," *England: Oxford Uni*, 1986.

[217] A. D. Baddeley, *Essentials of human memory.* Psychology Press, 1999.

[218] P. Chandler and J. Sweller, "Cognitive load theory and the format of instruction," *Cognition and instruction*, vol. 8, no. 4, pp. 293–332, 1991.

[219] R. E. Mayer, W. Bove, A. Bryman, R. Mars, and L. Tapangco, "When less is more: Meaningful learning from visual and verbal summaries of science textbook lessons.," *Journal of educational psychology*, vol. 88, no. 1, p. 64, 1996.

[220] R. C. Atkinson and R. M. Shiffrin, "The control of short-term memory," *Scientific american*, vol. 225, no. 2, pp. 82–91, 1971.

[221] A. Baddeley, "The episodic buffer: a new component of working memory?," *Trends in cognitive sciences*, vol. 4, no. 11, pp. 417–423, 2000.

[222] W. Schnotz, "An integrated model of text and picture comprehension," *The Cambridge handbook of multimedia learning*, vol. 49, p. 69, 2005.

[223] R. E. Mayer, "Systematic thinking fostered by illustrations in scientific text.," *Journal of educational psychology*, vol. 81, no. 2, p. 240, 1989.

[224] R. E. Mayer, K. Steinhoff, G. Bower, and R. Mars, "A generative theory of textbook design: Using annotated illustrations to foster meaningful learning of science text," *Educational Technology Research and Development*, vol. 43, no. 1, pp. 31–41, 1995.

[225] R. Moreno and R. E. Mayer, "Cognitive principles of multimedia learning: The role of modality and contiguity.," *Journal of educational psychology*, vol. 91, no. 2, p. 358, 1999.

[226] P. Chandler and J. Sweller, "The split-attention effect as a factor in the design of instruction," *British Journal of Educational Psychology*, vol. 62, no. 2, pp. 233–246, 1992.

[227] R. A. Tarmizi and J. Sweller, "Guidance during mathematical problem solving.," *Journal of educational psychology*, vol. 80, no. 4, p. 424, 1988.

[228] R. Moreno and R. E. Mayer, "Engaging students in active learning: The case for personalized multimedia messages.," *Journal of educational psychology*, vol. 92, no. 4, p. 724, 2000.

[229] R. E. Mayer, R. Moreno, M. Boire, and S. Vagge, "Maximizing constructivist learning from multimedia communications by minimizing cognitive load.," *Journal of educational psychology*, vol. 91, no. 4, p. 638, 1999.

[230] P. Baggett, "Role of temporal overlap of visual and auditory material in forming dual media associations.," *Journal of Educational Psychology*, vol. 76, no. 3, p. 408, 1984.

[231] P. Baggett, "6 understanding visual and verbal messages," in *Advances in psychology*, vol. 58, pp. 101–124, Elsevier, 1989.

[232] P. Baggett and A. Ehrenfeucht, "Encoding and retaining information in the visuals and verbals of an educational movie," *ECTJ*, vol. 31, no. 1, pp. 23–32, 1983.

[233] S. Y. Mousavi, R. Low, and J. Sweller, "Reducing cognitive load by mixing auditory and visual presentation modes.," *Journal of educational psychology*, vol. 87, no. 2, p. 319, 1995.

[234] S. F. Harp and R. E. Mayer, "The role of interest in learning from scientific text and illustrations: On the distinction between emotional interest and cognitive interest.," *Journal of educational psychology*, vol. 89, no. 1, p. 92, 1997.

[235] S. F. Harp and R. E. Mayer, "How seductive details do their damage: A theory of cognitive interest in science learning.," *Journal of educational psychology*, vol. 90, no. 3, p. 414, 1998.

[236] R. E. Mayer, J. Heiser, and S. Lonn, "Cognitive constraints on multimedia learning: When presenting more material results in less understanding.," *Journal of educational psychology*, vol. 93, no. 1, p. 187, 2001.

[237] S. Kalyuga, P. Chandler, and J. Sweller, "Managing split-attention and redundancy in multimedia instruction," *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, vol. 13, no. 4, pp. 351–371, 1999.

[238] J. B. Carroll *et al.*, *Human cognitive abilities: A survey of factor-analytic studies*. No. 1, Cambridge University Press, 1993.

[239] R. C. Clark and R. E. Mayer, *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. john Wiley & sons, 2016.

[240] R. E. Mayer, "Using multimedia for e-learning," *Journal of Computer Assisted Learning*, vol. 33, no. 5, pp. 403–423, 2017.

[241] R. E. Mayer and P. Chandler, "When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages?," *Journal of educational psychology*, vol. 93, no. 2, p. 390, 2001.

[242] R. Moreno and R. Mayer, "Interactive multimodal learning environments," *Educational psychology review*, vol. 19, no. 3, pp. 309–326, 2007.

[243] R. E. Mayer, G. T. Dow, and S. Mayer, "Multimedia learning in an interactive self-explaining environment: What works in the design of agent-based microworlds?," *Journal of educational psychology*, vol. 95, no. 4, p. 806, 2003.

[244] S. Guttormsen Schär and P. G. Zimmermann, "Investigating means to reduce cognitive load from animations: Applying differentiated measures of knowledge representation," *Journal of Research on Technology in Education*, vol. 40, no. 1, pp. 64–78, 2007.

[245] J.-M. Boucheix and E. Schneider, "Static and animated presentations in learning dynamic mechanical systems," *Learning and instruction*, vol. 19, no. 2, pp. 112–127, 2009.

[246] B. S. Hasler, B. Kersten, and J. Sweller, "Learner control, cognitive load and instructional animation," *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, vol. 21, no. 6, pp. 713–729, 2007.

[247] H. Hassanabadi, E. S. Robatjazi, and A. P. Savoji, "Cognitive consequences of segmentation and modality methods in learning from instructional animations," *Procedia-Social and Behavioral Sciences*, vol. 30, pp. 1481–1487, 2011.

[248] D. L. Lusk, A. D. Evans, T. R. Jeffrey, K. R. Palmer, C. S. Wikstrom, and P. E. Doolittle, "Multimedia learning and individual differences: Mediating the effects of working memory capacity with segmentation," *British Journal of Educational Technology*, vol. 40, no. 4, pp. 636–651, 2009.

[249] K. D. Stiller, A. Freitag, P. Zinnbauer, and C. Freitag, "How pacing of multimedia instructions can influence modality effects: A case of superiority of visual texts," *Australasian Journal of Educational Technology*, vol. 25, no. 2, 2009.

[250] P. D. Mautone and R. E. Mayer, "Signaling as a cognitive guide in multimedia learning.," *Journal of educational Psychology*, vol. 93, no. 2, p. 377, 2001.

[251] P. D. Mautone and R. E. Mayer, "Cognitive aids for guiding graph comprehension.," *Journal of Educational Psychology*, vol. 99, no. 3, p. 640, 2007.

[252] F. Amadieu, C. Mariné, and C. Laimay, "The attention-guiding effect and cognitive load in the comprehension of animations," *Computers in Human Behavior*, vol. 27, no. 1, pp. 36–40, 2011.

[253] J.-M. Boucheix, R. K. Lowe, D. K. Putri, and J. Groff, "Cueing animations: Dynamic signaling aids information extraction and comprehension," *Learning and Instruction*, vol. 25, pp. 71–84, 2013.

[254] B. B. De Koning, H. K. Tabbers, R. M. Rikers, and F. Paas, "Attention cueing as a means to enhance learning from an animation," *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, vol. 21, no. 6, pp. 731–746, 2007.

[255] E. Jamet, M. Gavota, and C. Quaireau, "Attention guiding in multimedia learning," *Learning and instruction*, vol. 18, no. 2, pp. 135–145, 2008.

[256] S. Kriz and M. Hegarty, "Top-down and bottom-up influences on learning from animations," *International Journal of Human-Computer Studies*, vol. 65, no. 11, pp. 911–930, 2007.

[257] J. Naumann, T. Richter, J. Flender, U. Christmann, and N. Groeben, "Signaling in expository hypertexts compensates for deficits in reading skill.," *Journal of Educational Psychology*, vol. 99, no. 4, p. 791, 2007.

[258] E. Ozcelik, I. Arslan-Ari, and K. Cagiltay, "Why does signaling enhance multimedia learning? evidence from eye movements," *Computers in human behavior*, vol. 26, no. 1, pp. 110–117, 2010.

[259] G. D. Rey, "Reading direction and signaling in a simple computer simulation," *Computers in Human Behavior*, vol. 26, no. 5, pp. 1176–1182, 2010.

[260] K. Scheiter and A. Eitel, "The effects of signals on learning from text and diagrams: How looking at diagrams earlier and more frequently improves understanding," in *International Conference on Theory and Application of Diagrams*, pp. 264–270, Springer, 2010.

[261] R. E. Mayer, A. Mathias, and K. Wetzell, "Fostering understanding of multimedia messages through pre-training: Evidence for a two-stage theory of mental model construction.," *Journal of Experimental Psychology: Applied*, vol. 8, no. 3, p. 147, 2002.

[262] R. E. Mayer, P. Mautone, and W. Prothero, "Pictorial aids for learning by doing in a multimedia geology simulation game.," *Journal of Educational Psychology*, vol. 94, no. 1, p. 171, 2002.

[263] E. Pollock, P. Chandler, and J. Sweller, "Assimilating complex information," *Learning and instruction*, vol. 12, no. 1, pp. 61–86, 2002.

[264] A. Eitel, K. Scheiter, and A. Schueler, "How inspecting a picture affects processing of text in multimedia learning," *Applied Cognitive Psychology*, vol. 27, no. 4, pp. 451–461, 2013.

[265] L. Kester, P. A. Kirschner, and J. J. Van Merriënboer, "Timing of information presentation in learning statistics," *Instructional Science*, vol. 32, no. 3, pp. 233–252, 2004.

[266] L. Kester, P. A. Kirschner, and J. J. Van Merriënboer, "Information presentation and troubleshooting in electrical circuits," *International Journal of Science Education*, vol. 26, no. 2, pp. 239–256, 2004.

[267] L. Kester, C. Lehnen, P. W. Van Gerven, and P. A. Kirschner, "Just-in-time, schematic supportive information presentation during cognitive skill acquisition," *Computers in Human Behavior*, vol. 22, no. 1, pp. 93–112, 2006.

[268] L. Kester, P. A. Kirschner, and J. J. van Merriënboer, "Just-in-time information presentation: Improving learning a troubleshooting skill," *Contemporary Educational Psychology*, vol. 31, no. 2, pp. 167–185, 2006.

[269] I. L. Beck, M. G. McKeown, C. Sandora, L. Kucan, and J. Worthy, "Questioning the author: A yearlong classroom implementation to engage students with text," *The Elementary School Journal*, vol. 96, no. 4, pp. 385–414, 1996.

[270] R. Moreno and R. E. Mayer, "Personalized messages that promote science learning in virtual environments.," *Journal of educational Psychology*, vol. 96, no. 1, p. 165, 2004.

[271] B. M. McLaren, K. E. DeLeeuw, and R. E. Mayer, "A politeness effect in learning with web-based intelligent tutors," *International Journal of Human-Computer Studies*, vol. 69, no. 1-2, pp. 70–79, 2011.

[272] N. Wang, W. L. Johnson, R. E. Mayer, P. Rizzo, E. Shaw, and H. Collins, "The politeness effect: Pedagogical agents and learning outcomes," *International journal of human-computer studies*, vol. 66, no. 2, pp. 98–112, 2008.

[273] N. Wang, W. L. Johnson, R. E. Mayer, P. Rizzo, E. Shaw, and H. Collins, "The politeness effect: Pedagogical agents and learning gains.," in *AIED*, pp. 686–693, 2005.

[274] R. E. Mayer, S. Fennell, L. Farmer, and J. Campbell, "A personalization effect in multimedia learning: Students learn better when words are in conversational style rather than formal style.," *Journal of educational psychology*, vol. 96, no. 2, p. 389, 2004.

[275] R. K. Atkinson, R. E. Mayer, and M. M. Merrill, "Fostering social agency in multimedia learning: Examining the impact of an animated agent's voice," *Contemporary Educational Psychology*, vol. 30, no. 1, pp. 117–139, 2005.

[276] R. E. Mayer and C. S. DaPra, "An embodiment effect in computer-based learning with animated pedagogical agents.," *Journal of Experimental Psychology: Applied*, vol. 18, no. 3, p. 239, 2012.

[277] R. E. Mayer, K. Sobko, and P. D. Mautone, "Social cues in multimedia learning: Role of speaker's voice.," *Journal of educational Psychology*, vol. 95, no. 2, p. 419, 2003.

[278] C. I. Nass and S. Brave, *Wired for speech: How voice activates and advances the human-computer relationship*. MIT press Cambridge, MA, 2005.

[279] L. Fiorella and R. E. Mayer, "Effects of observing the instructor draw diagrams on learning from multimedia messages.," *Journal of Educational Psychology*, vol. 108, no. 4, p. 528, 2016.

[280] A. L. Baylor and S. Kim, "Designing nonverbal communication for pedagogical agents: When less is more," *Computers in Human Behavior*, vol. 25, no. 2, pp. 450–457, 2009.

[281] Q. Dunsworth and R. K. Atkinson, "Fostering multimedia learning of science: Exploring the role of an animated agent's image," *Computers & Education*, vol. 49, no. 3, pp. 677–690, 2007.

[282] C. Frechette and R. Moreno, "The roles of animated pedagogical agents' presence and nonverbal communication in multimedia learning environments," *Journal of Media Psychology*, 2010.

[283] R. Moreno, M. Reislein, and G. Ozogul, "Using virtual peers to guide visual attention during learning: A test of the persona hypothesis.," *Journal of Media Psychology: Theories, Methods, and Applications*, vol. 22, no. 2, p. 52, 2010.

[284] D. G. Bobrow, "Qualitative reasoning about physical systems, daniel g. bobrow, ed," 1985.

[285] M. Gellevij, H. Van Der Meij, T. De Jong, and J. Pieters, "Multimodal versus unimodal instruction in a complex learning context," *The Journal of Experimental Education*, vol. 70, no. 3, pp. 215–239, 2002.

[286] D. C. Bui and M. A. McDaniel, "Enhancing learning during lecture note-taking using outlines and illustrative diagrams," *Journal of Applied Research in Memory and Cognition*, vol. 4, no. 2, pp. 129–135, 2015.

[287] M. A. Gernsbacher and K. R. Varner, "The multi-media comprehension battery," tech. rep., Tech. Rep, 1988.

[288] M. A. Gernsbacher, *Language comprehension as structure building*. Psychology Press, 2013.

[289] M. A. McDaniel, R. J. Hines, and M. J. Guynn, "When text difficulty benefits less-skilled readers," *Journal of Memory and Language*, vol. 46, no. 3, pp. 544–561, 2002.

[290] K. R. Popper, "Science as falsification," *Conjectures and refutations*, vol. 1, no. 1963, pp. 33–39, 1963.

[291] G. Gigerenzer, "Surrogates for theories," *Theory & Psychology*, vol. 8, no. 2, pp. 195–204, 1998.

[292] P. E. Smaldino, "Not even wrong: Imprecision perpetuates the illusion of understanding at the cost of actual understanding," *Behavioral and Brain Sciences*, vol. 39, p. e163, 2016.

[293] J. M. Epstein, "Why model?," *Journal of artificial societies and social simulation*, vol. 11, no. 4, p. 12, 2008.

[294] W. C. Wimsatt, "False models as means to truer theories," *Neutral models in biology*, pp. 23–55, 1987.

[295] P. E. Smaldino, J. Calanchini, and C. L. Pickett, "Theory development with agent-based models," *Organizational Psychology Review*, vol. 5, no. 4, pp. 300–317, 2015.

[296] J. C. Schank, C. J. May, and S. S. Joshi, "Models as scaffold for understanding," *Developing scaffolds in evolution, culture, and cognition*, pp. 147–167, 2014.

[297] T. Dragon and P. E. Dickson, "Memory diagrams: A consistant approach across concepts and languages," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, (New York, NY, USA), pp. 546–551, ACM, 2016.

[298] J. Heiser and B. Tversky, "Arrows in comprehending and producing mechanical diagrams," *Cognitive science*, vol. 30, no. 3, pp. 581–592, 2006.

[299] B. Tversky, J. B. Morrison, and M. Betrancourt, "Animation: can it facilitate?," *International journal of human-computer studies*, vol. 57, no. 4, pp. 247–262, 2002.

[300] O.-c. Park and S. S. Gittelman, "Selective use of animation and feedback in computer-based instruction," *Educational Technology Research and Development*, vol. 40, no. 4, pp. 27–38, 1992.

[301] L. P. Rieber, "Using computer animated graphics in science instruction with children.," *Journal of educational psychology*, vol. 82, no. 1, p. 135, 1990.

[302] A. Large, J. Beheshti, A. Breuleux, and A. Renaud, "Effect of animation in enhancing descriptive and procedural texts in a multimedia learning environment," *Journal of the American Society for Information Science*, vol. 47, no. 6, pp. 437–448, 1996.

[303] L. P. Rieber and M. J. Hannafin, "Effects of textual and animated orienting activities and practice on learning from computer-based instruction," *Computers in the Schools*, vol. 5, no. 1-2, pp. 77–90, 1988.

[304] W. Schnotz, J. Böckheler, and H. Grzondziel, "Individual and co-operative learning with interactive animated pictures," *European journal of psychology of education*, vol. 14, no. 2, pp. 245–265, 1999.

[305] J. B. Morrison and B. Tversky, "The (in) effectiveness of animation in instruction," in *CHI'01 extended abstracts on Human factors in computing systems*, pp. 377–378, 2001.

[306] M. D. Byrne, R. Catrambone, and J. T. Stasko, "Evaluating animations as student aids in learning computer algorithms," *Computers & education*, vol. 33, no. 4, pp. 253–278, 1999.

[307] J. F. Pane, A. T. Corbett, and B. E. John, "Assessing dynamics in computer-based instruction," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 197–204, 1996.

[308] R. K. Lowe, "Extracting information from an animation during complex visual learning," *European journal of psychology of education*, vol. 14, no. 2, pp. 225–244, 1999.

[309] T. Slocum, S. Yoder, F. Kessler, and R. Sluter, "Maptime: software for exploring spatiotemporal data associated with point locations," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 37, no. 1, pp. 15–32, 2000.

APPENDIX A: THE MENTAL MODEL TEST OF ARRAYS (MMT-A)

Preamble: This preamble was put in the Qualtrics survey form to collect responses from the participants:

> Please answer to the best of your abilities. We acknowledge that you do not know many of the answers. Try to use your intuition to guess the correct answer with your existing knowledge. Please do not refer to any other information (books, web pages) or execute the statements on a computer. You will not receive or lose points based on your correct or incorrect responses.

Questionnaire Unfortunately, the full questionnaire of the *MMT-A* is only shared with the committee members and is not available to the public. The test *MMT-A* remains the sole intellectual property of the author of the dissertation. Sharing the whole questionnaire may impact and bias participants involved in future studies.

APPENDIX B: EXPLANATIVE DIAGRAMS OF ARRAYS: A MODEL AND A NOTIONAL MACHINE

B.1 Introduction

To minimize programming misconceptions, researchers [2,26,30] urge CS educators to explicitly teach the notional machine. A notional machine (NM) is a pedagogic device or approach to assist the understanding of programming [2]. Theories surrounding notional machines suggest 1) notional machines should be made available to students early, 2) the design of notional machines should be in line with the cognitive load of the learners, 3) the presentation of the NM should not overload the learners' perceptual and mental processing [30, 202]. Explanative diagrams designed based on the literature on the cognitive psychology of visual instructional methods can represent a notional machine. In this chapter, I describe three theories related to the design of my explanative diagrams of arrays: the theory of notional machine, the cognitive theory of multimedia learning, and the theory of explanative diagrams. Additionally, I outline when and why explanative diagrams are effective and how they can serve the purpose of a model and a notional machine. At the end of this chapter, I propose the model of explanative diagrams of arrays and describe its implication in learning programming.

B.2 Background

B.2.1 Notional Machine

According to the recent ITiCSE working group report [2], the term "notional machine" (NM) arose in the 1970s. At that time, researchers started learning about the psychology of learning programming. The researchers concluded that programming is specifically hard for learning for being abstract in nature [26, 202]. The structure and behavior of programming concepts are hidden under the coding syntax. We teach children abstract symbols of numbers with a concreteness fading framework [203], start-

ing from physical objects to pictorial and then abstract representations. However, programming learners presented with only the abstract code had many difficulties comprehending the hidden machine [26, 202]. The concept of an NM emerged when Du Boulay [26, 202] proposed a pedagogical approach: the glass box approach. With this approach, learners attempt to understand how each command changes the states of the parts of a computer within a relevant abstraction level. Du Boulay [26, 202] later termed this relevant pedagogical abstraction of a computer- a notional machine. The term was buried in the literature for four decades until it became prominent in the mid-2000's [2]. At first, Robins, Rountree, and Rountree's [204] publication reintroduced NM to the computing education research community. Most specifically, Sorva's [34] work on the notional machine and introductory programming attracted many researchers to contemplate the implications of the notional machine.

Definition

Though the term NM got lots of attention in recent (2016-2019) works, researchers began to adopt, refine, and in some cases re-develop [2]. The ITiCSE working group [2] found that papers that used the term NM only under 50% of them defined it accurately. Therefore, the term was circulating in the research community in a confounding manner. To remove this ambiguity, a group of researchers gathered in the Dagstuhl seminar 19281, Notional Machines and Programming Language Semantics in Education, from 7th-12th July 2019 [205]. From there, an ITiCSE working group [2] was formed to capture and characterize the notional machine.

Du Boulay [202] initially defined an NM as an idealized yet simplified conceptual model of a machine one is trying to control. The properties of the NM are not the actual machine (hardware) but the programming language by which a human is trying to instruct the machine. According to Du Boulay [202], the purpose of an NM is to make the machine's parts and processes in action visible, simple, and relevant. Later, Sorva [34] elaborated,

> "A notional machine encompasses capabilities and behaviors of hardware and software that are abstract but sufficiently detailed, for a certain context, to explain how programs get executed and what the relationship of programming language commands is to such executions." [34, p.8:2]

Recently the ITiCSE working group [2], including Du Boulay, formalized the definition as, "a notional machine (NM) is a pedagogic device to assist the understanding of some aspect of programs or programming." [2, p. 22]

Here, the word notion is implying that it's a simplified version of the truth [2]. The complete truth of what happens when a piece of code runs can go from the details of byte code to the machine code, transistors, and so on. The NM is putting necessary layers of abstraction to ignore irrelevant complex details to simplify the truth to the appropriate audience.

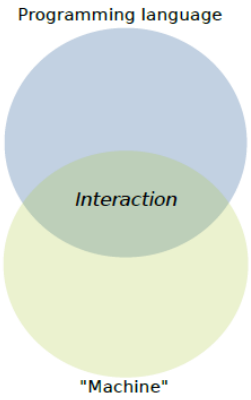


Figure B.1: The scope of the notional machine defined by ITiCSE working group [2].

An NM is called a machine because it makes an explicit analogy to a mechanism consisting of parts that interact to produce a behavior [2], like a switch with clapper, coils, and wires. A piece of code implicitly interacts with the parts of a computer and produces behaviors with the parts' state change. An NM aims to make the implicit events of interactions explicit. As depicted in Figure B.1, an NM focuses on the interaction of a programming language with the actual machine.

Though these definitions give us a sense of what an NM is, they are not complete. To make the definition of an NM more concrete, the ITiCSE working group [2] listed some defining characteristics summarized below with additional synthesis of other scholarly works on the notional machine.

A notional machine is tied to a programming language.

A notional machine is strongly tied to the semantics of a programming language [26, 34, 202]. Therefore, different kinds of programming languages will have distinct notional machines. For example, a notional machine representing an array in Java contains references even though an NM of an array in the programming language C does not have one. Krishnamurthi and Fisler [30], in the Cambridge Handbook, identified notional machines as a tool to classify programming languages.

A notional machine is a simplified conceptual model.

An NM is a model. It is a model of program execution, but not a detailed one [3]. Using a model to teach science is not new. Bohr's atom model [206] or Newton's planetary gravitation model [207] was designed to present complex science in a simplistic way. These models did not include all the details but included the relevant ones, which can offer accurate information in a simplistic, comprehensible way. A detailed model is essential for a designer, but for the one who is learning about a system can get lost in too much detail. As discussed in the Section **??**, a conceptual model is the most detailed and complete model. As there are layers of abstraction, a notional machine is considered a simplified conceptual model [2]. Models and conceptual models articulate parts and states of a system [208] that can answer questions such as: what happens when you press the brake of a car?

A notional machine is a simplified conceptual model [2], and simple models are strong [208]. The layer of abstraction determines the simplicity of the conceptual model. The thicker the layer of abstraction is, the more NM becomes a simpler conceptual model (see Figure B.2). The thinner the layer of abstraction is, the more

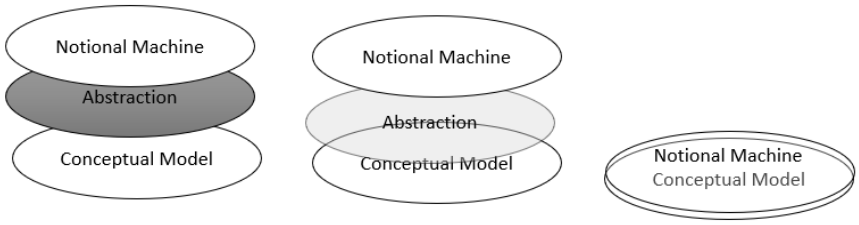


Figure B.2: As the layer of abstraction gets thinner, the NM resembles more of the conceptual model.

NM becomes closer to the complete conceptual model. The teacher or the teaching agent (e.g., textbooks, program visualization tools) determines the abstraction layer's thickness based on their audience. For example, we can explain variables as a chair to the high school students where only one value can fit in. To CS1 majors, we can explain a variable as a memory location where a value is stored with its name as an alias.

A notional machine is a pedagogical tool.

The main aim of a notional machine is to serve as a pedagogical tool [2]. The pedagogical decision determines the level of abstraction of the notional machine. However, the notional machine's purpose will be served when it is designed with simplicity and visibility [202]. As stated by Du Boulay,

> "A conceptually simple notional machine does not necessarily imply either a low-level language (such as a simulated assembler) or a weak high-level language (such as BASIC)." [202, p.239]

An ideal notional machine should have the right amount of detail based on its learners. The function of an NM is to uncover something about programming that is hidden from the students [2]. The visibility of an NM is the "glass box" through which the novice can see the "black boxes" of how programming works [202]. Lack of visibility creates a risk of implicit assumptions by novices, which can later lead to

a misconception. Ben-Ari [46], by gathering evidence from the literature, argues that students will necessarily construct their own knowledge of the notional machine when visibility is not ensured. Consequently, Sorva remarked, "intuitive models of computers are doomed to be nonviable" [34, p. 8:15]. With the right level of detail and explicit information of how a programming concept works, a teacher can eliminate implicit assumptions' risks, thus non-viability.

A notional machine has representations.

A notional machine has a simple and visible representation. This representation will focus on appropriate detail and ignore irrelevant details. A representation of an NM can have many forms [2]. For example, by saying to the class a variable is like a box has a verbal representation. When the same thing is drawn as a diagram to the class, it has a visual representation. Also, two NMs can complement each other. For example, an instructor may create a variable table on a whiteboard and show the variable values in a debugger [2]. More details can also be added to an NM when transitioning towards more advanced concepts. For example, by adding an arrow (indicating a reference) to a box of a primitive variable, we can introduce a reference variable. The ITiCSE working group [2] categorized the NM representations into three groups: Machine-generated representations, Handmade representations, and Analogy.

The Interplay of the Notional Machine and Mental Models

The notional machine is an abstracted conceptual model, consistently accurate. However, novice programmers' mental models are idiosyncratic, incomplete, and often inaccurate [34]. A novice programmer does not form the mental model of the actual machine; rather they form the mental model of the notional machine presented with [2]. When the notional machine is completely transferred to a mental model, the mental model becomes accurate and consistent. The portions of the mental model not formed by a notional machine, rather formed intuitively, create the

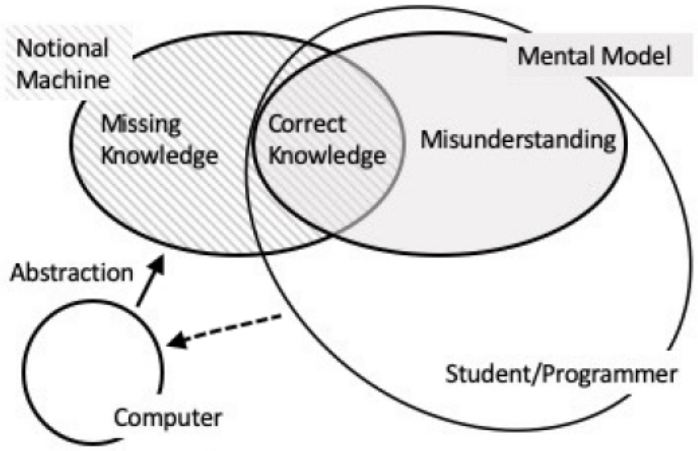


Figure B.3: Interaction between a notional machine, mental model, and the programming behavior presented in [3].

opportunities of misconceptions [3] (see Figure B.3). To be more precise, a notional machine is not a mental representation or 'notion' of someone. Figure B.4 states that a notional machine is a kind of conceptual model, simplified, abstracted, and analogous. Learners form a mental model of the notional machine.

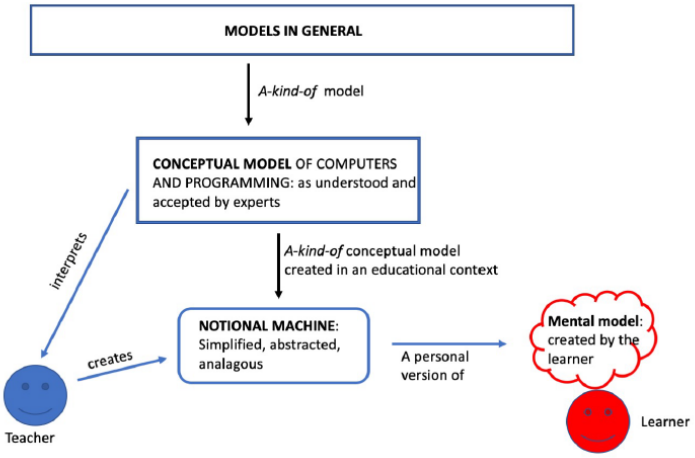


Figure B.4: The interplay between the NM and the mental model from [2].

I believe the cognitive theory of multimedia learning can provide design guidelines for notional machines. Below, I describe the key aspects of Mayer's cognitive theory

of multimedia learning.

B.2.2 Mayer's Cognitive Theory of Multimedia Learning

The cognitive theory of multimedia learning (CTML) [209] is about how people learn from words and pictures based on consistent empirical evidence (e.g., [210–212]) and established on the principles of cognitive science (e.g., [211, 213, 214]). The hypothesis underlying this theory is that:

"Multimedia instructional messages that are designed in light of how the human mind works are more likely to lead to meaningful learning than those that are not" [4, p.32].

The cognitive theory of multimedia learning derived from cognitive science internalizes three assumptions, utilizes three memory stores, and describes learning as five processes.

The Three Assumptions of the CTML

Most of the time, multimedia messages are designed based on the designers' conception of how the human mind works. For instance, consider an educational multimedia message of teaching alphabet to a child- letters flashing with overflowing numbers of colors. This video reflects that the designer knows that humans possess a single channel, unlimited capacity, and a passive processing system. However, researchers on how the human mind works believe in different assumptions [211, 213, 214]. These assumptions are also the basis of the cognitive theory of multimedia learning. The assumptions follow:

Dual-Channel Assumption

The dual-channel assumption has a long history in cognitive psychology and is most closely related to Paivio's [215] dual-coding theory. This assumption refers to the fact that humans process words and images separately with two distinct channels. Based on this assumption, the cognitive theory of multimedia learning proposes that

the human information processing system contains an auditory/verbal channel and a visual/pictorial channel, which is most consistent with Baddeley's [216, 217] view. Therefore, learners process the presented information with two sensory memories: 1) eyes (e.g., for pictures, video, animation, or printed words) and 2) ears (e.g., for spoken words or background sound). Although this assumption believes in two separate sensory channels, Mayer [4] believed learners might also perform cross-channel representations. Paivio's [215] dual coding theory introduced the cross-channel representations of the same stimulus. Cross-channel representation refers to the fact that humans can absorb information in one form and can transform the information modality in their minds. For example, though you hear the narration describing that "when the lever is pulled, the weight instantly falls," you are also creating a mental visual image of someone pulling the lever and the weight falling off. Conversely, an experienced reader presented with a printed text may initially process the information with the visual channel. However, after processing, the reader may mentally convert the text into a narrative (i.e., sound).

Limited Capacity Assumption

The second assumption is that humans cannot process unlimited information; rather, humans process information in portions with a limited capacity. For example, when you read this text, only some portions of this document are in your working memory, not the entire text you read. Similarly, when a learner is presented with a long narration, they would only remember some portions of the narration, not the verbatim recording. This assumption is supported by the theory of working memory by Baddley's [216, 217] and the cognitive load theory by Sweller and Chandler [218].

Active Processing Assumption

The cognitive theory of multimedia learning also believes that learners actively construct knowledge, not passively [4]. Active learning occurs by selecting relevant information, organizing selected information, and integrating selected material with

existing knowledge [210, 218, 219]. CTML [4] denies the passive learning view that the human information system is like a tape recorder, adding every single piece of information as much as possible. By engaging in active learning, learners engage in cognitive processes while processing a piece of information and constructing a coherent mental model. Mayer describes a mental model in his theory of multimedia learning as: "A mental model (or knowledge structure) represents the key parts of the presented material, and their relations" [4, p.36]. He further explains that a multimedia presentation explaining some phenomena can help the learner develop the cause-and-effect system and understand how a change in one part causes a change in another part. Active processing assumptions advocate two implications [4] for multimedia design: 1) multimedia material should have a coherent structure, not a collection of isolated facts, and 2) the material should guide the learner on how to structure the presented material.

A Cognitive Integrated Model of Text and Picture

CTML [4] proposes a cognitive model of multimedia learning based on the above three assumptions about the human information processing system. This integrated model is based on multiple memory systems [220], working memory [216, 221], and dual coding [215]. CTML describes five cognitive processes in three kinds of memories: sensory memory/registers, short-term memory/working memory, and long-term memory. Figure B.5 represents a cognitive model of how humans process an integrated model of text and pictures. Mayer [4] argues that to achieve meaningful learning, a learner must engage in five cognitive processes: 1) selecting relevant words from the text, 2) selecting relevant images from the pictures, 3) organizing selected words into a verbal model, 4) organizing selected images into a pictorial model, and 5) integrating the verbal and pictorial presentations and with prior knowledge. Below, I summarize each of the five processes:

Selecting Relevant Words

Words can come from the outside world by spoken words (e.g., computer-generated narration) or written text. Based on Mayer's [4] model, depending on the medium, words can enter the sensory memory through either ear (for narration) or eyes (for printed text). According to Schnotz [222], people have multiple sensory channels based on multiple sensory modalities between the outside world and working memory. For the integrated model of words and pictures [4], only two sensory channels are considered: visual and auditory. According to Schnotsz [222], the visual channel transfers information from the eye to the visual working memory. Similarly, the auditory channel transfers information from the ear to the auditive working memory. Schnotz [222] claims that information is stored for a concise amount of time in visual (i.e., less than 1 seconds) and auditory registers (i.e., less than 3 seconds). According to Figure B.5, in selecting relevant words, learners pay attention to a sentence or speech's pertinent words. When the selected words are passed as input to the sensory memory, it generates a sound base's output (sounds in Figure B.5). This sound base is a mental representation of the selected words or phrases. When the outside media is printed text, the words reach the eyes. As discussed in the dual-channel assumption, a learner can form a cross-channel representation. Thus, while processing the visual words, a learner can transform them into sounds. In the same vein, Schnotz [222], relying on the theory of mental models, further explains this step. According to him, when a learner listens to a text or reads a text, they create a text-surface representation. For example, when reading a statement like this: "Clouds form when the invisible water vapor in the air condenses into visible water droplets or ice crystals," a learner pays attention to the relevant keywords: *cloud, form, vapor, condenses, droplets*. These relevant keywords form the text-surface representation. This text-surface representation is not meaningful yet; instead, it extracts the read's relevant words.

Organizing Selected Words

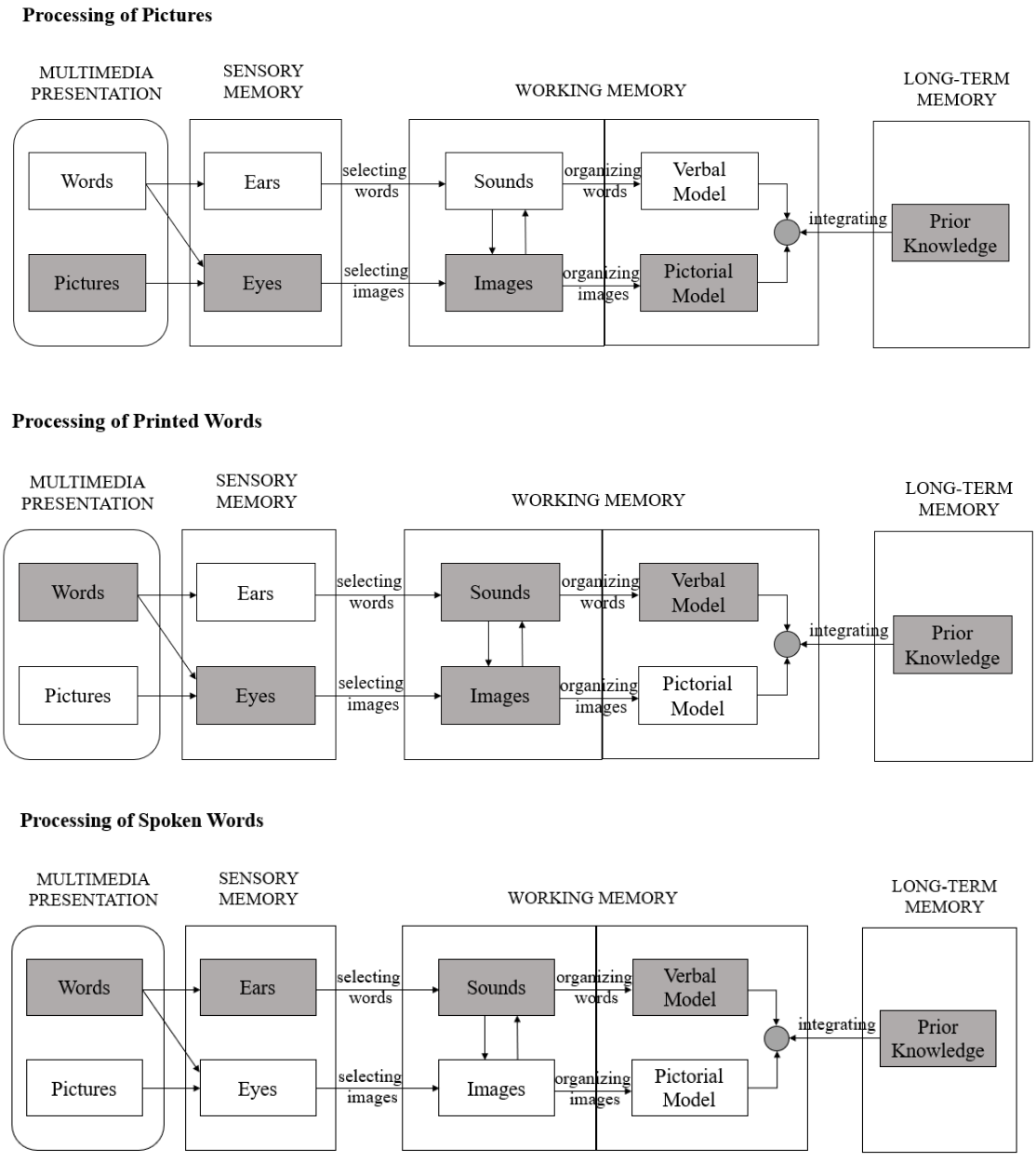


Figure B.5: How human mind processes pictures, printed words, and spoken words from Mayer's CTML [4]

.

Furthermore, Mayer [4] explains that when a learner has formed the sound base, they start to organize the selected words to form a knowledge structure. Mayer termed this structure as a verbal model. Figure B.5 shows that the input here is the word-sound base; after organization, it transforms into a coherent verbal model. According to Schnotz [222], the representation of this model is propositional. This representation does not include verbatim wording; rather, it includes the idea expressed in the text on a conceptual level, like a cause-and-effect chain. For the previous example of cloud formation, the propositional representation can be: FORM(clouds) if CONDENSE(vapor $\rightarrow$ ice crystals). This formation of the concept is performed in the auditory working memory. Baddley [216] reported that auditory and visual working memory are the two subsystems of working memory along with the central executive systems. As working memory has limited capacity, these two subsystems can store limited information for a shorter period.

Selecting Relevant Images

Similar to selecting words, when a learner is presented with the external representation of pictures (e.g., an animation or an illustration), information enters through the eyes, and concise, relevant portions of the visual representation get stored in the visual register [4]. Here, in Figure B.5, the box 'images' represents the relevant images. Similar to the sound base, learners create an image base from the selected images. The image base is sensory-specific as it is tied to the visual modality. Due to the limited capacity of human information processing systems, the learner cannot store all the details of a complex illustration or animation. Hence, a learner can only focus on the key parts of the pictorial information, helping the learner in sense-making. As Mayer [4] states, the input of this process here is the frames from the external multimedia message, and the output is the visual image base (referred to images in Figure B.5). For further processing, the selected images are transferred to the working memory by visual channel.

Organizing Selected Images

The process of organizing selected images corresponds to the process of organizing the selected words. After forming the image base, the next step is to organize the images to form a coherent pictorial model [4]. The pictorial model is a structured visual representation in the learner's working memory of the image base. Mayer [4] calls this step organizing selected images (see Figure B.5). In this step, learners make connections among segments of pictorial information and try to make sense of it. Most importantly, learners can only build the simple set of connections that can enable them to build a cause-and-effect structure. In short, with this step, a learner can form causal links by synthesizing the pictures. This process happens in the visual channel of the working memory.

Integrating Word-Based and Image-Based Representations

According to Mayer [4], the process of integrating word-based representations with an image-based representation is an embodiment of sense-making. When information from the dual channels integrates, the two representations transform into one coherent representation. In this process, the verbal and pictorial models' elements and relations are mapped onto the other (see Figure B.5). Here, Mayer [4] takes the constructivist approach and believes that prior experience is also included in the integration process. Mayer [4] termed this process integration as it involves making connections between the pictorial and verbal model and prior knowledge residing in long-term memory. As Mayer [4] and Schnotz [222] stated, the integration process occurs in the working memory. For example, in the previous example above, while reading the causal chain of cloud formation, a learner links this information with the illustration that shows arrows and transitions of water in another form. In short, both the verbal and pictorial mediums complement each other and overcome each channel's limitations. This process is also visible in our regular life. For example, when we see an illustration but do not know the written language, we try to connect

the written language and the picture. As an observation, when a child reads a book filled with illustrations alongside the texts, it can interpret the text's meaning by seeing the pictures even if it can not read.

Though the five processes summarize the cognitive activity, the order is not implied here. One can think of these five processes happening in the human mind sequentially in the order written above, but this is not the case. As stated by Mayer [4], each of these five processes is likely to occur multiple times in multiple forms during a learner's interaction with the multimedia presentation. Also, it is a segmented process; it does not occur as a whole [4]. Learners go through these cognitive processes segment by segment: they may select some relevant words to form the first sentence of narration and then focus on the images, move back the focus to narration, and then integrate the information. The process does not necessarily follow an order and, for certain, does not happen as a whole. The learners do not finish reading all the texts, watch all of the pictures, and finally integrate. The five processes are simultaneous and spontaneous cognitive processes.

Integrating words with pictures does not always generate meaningful learning. Unlike dual-coding theory [215], the theory of multimedia learning offers principles that consider the negative effects of integrating words and pictures. Inculcating the three assumptions and five processes, Mayer [4] offers empirically established design principles on integrating words and pictures to generate effective, meaningful learning.

Principles of Cognitive Theory of Multimedia Learning

By conducting empirical studies, Mayer laid out twelve principles of multimedia learning. Below, I summarize the principles. **Principle 1: Spatial Contiguity Principle**

An integrated model of text and pictures is more effective for learning when corresponding pictures and words are presented near rather than far from each other on a page or screen [210]. The rationale behind this principle is that if placed to-

gether, the learners do not need to use their cognitive resources to search the page or screen visually. Also, by viewing the corresponding texts and pictures together, learners are more likely to hold them in working memory simultaneously. With five experiments of Mayer's, he demonstrated that if corresponding texts and pictures are placed together, learners performed better in recall and knowledge transfer for both book-based [223,224] and computer-based environments [225]. Other researchers also found the spatial contiguity effect under the name of Sweller's split-attention effect [218, 226, 227].

Principle 2: Temporal Contiguity Principle

As the spatial contiguity effect discusses texts and pictures' placement in terms of space, temporal contiguity discusses the same issue for time. According to this principle, learners learn better when corresponding words and pictures are presented concurrently rather than successively. It can seem that spatial and temporal contiguity are identical, but they are not. Spatial contiguity offers suggestions for the layout of materials processed by the eyes (such as a book page with words and illustrations). In contrast, temporal contiguity is important for the layout of the materials processed by the eyes and ears (e.g., an animation with narration). Mayer [228, 229] proved that simultaneous presentation is better for retention and transfer tests than successive presentation when the successive presentation is not small. Mayer based his work on Bagget, and her colleagues [230–232] previous works on temporal contiguity. They found that when students viewed films with voice overlay, they performed less in recall tasks. Sweller [233] included temporal contiguity as a part of the split-attention effect.

Principle 3: Coherence Principle

This principle deals with learning outcomes when extraneous materials are included. This coherence principle [210] offers three implications:

1. Students' learning outcomes struggle when interesting yet irrelevant words and

pictures are added.

2. Students' learning outcomes struggle when interesting yet extraneous sounds and music are added to multimedia presentations.

3. Students' learning outcomes improve when unnecessary words are removed from a multimedia presentation.

This principle's rationale is that extraneous materials compete for cognitive resources in working memory and divert attention from important, relevant information. In eleven of eleven experiments [234–236], learners who received multimedia presentations without extraneous material performed significantly better on retention and transfer tests.

Principle 4: Modality Principle

Does modality matter if you present words and pictures? Yes. Modality principle suggests that learners learn better when words in multimedia are presented as spoken words rather than printed text [210]. In four out of four experiments, Mayer [225,229] found that participants who received animation with narration performed better than those who received animation with printed text in the recall and transfer test. These findings have corroborated with a similar finding in previous research [233].

Principle 5: Redundancy Principle

If narration and animation are better, then narration, animation, and text can be the best. But this is not the case. Humans have a limited capacity in working memory. Therefore, adding information in more than two forms (e.g., narration, animation, printed text) will overload working memory channels [210]. Experiments surrounding the redundancy principle proved that learners who received animation with narration performed better than those who received animation, narration, and text [236]. In the same vein, a previous work led by Kalyuga, Chandler and Sweller [237] gave each group of participants diagrams with text, diagrams with narration, and diagrams

with both narration and text. The results showed the redundancy effect. Participants who learned from diagrams accompanied by narration learned better than those who learned from diagrams accompanied by narration and text.

Principle 6: Individual Difference Principle

Even if all of the multimedia design principles are retained in a multimedia message, it can not be effective for all learners. As constructivism [11] (see section **??**) says, learners' prior knowledge acts as a learning catalyst. Multimedia learning principles also account for individual differences [210]. The individual difference principle states that multimedia design effects have a difference among low-knowledge, low-spatial learners and high-knowledge, high-spatial learners. With this principle, Mayer [210] specifies for whom the multimedia design effects will be effective. When considering knowledge as the main effect, in two experiments [5], multimedia design implementation caused a higher impact on retention when learners' knowledge was low rather than high. Similar results were found in four experiments led by Mayer to measure learners' transfer skills [160]. Higher transfer skill was found in learners who also had high spatial ability. Spatial ability is the human mind's ability to mentally generate, maintain, and manipulate visual images [238].

Principle 7: Segmenting Principle

People learn more when multimedia is segmented and broken down into chunks. Mayer [239,240] suggested that to manage complexities, a multimedia message should be segmented. The procedure to do this is to tally the number of elements in a concept that the multimedia message is presenting and their interactions. Then, divide the material into that many segments. Without segmenting, a learner who is unfamiliar with the topic gets overwhelmed with too much information, which surpasses the learner's cognitive capacity. This principle was empirically validated by conducting ten experimental comparisons by Mayer [241–243] and other researchers [244]. The results proved that students who learned with segmented lessons [241–243, 245–249]

that involving computer-based multimedia lessons on lightning, electric motors, pulley systems, how the the human eye works, astronomy, and history performed better on comprehension or transfer tests than students who learned with continuous lessons covering the identical material.

Principle 8: Signaling Principle

Signaling, often known as visual cues, helps a learner focus on a particular portion by highlighting text and pictures with coloring, spotlights, or arrows. Mayer's [240] signaling principle states: "people learn better from a computer-based multimedia lesson when essential parts of text or graphics are highlighted" [240, p. 408]. Results from 18 experiments led by Mayer [242,242,250,251] and other researchers [245, 252–260] on various topics involving computer-based lessons on airplanes, visual perception, geography, mechanical systems, braking systems, cardiovascular system, jet engine, neural networks supported this principle. Like other principles, this principle holds by instilling individual difference principle.

Principle 9: Pre-Training Principle

When learners are taught unfamiliar concepts, complex terms, and their cause-effect relationships can at first overwhelm them. If the learners are pre-trained about the key terms, then they can better comprehend the cause-effect chains [240]. For example, if a learner is learning about how the brake system of a car works, With lots of new information, if the learner keeps listening or reading the terms such as pedal, master cylinder, and piston, it will be hard for him/her to understand the working mechanism properly. By knowing the details of each component's parts (e.g., pedal, master cylinder) before learning the mechanism, the learner will contextualize more and integrate the working mechanism. Three separate studies led by Mayer [261,262] indicated that participants who were pre-trained before the narrated animation of a braking system performed better than those who did not receive any pre-training. In the other two experiments led by Chandler and Sweller [263], electrical engineering

trainees showed how each electrical component worked before the lesson on safety tests for electrical appliances performed better than those who learned each component alongside the lesson. Other researchers also found similar results [264–268].

Principle 10: Personalization Principle

Does the communication tone of information delivery matter? Yes. Learners are more engaged and connect with E-learning material when the communication mode is informal, like a conversation [240]. Previous research [269] on discourse processing revealed that people engage more and try harder to understand the material when conversing. By following this finding, Mayer [228, 270] presented two multimedia materials to the learners on botany. One delivers information in a formal tone (i.e., passive voice, no pronouns). The other offers the same information in an informal way (i.e., like a conversation that uses pronouns such as you I). In five out of five experiments led by Mayer [228, 270–273], participants from the informal material performed better and even made more solutions to a transfer test than the formal material group. Mayer [228, 274] later ran two more experiments on the topic of lightning formation. Learners who received the personalized narrated animation of the lightning lesson performed substantially better on a transfer test than those who received formal narrated animation in two experimental setups.

Principle 11: Voice Principle

Along with the tone of communication, the voice also affects multimedia learning. When multimedia is presented with visuals and sounds, people learn more when the sound is a human voice, not a machine voice [240, 275, 276]. The rationale behind this is the same as the personalization principle. Learners feel more engaged and try to have a deeper understanding when they feel they are communicating with a human. Mayer [277] found that participants learned more from a narrated animation by a human about how lightning forms than from a narrated animation of a machine. Mayer's hypothesis was based on Reeves and Nass's [278] work. Reeves and Nass [278]

proved that under the right conditions, people treat computers like humans. Mayer concluded that more research is needed to comment on narrators' gender and ethnicity in learning [240].

Principle 12: Embodiment Principle

People learn better when an instructor draws while explaining than explaining with an already drawn corresponding diagram [240]. This principle suggests the effect of embodiment in learning. Agents who draw or explain concepts with a gesture, facial expression, and eye gaze are considered high embodied agents [240]. Learners can socially and mentally connect more with a high embodied on-screen agent rather than a low-embodied agent who stands motionless [240]. Mayer [279] asked his participants to watch two types of video lectures on the Doppler effect. In one video, the instructor explained the Doppler effect by standing beside a corresponding diagram implying a low embodied agent. The same instructor was describing the same concept while drawing the corresponding diagram, implying a high embodied agent. Mayer [276,279] and other researchers [280–283] found that in 13 of 14 experimental comparisons on various topics, participants learned better with a high embodied agent than a low embodied agent.

In this section, I presented a detailed overview of multimedia learning. The theory acts as one of the foundational theories as I aim to design the notional machine of arrays as an integrated model of explanative texts and diagrams. Though my design is inspired by the theory of integrated words and pictures, I do not utilize all of the principles in my design. Therefore, some of the design principles are outside the scope of this dissertation. Chapter 3 describes which design principles of CTML I used in designing the notional machine of arrays.

B.2.3 Explanative Diagram

In Section B.2.2, I presented the cognitive theory of multimedia learning, which emphasizes the importance of the integrated model of words of pictures. The princi-

ples mentioned there recommended how to design the integrated model of text and pictures. However, if we now put our focus on the pictures, we also need to know what makes a good illustration and which features of illustrations make them effective. Richard E. Mayer and Joan K. Gallini's [5] work titled "When Is an Illustration Worth Ten Thousand Words?" aims to answer these questions. This empirical research aims to investigate the educationally relevant issues of how diagrams can be designed and used to promote the acquisition of runnable mental models. Their research question included: "When is an illustration most likely to be effective in promoting scientific understanding?" [5, p.716]. Understanding the integrated model of words and pictures of a system means building a mental model from the words and pictures. From the theories of mental model [32, 55–58], Mayer et al. [5] determined two features of a diagram, if retained, can help learners build a runnable mental model: 1) system topology and 2) component behavior.

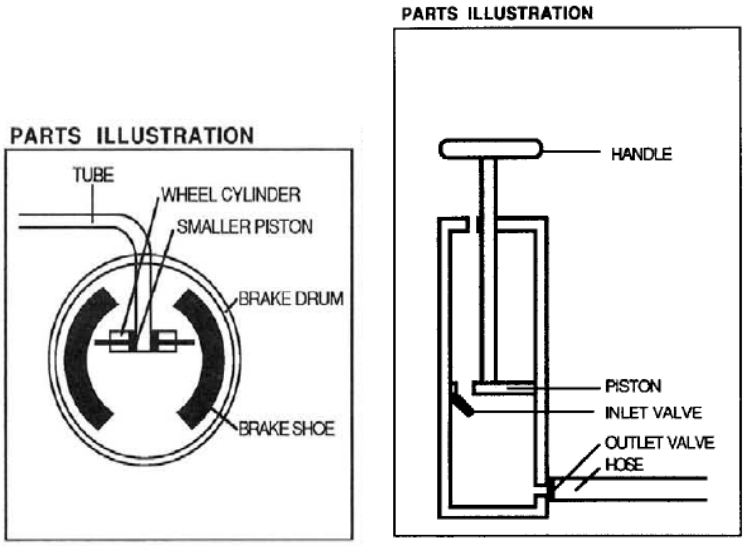


Figure B.6: From left to right: the system topology of a car's brake system and a bicycle pump from [5].

System topology refers to the portrayal of a system's major components or parts, in short, the internal structure of the system [5]. For example, Figure B.6 shows (left)

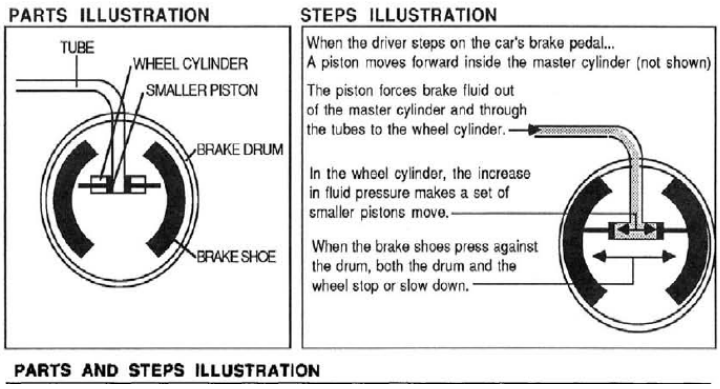


Figure B.7: Component behavior of a car's brake system from [5].

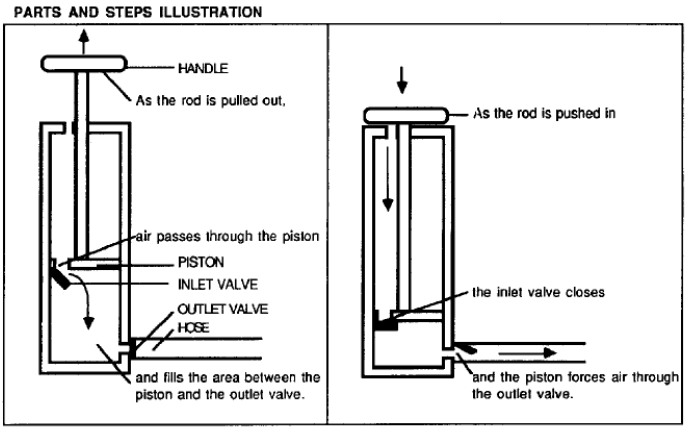


Figure B.8: Component behavior of a bicycle pump from [5]

the system topology of the brake system. Here each major component consisting of a tube, wheel cylinder, smaller piston, brake drum, and brake shoe are illustrated and outlined. Similarly, Figure B.6 (right) shows the system topology of a bicycle pump.

Component behavior refers to the portrayal of each major state that each component has [5]. It also shows how a state change in one component relates to the state changes in other components. Moreover, component behavior portrays the "before" and "after" states of each component. As depicted in Figure B.7 and Figure B.8, the component behavior illustrates parts and steps. Figure B.8 represents the before and after states of a bicycle pump when pulled.

B.3 Effectiveness of Explanative Diagrams in Learning

By conducting two experiments, Mayer [5] proposed a framework (see Figure B.9) for designing a successful instructional method with integrated texts and pictures. The framework includes the type of learner, the type of text, the type of illustration, and the type of performance evaluation. This framework also serves as four conditions that must be met for illustrations to promote understanding of a scientific text effectively.

As discussed in Section B.2.2 multimedia will only be helpful to less knowledgeable learners or novices. Knowledgeable learners already hold a mental model; therefore, offering explanative diagrams seems redundant to them. An integrated model of text and explanative diagrams will only retain its effectiveness when the material is presented to a novice learner [5].

The second condition is that explanative texts must accompany the diagram to promote meaningful learning [5]. Expository text can be descriptive, filled with facts, and also explanative whose purpose is to explain. Specifically, explanative texts aim to explain the cause-and-effect systems allowing qualitative reasoning [284]. In one experiment, Mayer [223] found that the group who received diagrams recalled almost twice as much of the explanative information relative to the non-explanative information

Diagrams in a text can range from irrelevant photography to systematic illustrations. To be effective in developing mental models, diagrams need to be explanatory [5]. Explanative diagram aims to serve the interpretation function [5]. It promotes a reader's comprehension of how a system works. With system topology and component behavior as features, explanative diagrams illustrate the cause-and-effect relationships of the explanative text [5]. Mayer et al. [5] found that explanative diagrams improved recall and problem-solving skills than non-explanative diagrams with three experiments. Moreover, their findings suggest that an explanative diagram im-

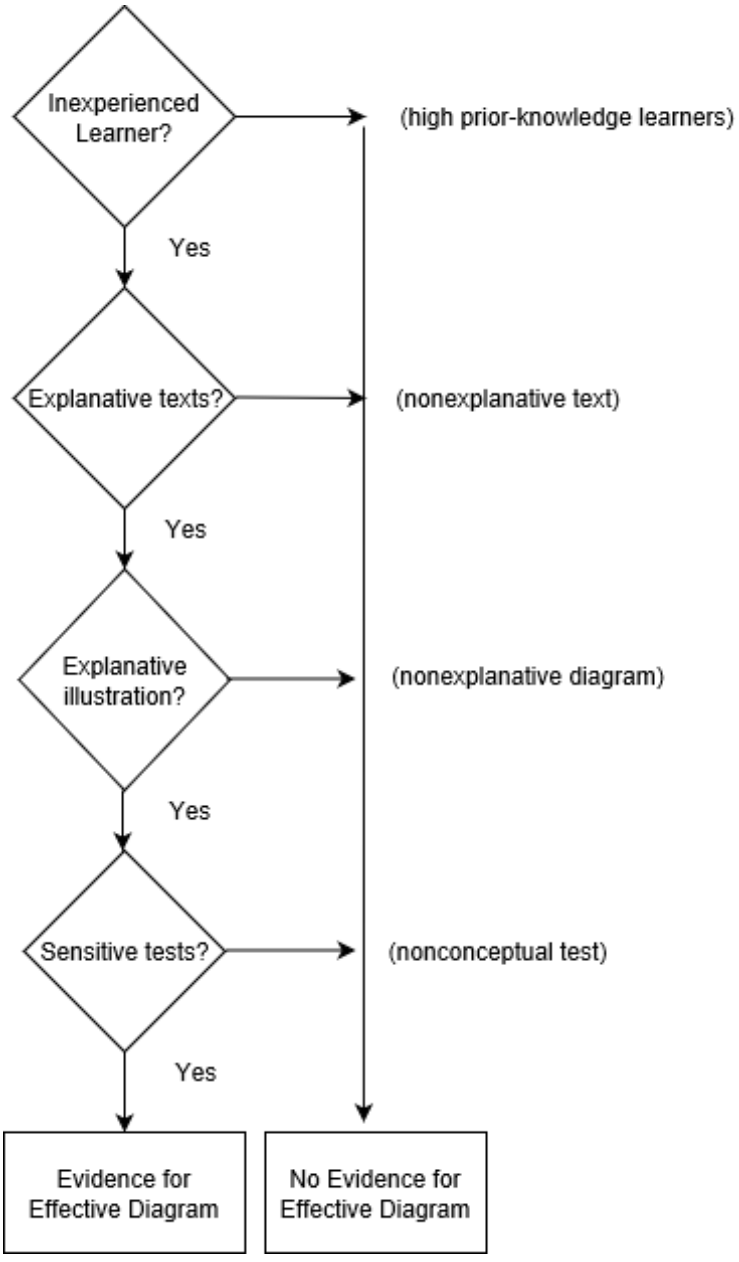


Figure B.9: Four conditions of effective diagrams by Mayer et al. [5].

proves creative problem-solving but not verbatim retention [5]. This improvement was more noticeable in the low knowledge learners than the high knowledge learners.

The effectiveness of diagrams in learning can be measured with sensitive tests- a performance test that measures meaningful learning. Sensitive tests aim to measure learners' understanding and qualitative reasoning about a system [5]. Recall of non-explanative text and verbatim retention measure rote learning, not meaningful learning [5]. Thus, they do not serve as appropriate tests to measure the effectiveness of the explanative diagram. Mayer et al. [5], in their three experiments, used explanative recall and problem-solving tests to measure their diagrams' effectiveness in promoting meaningful learning.

By retaining these four conditions, diagrams in an expository text can entail the power to be worth ten thousand words [5]. Here, diagrams aim to scaffold the learning experience by dissecting a system and placing explicit attention to certain features.

Mayer et al. [5] conducted three separate experiments to validate their four conditions. Ninety-six psychology students from the University of Santa Barbara participated in the first experiment. The participants were equally divided into the four treatment groups: 1) the no diagram group was given a booklet about the braking system with only texts; 2) the parts diagram group's booklet contained a diagram with major parts of the braking system; 3) the steps diagram group depicted major state changes of a braking system, and 4) the parts and steps contained explanative diagrams portraying parts and steps. Mayer et al. [5] equally divided the low and high prior knowledge learners into these four groups. The booklet contained information on how different brake systems work, such as mechanical brakes, hydraulic disk brakes, hydraulic drum brakes, etc. The booklet contained explanative text describing each part's state changes and factual information like the historical description or manufacture information. Mayer et al. [5] used three post-tests to gather evidence for their hypothesis. The first is the recall test, which asks the participants to write

anything about the booklet they can recall. The second test is to measure participants' problem-solving skills. This test asks five open-ended questions such as "why do brakes get hot?" and "what could have gone wrong when your brakes do not work?". The last test measures verbatim retention, which asks to place a checkmark next to the sentence that is a word-for-word match to a sentence in the passage.

In the first experiment [5], low prior learners of the explanative diagram group significantly ($p < .001$) outperformed in recalling explanative information but not non-explanative information. Similar results ($p < .001$) were found while analyzing the problem-solving skill. Low prior-knowledge students who read explanative text with explanative diagrams generated more creative answers to the second post-test than the other groups [5]. However, the explanative diagram group could not outperform others in terms of verbatim retention. No difference was found between the four treatment groups in any post-tests for high prior knowledge learners.

Later, in 2002, Gellevij et al. [285] incorporated Mayer et al.'s [5] four conditions into their instructional material and evaluated its effects. Their aim was to compare the multimodal versus the unimodal instruction in a complex learning context. Gellevij et al. [285] designed two instruction manuals for physics teachers to teach the use of the SimQuest application. One manual contained only textual instructions on how to use the software. On the other hand, the other manual additionally contained screen captures with text. Both the manuals contained explanative text. The authors [285] also designed the screen captures in terms of Mayer et al.'s [5] system topology and component behavior. The screen capture showed the major parts (system topology) of the application and also state changes (component behavior) after interacting with the application. The participants were also novices in using computer applications. The authors also instilled Mayer's [5] fourth criteria (sensitive test) to measure learning outcomes. They measured participants' mental model development by asking recall questions, predicting successive screens, and by asking

them to identify errors [285]. The authors also measured participants' cognitive load and training time. The results from 44 participants showed a statistically significant overall effect on the development of mental models [285]. Participants from the visual manual group scored 14% higher than participants of the textual manual. The training time of visual manual group participants was also statistically significantly lower (11% slower) than the textual manual participants. Though the visual manual group contained both texts and images, there was no significant difference in cognitive load between the participants of the two groups. The authors [285] concluded that their study satisfied the four conditions for effective illustrations proposed by Mayer and Gallani [5] and proved their effectiveness.

Though these studies revealed the potential of explanative diagrams, the role of explanative diagrams in a lecture has not been studied. Recently, in 2015, Bui and McDaniel [286] aimed to investigate the potential benefits of explanative diagrams during a lecture. They also compared the learning outcome of an explanative diagram with an outlined note-taking. In this study [286], 144 undergraduates were randomly distributed into three groups: control, outlines, and illustrative diagrams. Participants were asked to listen to a 12-minute lecture about brakes and pumps; each topic presented one after the other. The lecture was created from the two different passages by Mayer and Gallini [5]. The participants were encouraged to take notes while listening to the lecture. The illustrative diagram group was given explanative diagrams of brakes and pumps (taken from Mayer and Gallini [5]) while the participants of the group listened to the audio lecture. The authors provided a skeletal note-taking outline to the outline group [286]. Lastly, the control group was given blank notepads to take notes. Afterward, two tests, recall and short-answer tests, were administered to measure learning outcomes [286]. The short-answer test was adopted from Mayer and Gallini [5]. The authors also analyzed structure building ability of participants across the three groups [286]. "Structure building reflects the ability to build coherent

mental representations (structure) of information." [286, p. 131]. It is a standardized test that asks to answer multiple-choice questions after each reading to assess understanding [287]. Higher scores on the multiple-choice tests mean a greater ability to create good mental models of the readings [288, 289]. The results revealed that the outline and illustration group participants significantly outperformed the control group in the recall test [286]. However, unlike Mayer et al. [5], there was no difference found across the outline and illustration group regarding the recall. Importantly, the authors found that low structure builders performed better with the illustrations than with the outline in the short answer questions. This finding implies that explanative illustrations serve as a scaffold to the learners who cannot easily generate mental models after a reading task. However, for high structure builders, there was no difference in performance for the short answer questions [286]. The authors concluded that the explanative diagrams likely engaged the participants in deeper levels of comprehension while listening to the lecture [286]. They also concluded that providing some aids to the students while note-taking (e.g., diagrams, outlines) is better than no aids [286].

The above studies reflect the qualitative advantage of diagrams in reasoning and how by retaining the two criteria system topology and component behavior, diagrams can help students build strong mental models.

B.4 Explanative Diagrams: A Model

In 2017, Paul E. Smaldino [208] wrote a chapter in Computational Social Psychology titled "Models are stupid, and we need more of them". There, he wrote:

"Stupid models are extremely useful. They are useful because humans are boundedly rational and because language is imprecise. It is often only by formalizing a complex system that we can make progress in understanding it" [208, p.311].

Smaldino sheds light on the importance of formal models. He defined formal models as articulating the parts of a system and the relationships between those parts.

Explicit articulation of parts and relationships defines a scientific system and separates it from "wishy-washy" irrelevant information [290–292]. Formal models remove the vagueness of verbal models [208]. Verbal models are "implicit models in which the assumptions are hidden, their internal consistency is untested, their logical consequences are unknown, and their relation to data is unknown" [293]. Smaldino [208] describes the danger of verbal models as there are many ways to describe the parts and relationships of a system. By precisely formalizing the parts and steps, formal models lay out the assumptions in detail and illuminate core dynamics [293]. From laying out the assumptions, we can reach conclusions[[293]. Even if the conclusions are flawed, we can examine how they differ from reality, then refine our models, ultimately becoming less wrong [294–296]

Formal models also explicitly articulate in which way they are simplifying reality [208]. Models should be stupid because humans are stupid [208]. Here, stupidity is a feature, not a bug. Our brain can not capture all the details; it always ignores some information. By ignoring all but the irrelevant information, we are making the best use of our cognitive capability. Formal models also have layers of abstraction. The hidden details are the strength of the formal model. According to Smaldino [208], modelers can sometimes be stupid. Modelers often inarticulately describe parts and the relationship between them. It depends upon the modeler to clearly maintain the criteria of formal models.

Explanative diagrams are formal models. Here, I will present the rationale behind my claim. First, if you look into the definition of the explanative diagram and formal models, they are identical. Formal models delineate a system's parts and the relationships between those parts. Here, the parts were defined by the system topology criteria, and the component behavior defined the relationships between those parts. An exaplanative diagram retains these two criteria. Thus, by definition, an explanative diagram is a formal model.

Second, explanative diagrams have abstraction layers. Every single detail of a system is not portrayed in explanative diagrams. Explanative diagram's system topology only illustrates the major parts, not all the parts. As an example, let's consider the explanative diagram of the braking system. Only the major parts and state changes of the parts are shown here. Several irrelevant information such as the size of the width of the brake shoe, the depth of the brake tube are not emphasized here.

Third, explanative diagrams make implicit assumptions explicit. With a diagram, a structure can be represented in a coherent way. Imagine the verbal model of the braking system. Words are not spatial. They can not fully draw the picture of which parts are connected with which parts and in what direction. By explicitly portraying the complex mechanism of a system with diagrams, every major aspect of a system becomes clear and explicit.

B.5 Explanative Diagram: A Notional Machine

Explanative diagrams can be a representation of a notional machine. Below I describe in the context of this dissertation, how explanative diagrams can serve the purpose of a notional machine.

Notional machines have pedagogical intent, as do the explanative diagrams. In Section B.2.1, we saw that notional machines are designed for pedagogical purposes, in short, for teaching. The purpose of creating explanative diagrams is also to help students to learn. The origin of explanative diagrams came from the need to promote meaningful learning [5]. It specifically deals with improving learners' mental models. Therefore, the purpose of explanative diagrams and notional machines are the same.

Notional machines are simplified conceptual models, so as explanative diagrams. Notional machines are simplified abstractions of a conceptual model. I argue explanative diagrams are the same. Mayer, while analyzing the role of models in learning, wrote: "A conceptual model highlights the major objects and actions in a system as well as the causal relations among them" [223, p.43]. From this definition, we can see

that the explanative diagram defines the same. Here, the explanative diagram is not the most detailed one but covers the major aspects of a system. From the literature, we know notional machines are simplified conceptual models created in an educational context. Thus, I can conclude that explanative diagrams are the representation of a notional machine.

Notional machines are tied to a concept, so as explanative diagrams. As summarized in Section B.2.1, a notional machine is an idealized abstraction of a programming concept. The programming language's semantics governs its representation. Explanative diagrams are also tied to a system. They represent the major components and dynamics of a system. Therefore, explanative diagrams of a programming concept will represent its structure and states, just like a notional machine.

The above arguments suggest that explanative diagrams can serve as a notional machine in the context of Computer Science education.

B.6 Explanative Diagram of Arrays (EDA)

Keeping in mind the theories of mental models, multimedia learning, notional machines, and the theory of effective diagrams, I designed a diagrammatic notional machine of the programming concept 'arrays'. Arrays are one of the fundamental programming concepts and also have an inherent structure. Programmers interacting with the structure produce dynamic states of the array. Arrays are more than variables and also carry some of the complexities of objects (e.g., reference variable, memory allocation). I reported many misconceptions in Section **??**that a novice can have in variables, arrays, and objects. In that Section, we saw that arrays could carry misconceptions of variables and objects. This makes the array a programming concept that can be a hub of many misconceptions carried from primitive variables and objects. The misconceptions suggest there exist hidden ambiguities among novice programmers about arrays. In this case, a notional machine portraying the implicit structure and states of an array can improve novices' flawed mental models. Here, I

propose a diagrammatic representation of the notional machine. I followed the principles of explanative diagrams to design the diagrams and the principles of multimedia learning to create an integrated model of text and pictures.

The EDA retains the two criteria of explanative diagrams: system topology and component behavior.

System Topology: The Structure

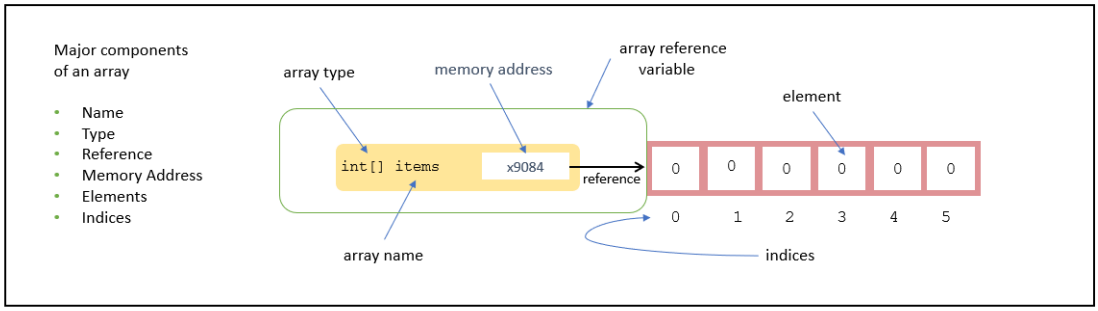


Figure B.10: The system topology diagram of an array.

I will first introduce the structure of the array and show each major component of an array. Figure B.10 is the system topology of an array named *items*. Here, a rectangular box (yellow box in Figure B.10) embedded with another rectangle with an arrow is used to represent the array reference variable. The name is inside the yellow rectangular box associated with the type. To give the detail that reference variables store memory addresses, not values, we placed a 5-digit number inside the embedded rectangle which represents the memory address. Our textbook survey [159] found that the authors portrayed array reference variables with a rectangle box with an arrow in it; no indication of a memory address is there. Memory diagrams [196, 297], object diagrams [195] also followed the same approach. In the literature on programming misconceptions (Section 2.3.3), I reported that students think no memory is allocated after an array is created. Furthermore, the absence of memory addresses can not explicitly portray that array assignment copied memory addresses, not val-

ues. Though in my diagram, I introduced the subtlety of memory, I did not include further unnecessary details about memory. In my EDA, square boxes are used to portray elements with the mention of indices right beneath them. The system topology diagram outlines each major part with arrows. In one experiment, Julie Heiser and Barbara Tversky [298] found that participants who viewed a diagram outlined with an arrow produced a more functional description of the system than a diagram without an arrow. Moreover, I also listed the major components of the array annotated in the diagram. By doing this, my diagram followed the CTML's spatial contiguity principle (described in Section B.2.2), where corresponding words are placed next to the corresponding part of the diagram.

Component Behavior: The States

After describing and portraying the array's structure, I introduce each state change that an array can have. The state changes were described chronologically according to the execution order. For example, the state declaration is introduced before instantiation.

Declaration For each state change, I portrayed the before and after state of an

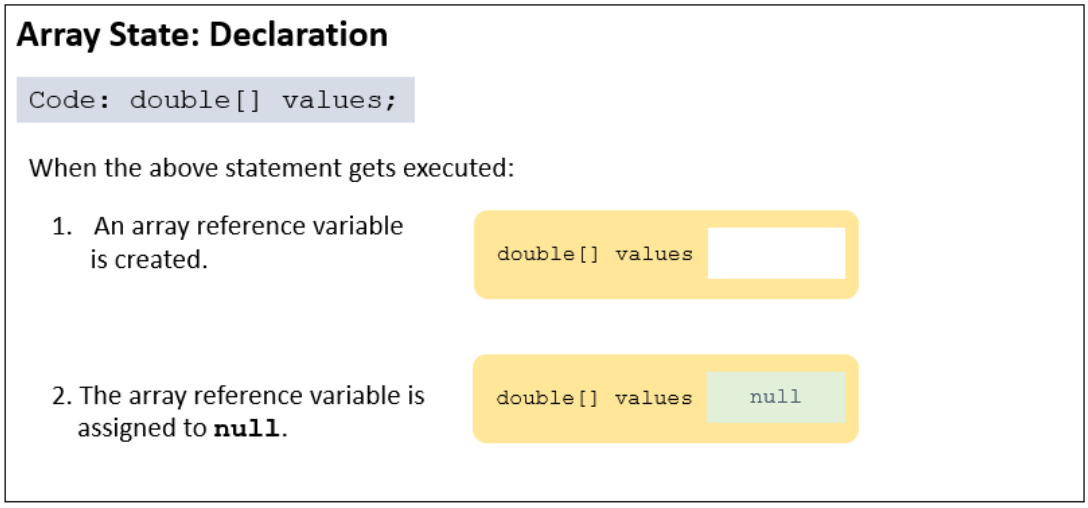


Figure B.11: The explanative diagram illustrating an array's state after declaration.

array. The name of the state is highlighted in the header of the diagram (see Figure B.11). I also placed the corresponding code adjacent to the diagrams. The explanative texts on what the code does are also placed next to the diagrams following the spatial contiguity principle (described in Section B.2.2). I incorporated the signaling principles in the diagrams by highlighting essential information. For example, in the diagram explaining declaration (see Figure B.11), the word *null* is a bold and white rectangular box containing nothing changed to green to show the change in that particular component. The two diagrams showing the change in the memory address are crucial. By showing both the blank state and the null state of the reference variable, I draw students' attention to the fact that null is not the same as nothing. Lewis [194] reported being asked by the students how *null* is both nothing and something? She claimed it helped students to understand the difference by describing a *null* reference as an empty pocket [194]. Later, the transition from the empty pocket to a box with an X in it. Here, she used X to represent *null*. The transition of the white rectangular box also emphasizes the fact that it is an array reference variable whose value changes. Here, one subtle thing to note is that I showed two changes of the array reference variable by the execution of one line of code in this diagram. First, the array reference variable gets created, and then the *null* is assigned to it.

Instantiation

The array structure takes full form after instantiation. I addressed the instantiation process as three steps: 1) allocating the memory for the array, 2) assigning the default values to the elements, and 3) assigning the reference to the array reference variable. As Figure B.12 shows, I first showed the creation of the array. The top diagram of Figure B.12 shows the state of the array after it has just been created. The explanative text next to the diagram explains how it happened.

The next diagram (bottom one) shows the assignment of default values to each

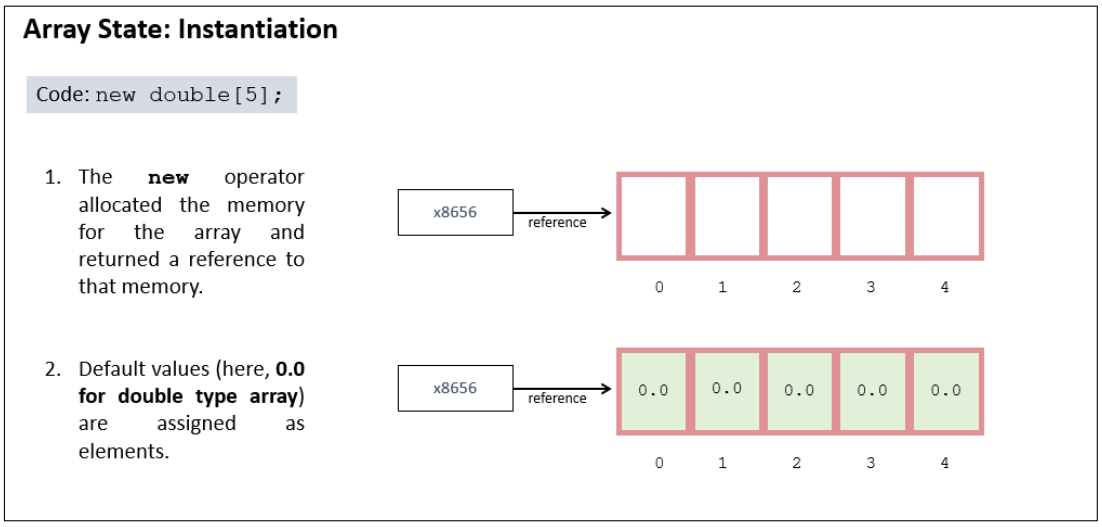


Figure B.12: Explanative diagrams showing the each state change after instantiation.

of the elements. By doing that, I try to draw students' attention to the fact that in Java, there are default values when an array is created. As students often think there are no default values [7] and textbooks portray *blank* or question mark *'?'* as default value [159], I wanted to address this issue explicitly. Here, I also utilized the spatial contiguity principle, signaling principle, and segmenting principle. The next diagram (Figure B.13) portrays the assignment of the reference variable returned by the *new* operator to the array reference variable. The diagram shows the before and after state of the reference variable. The top diagram shows the state of the array reference variable after declaration; the diagram below shows the state of an array reference variable when it is initialized. If one focuses on the memory address, the same memory address which was generated by executing the code `new double[5]` is assigned to the *values* array reference variable. If I did not portray the memory address with numbers and placed only the arrow, this subtle change would not be explicit. I signaled the change of the array reference variable by making the rectangle box green.

Assigning Elements Assigning elements is like assigning a value to a variable.

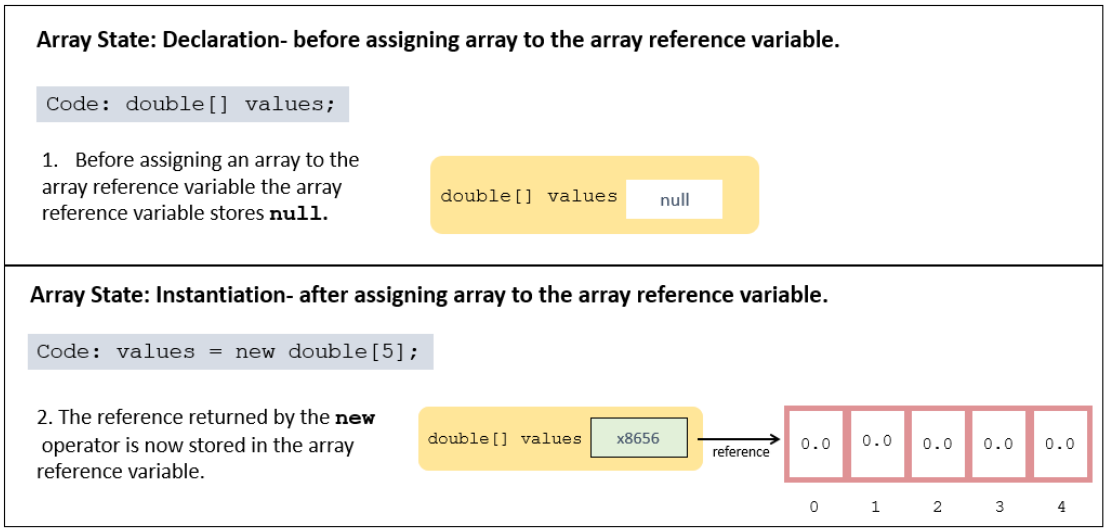


Figure B.13: The explanative diagrams showing the before and after state change of an array after assignment.

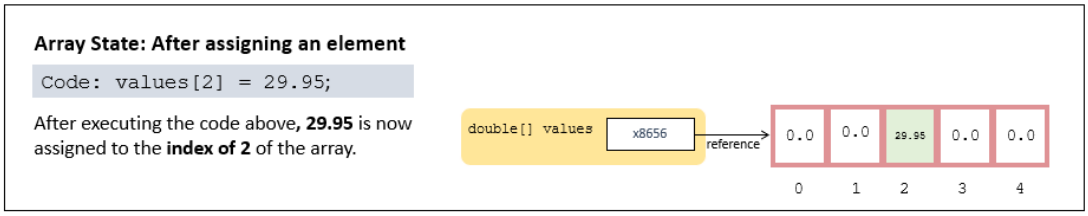


Figure B.14: The explanative diagrams of assigning a value to an element.

In this case, I only showed the 'after' state of the assignment as the state of other unassigned elements represents the 'before' state (see Figure B.14). By doing this, I am adhering to the coherence principle (described in Section B.2.2), which ensures extraneous information is excluded. Here, I retained the signaling principle and spatial contiguity principle.

Array Assignment

The array assignment state describes what happens when an array is assigned to another array. Here, a crucial aspect is that the values of the reference variables change, not the values of the elements. To address this issue in detail, I showed the before state of both the arrays related to the assignment. Figure B.15 shows the

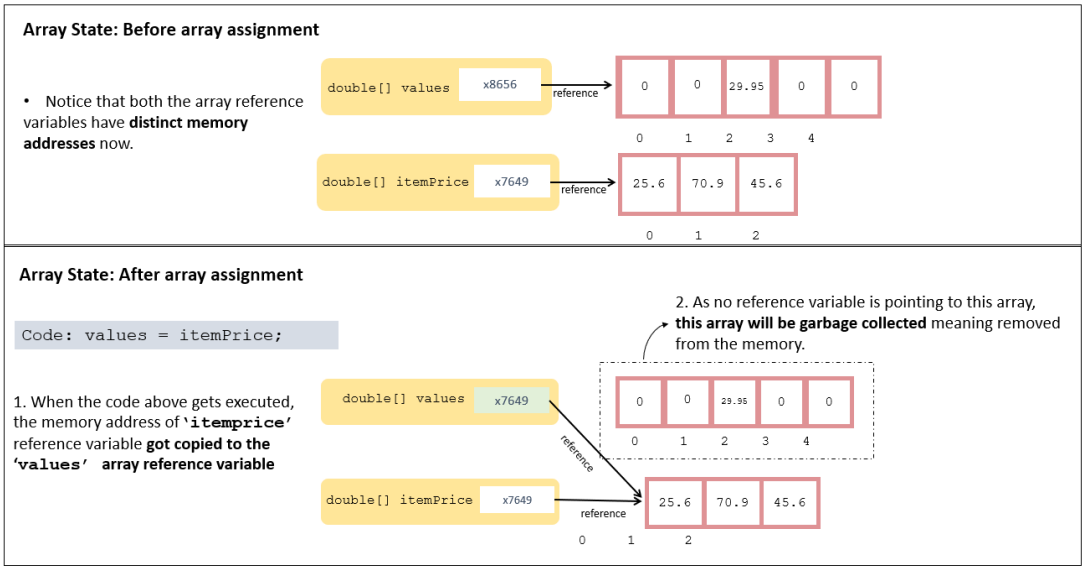


Figure B.15: The dynamics of array assignment are portrayed with this explanative diagram.

before assignment state of the arrays *values* and *itemPrice*. When the assignment code gets executed, the memory address of the *values* changes. This phenomenon is highlighted in the diagram by showing that the memory address of *values* and *itemPrice* are the same. As students have the misconception that values move from right to left during an assignment, I explicitly signaled the accurate change by making the rectangle box of values green. The fact that both the array reference variables can be used to access the same array is explicitly indicated by the two arrows pointing toward the same array.

The disposal of the unreferenced variable is also indicated in the diagram by placing dotted lines around it. The explanative texts here are also described in order of execution.

The structure of the array and the four states embody the system topology and the component behavior. An argument can arise here, such as this can be easily and intuitively shown with animation or other program visualization tools. Below, I describe my rationale behind choosing static diagrams vs. other animated graphics.

B.7 Why Static Diagrams, not Animation?

Diagrams have an ancient history. The history holds for both spatial and abstract elements and relations [299]. Research on static diagrams suggests that carefully designed and appropriate diagrams are beneficial in explaining complex systems [299]. Animation containing dynamic graphics often appears attractive and compelling. Because of its dynamics and convenience to show changes, it might be thought that the learning expands the learning of static diagrams. However, that is not the case in every situation. Tversky and Morrison [299] did a comparative literature review on animation and static diagrams. They reviewed the literature on animation used to teach complex systems, mechanical, biological, physical, operational, or computational. On these topics, previous research (e.g., [300–302]) does show the effectiveness of animation over static diagrams. However, the study methods are frowned upon [299]. Tversky et al. [299] reported that with a closer look at these studies, it is evident that the study methods have put extra elements in the animation, which can result in superiority. For example, for teaching circulatory systems, the use of animation was found to be more effective than diagrams [302]. However, in this case, the content of the animation was superior to the diagrams'. The animation portrayed blood pathways, whereas the static diagram did not. In these kinds of findings, Tversky et al. [299] noted: "When examined carefully, then, many of the so-called successful applications of animation turn out to be a consequence of a superior visualization for the animated than the static case or of superior study procedures such as interactivity or prediction that are known to improve learning independent of graphics" [299, p. 254].

Moreover, various studies (e.g., [301, 303–305]) reported no benefits of animation over static diagrams. As an example, Byrne, Catrambone, and Stasko [306] were surprised to find out that there was no difference in the effectiveness of animated graphics and static diagrams in teaching students computer algorithms such as depth-

first search and binomial heaps. In some cases, viewing animation slowed down the learning process [299,307] or even felt difficult to perceive and understand (e.g., [308, 309]. Tversky et al. [299] commented on that by saying animations may be distracting in converting essential information. Animations presented to novices help extract perceptually salient information but not the causal information of a system [299,308]. Tversky et al. [299] argued the most likely cause of animations' failure is the perceptual and cognitive limitation in the processing of a changing visual situation, lacking the apprehension principle. On the other hand, multiple diagrams can provide the same information as animation with an additional advantage. Multiple diagrams allow the learners to compare, contrast, and make deep inspection [299]. An animation where frames are fleeting with time can be harder to focus on [299].

The above findings suggest that there are more benefits in incorporating effective diagrams than including animations. Due to diagrams' accessibility and ease in use, diagrams designed by reflecting on how people learn can be a perfect candidate for promoting meaningful learning.

B.8 The Implications of EDA in Learning

The explanative diagram of arrays is an integrated model of text and pictures. The diagrams are designed by following principles and criteria, which have been proven to generate deeper understanding and develop better mental models. The model's design principles utilized the principles of cognitive theory of multimedia learning (CTML). CTML has proven to promote meaningful learning. Meaningful learning enables learners to recall, predict, and troubleshoot. All three are included in the purpose of the mental model.

From the mental model theory perspective, Mayer et al. [5] claim explanative diagrams help learners achieve runnable mental models. It does that in two ways. First, explanative diagrams establish the connection between a system's structure and its function with the two criteria: system topology and component behavior [37]. There-

fore, by viewing EDA, novice programmers can readily connect an array's structure with its states. Most importantly, the placement of corresponding code in the diagrams explicitly connects the structure with its appropriate behavior. When novices interact with the structure through the programming language without the diagrams, the representation becomes abstract, and the outcomes of the interaction become obscure. Representing the internal system of arrays spatially with diagrams, each behavior, and function of the array becomes clearer. By doing this, there is a lesser chance of novices embedding unwitting assumptions, which can later lead to misconceptions. Second, explanative diagrams make the structure-function connection more robust by making implicit assumptions explicit. The robustness of a learner's mental model is increased when they can make their implicit assumptions explicit [37]. By portraying each state changes in relation to the array's structure, explanative diagrams help the learners to identify implicit assumptions and remove their ambiguities.

I propose a model of texts and diagrams to teach a programming concept. Models have many advantages. People often believe models empower us with only predictability. Nonetheless, models can offer more. Dr. Joshua M. Epstein [293] known for agent-based modeling, listed 16 reasons to build models for other than prediction. According to him, models can explain when we make them explanatory. EDA retains the feature of being explanatory. Models can illuminate core dynamics. Models form the conceptual base in learners by capturing the qualitative behaviors of overarching interest [293]. EDA, serving as a conceptual model, illuminates the core dynamics of arrays. Models raise new questions [293]. Models illuminate core uncertainties, which in turn provoke curiosity [293]. When Lewis [194] with her physical memory model demonstrated empty pockets as nothing and a cross (X) as null, students were more inquisitive to learn the difference between nothing and null. EDA portraying subtle details of state changes opens the avenue to raise questions in students' minds.

EDA is a representation of notional machines. Researchers [2,3,30,34] of CS educa-

tion agree that we should teach notional machines in classrooms. By explicitly teaching the notional machine, we prevent students from building knowledge intuitively. However, notional machines can be effective if they are designed carefully [30]. EDA is a representation of notional machines that follow the guidelines of cognitive science and mental models. The evaluation of EDA with empirical studies will ensure that the theories behind its design can achieve pedagogical goals or not.

APPENDIX C: MARKING SHEET FOR CONTRADICTORY ASSERTIONS

The sets of assertions in each cell contain contradiction. If any two assertions are found among each sets of assertions for a component, the mental model of that component is marked inconsistent. The assertions among different cells may not be contradictory. For example, MI4: Array index starts with 1 and MI5: Array index ends with n (index of last element) are not contradictory. MI4 is about the start index of an array and MI5 is about the end index of the array. Thus, they are placed in different cells (see Figure C.2).

Table C.1: Contradictory Assertions for *name*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MN1: Array reference variable is the name of the array. MN2: Array type is the name of the array. MN3: Keyword new is the name of the array. MN4: Whatever comes after equal sign in an initialization is the name of the array. MN5: First element in the array is the name of the array.

Table C.2: Contradictory Assertions for *index*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MI1.1: Array index starts with 0. MI2: Index of an array can be of any type, not just integers. MI3: There is no indexing into the array. MI4: Array index starts with 1. MI6: Array index does not map to its corresponding location in the array. MI7: Students think the index is the element.
MI1.2: Array index ends with n-1 (index of last element). MI2: Index of an array can be of any type, not just integers. MI3: There is no indexing into the array. MI5: Array index ends with n (index of the last element). MI6: Array index does not map to its corresponding location in the array. MI7: Students think the index is the element.

Table C.3: Contradictory Assertions for *type*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MT1: Name appearing before the array name is the type of the array. MT2: Name of the array is the type of the array MT3: Keyword new is the type of the array.

Table C.4: Contradictory Assertions for *element*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
ME1.1: The array contains n number of elements. ME2: The array contains size+1 elements. ME3: The array contains size-1 elements. ME4: After instantiation, the array doesn't have space to store any elements (size 0).
ME1.2: Elements stored in the array can be only of the declared type. ME5: Element values and array names are related (e.g., books and "Harry Potter"). ME6: Keyword 'new' is an element in the array. ME7: Type of values stored do not match type of array.

Table C.5: Contradictory Assertions for *declaration*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MD1.1: After declaration, the default value of array reference variable is set to null. MD2: There is no default value for the elements of the array (blank/no value). MD3: The default value for the array reference is the default value for type (e.g., int is 0, boolean is false). MD4: The default value for the array reference is stored as '?'.
MD1.3: After declaration, no memory is allocated for the array. MD5: After declaration, memory is allocated for the elements.
MD6: After declaration, the number of elements that can be stored is unlimited. MD7: After declaration, there is a default size for an array.

Table C.6: Contradictory Assertions for *instantiation*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MIn1.1: After instantiation, memory is allocated for the array. MIn4: After instantiation, no memory is allocated.
MIn1.2: After instantiation, the appropriate default value is assigned to the elements. MIn2: After instantiation, '?' is stored as a default value. MIn3: After instantiation, there is no default value (blank) stored for the elements.

Table C.7: Contradictory Assertions for *assigning elements*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MAE1.1: Assignment copies the values from right to left. MAE2: The value of a variable never changes. MAE3: A variable can hold multiple values at a time / 'remembers' old values. MAE4: Assignment swaps values of the left and right hand side. MAE5: Primitive assignment is the same as reference assignment.
MAE1.2: The variable on the right-hand side remains the same after assigning. MAE4: Assignment swaps values of the left and right hand side.

Table C.8: Contradictory Assertions for *assignment*. Assertions belonging to the same cell are contradictory.

Sets of Contradictory Assertions
MA1: Array assignment copies reference from right to left, sharing the memory. MA2: Array assignment appends value at the end of the array. MA3: Array assignment copies the values. MA4: Array assignment transfers (cuts) values. MA5: Array assignment copies the reference but does not share memory.

APPENDIX D: ITEM RESPONSE THEORY ANALYSIS

This appendix contains the item characteristic curves (ICC) for each item included in the Mental Model Test, as discussed in Chapter 9.

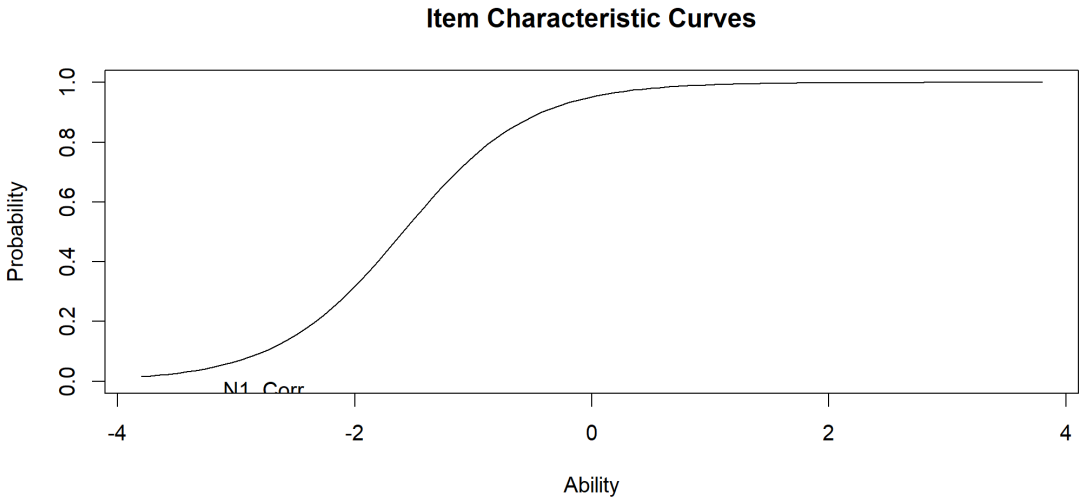


Figure D.1: Item Characteristic Curve for the item N1 included in the array's *parts* component- *name*.

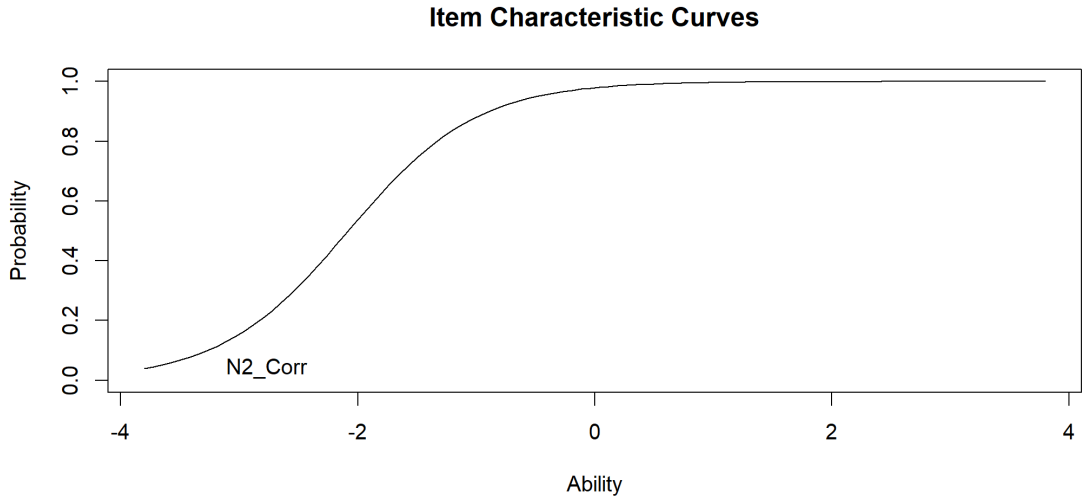


Figure D.2: Item Characteristic Curve for the item N2 included in the array's *parts* component- *name*.

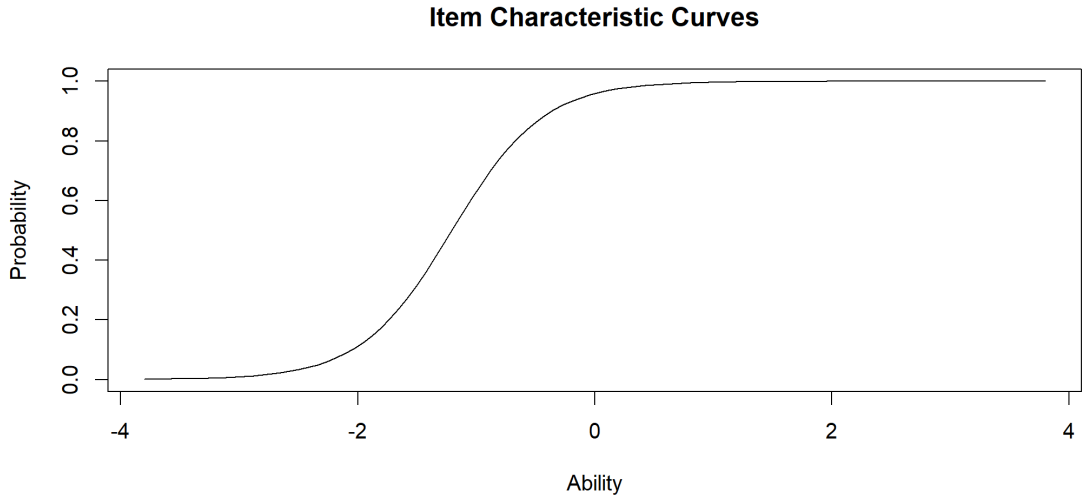


Figure D.3: Item Characteristic Curve for the item I1 included in the array's *parts* component- *index*.

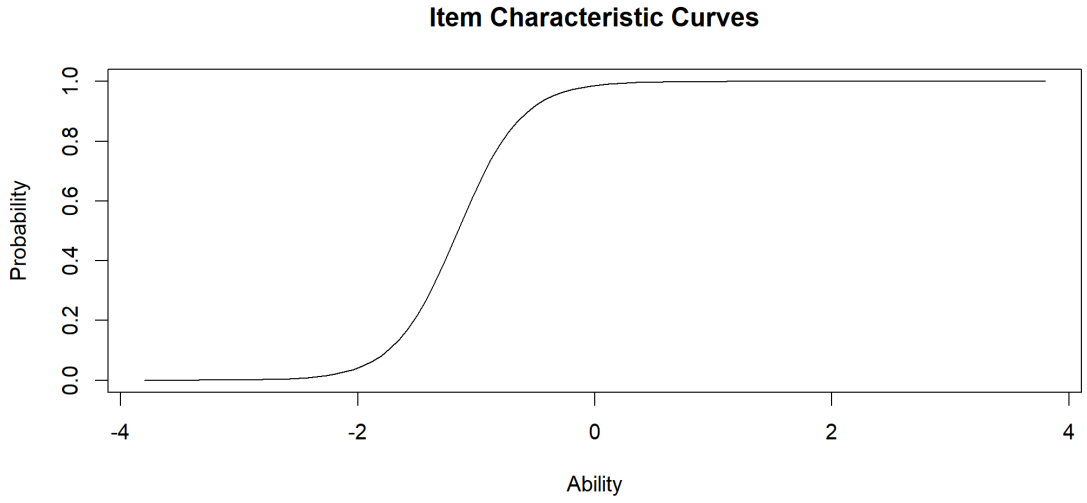


Figure D.4: Item Characteristic Curve for the item I2 included in the array's *parts* component- *index*.

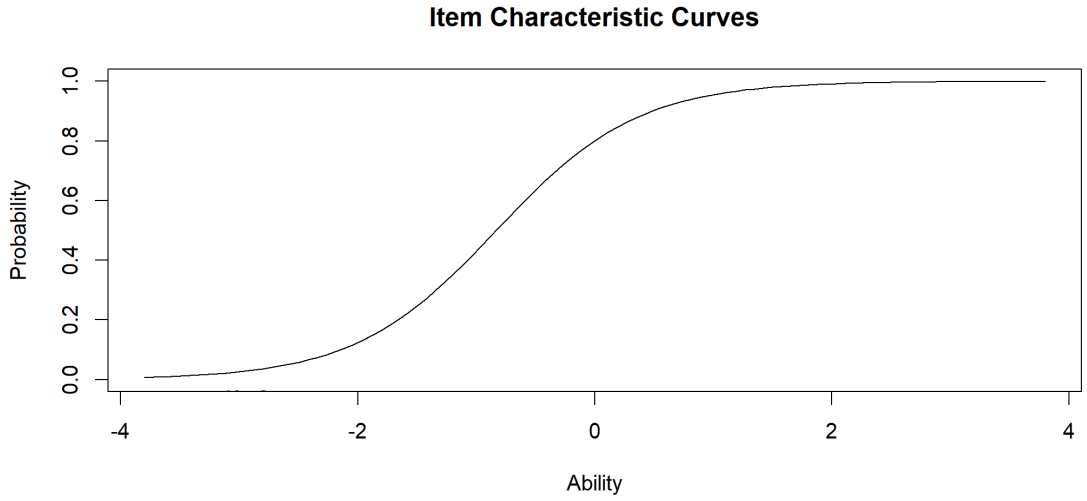


Figure D.5: Item Characteristic Curve for the item I3 included in the array's *parts* component- *index*.

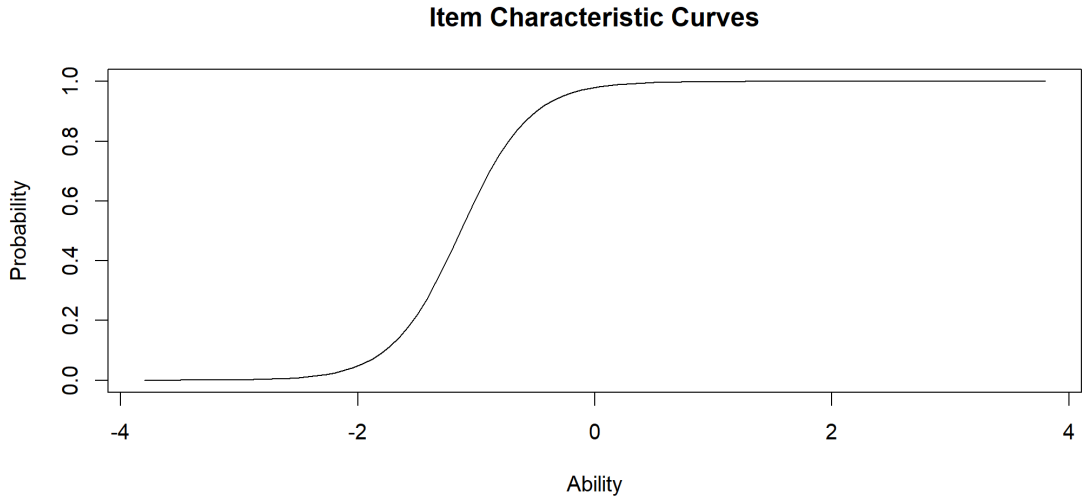


Figure D.6: Item Characteristic Curve for the item I4 included in the array's *parts* component- *index*.

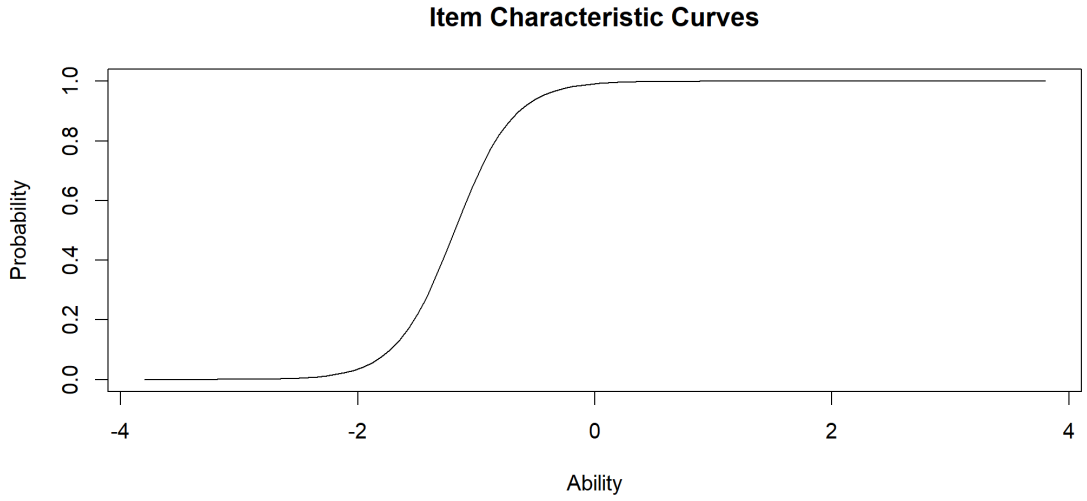


Figure D.7: Item Characteristic Curve for the item I5 included in the array's *parts* component- *index*.

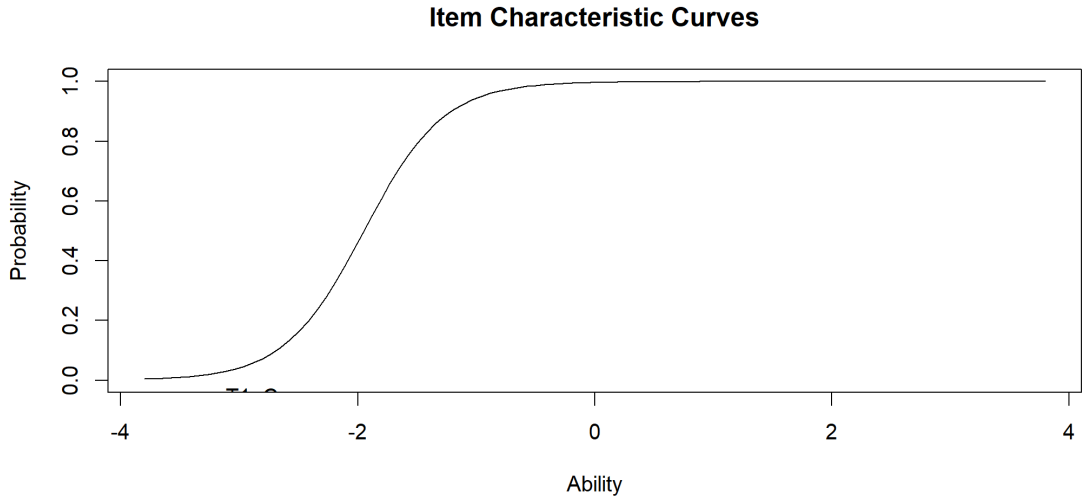


Figure D.8: Item Characteristic Curve for the item T1 included in the array's *parts* component- *type*.

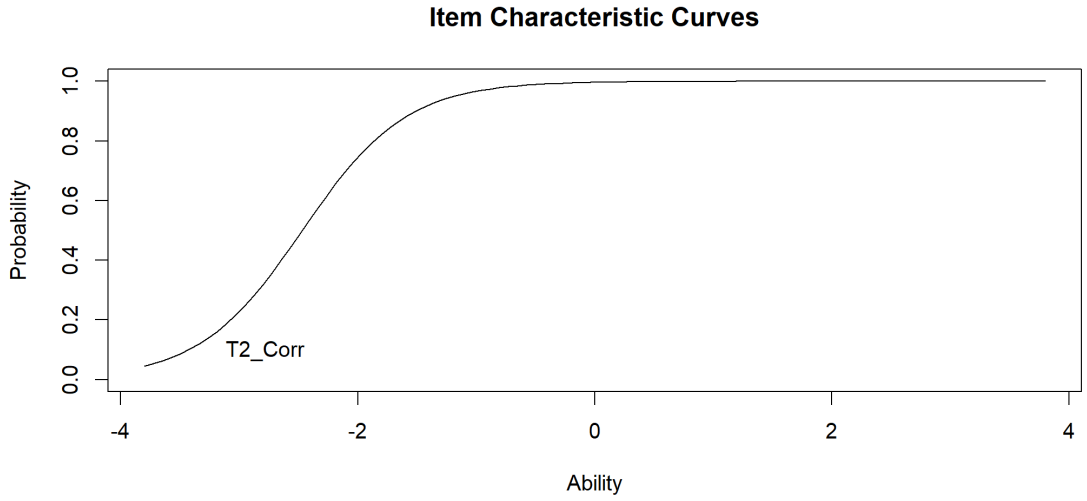


Figure D.9: Item Characteristic Curve for the item T2 included in the array's *parts* component- *type*.

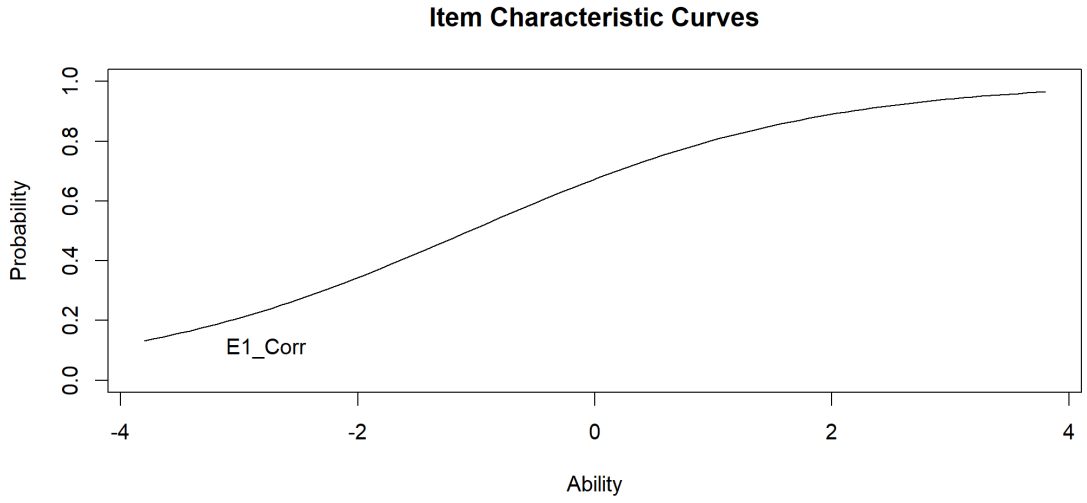


Figure D.10: Item Characteristic Curve for the item E1 included in the array's *parts* component- *elements*.

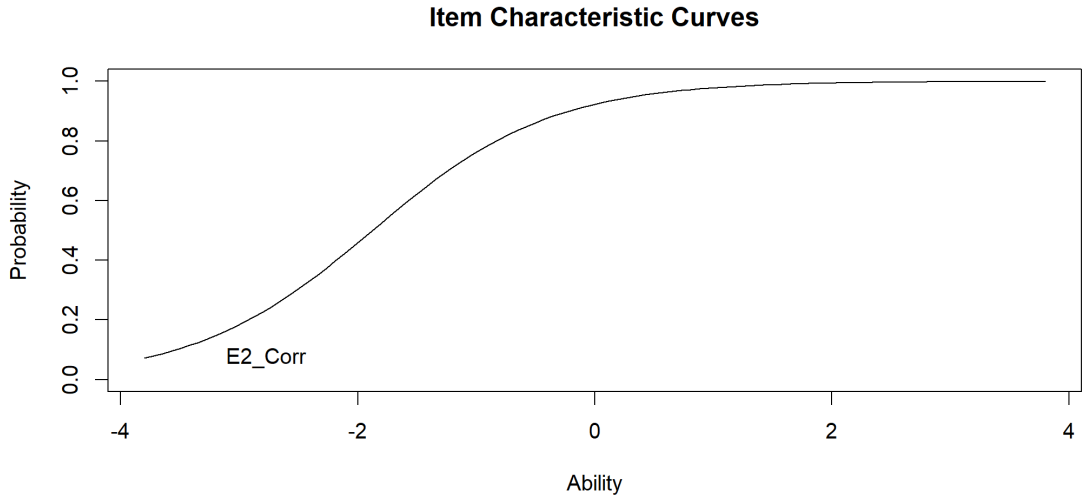


Figure D.11: Item Characteristic Curve for the item E2 included in the array's *parts* component- *elements*.

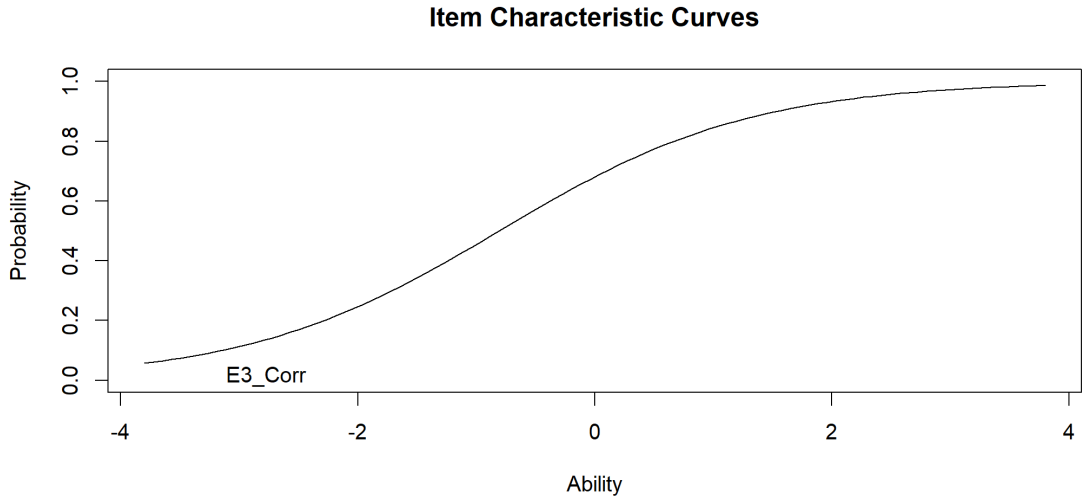


Figure D.12: Item Characteristic Curve for the item E3 included in the array's *parts* component- *elements*.

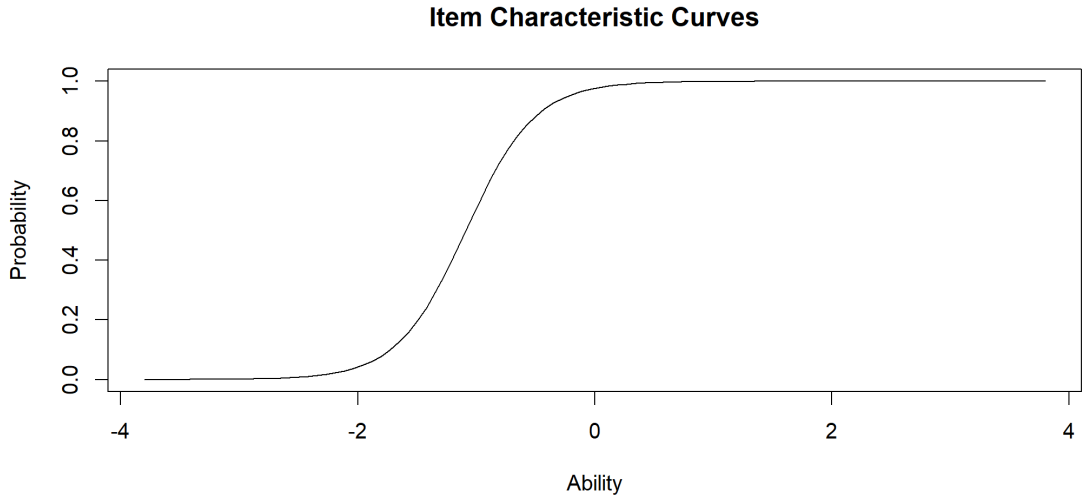


Figure D.13: Item Characteristic Curve for the item E4 included in the array's *parts* component- *elements*.

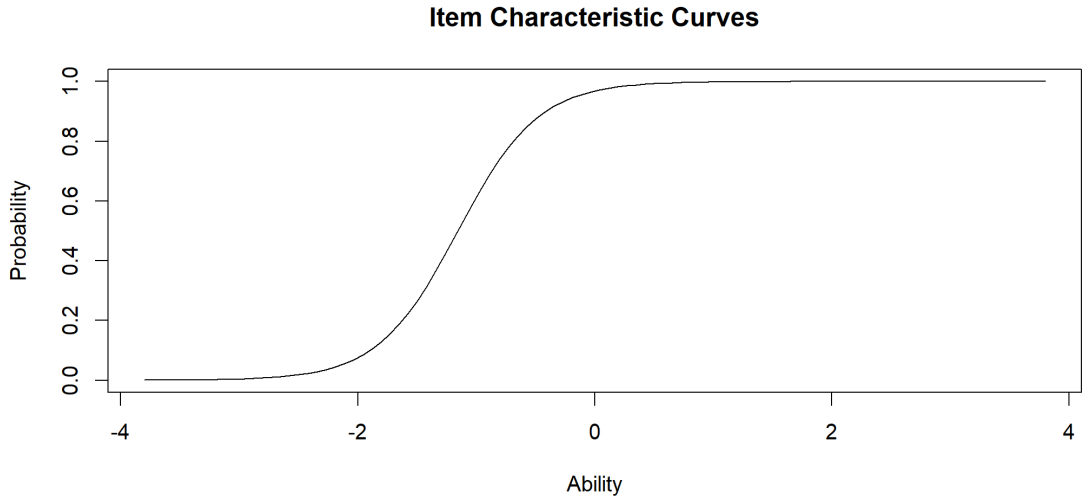


Figure D.14: Item Characteristic Curve for the item E5 included in the array's *parts* component- *elements*.

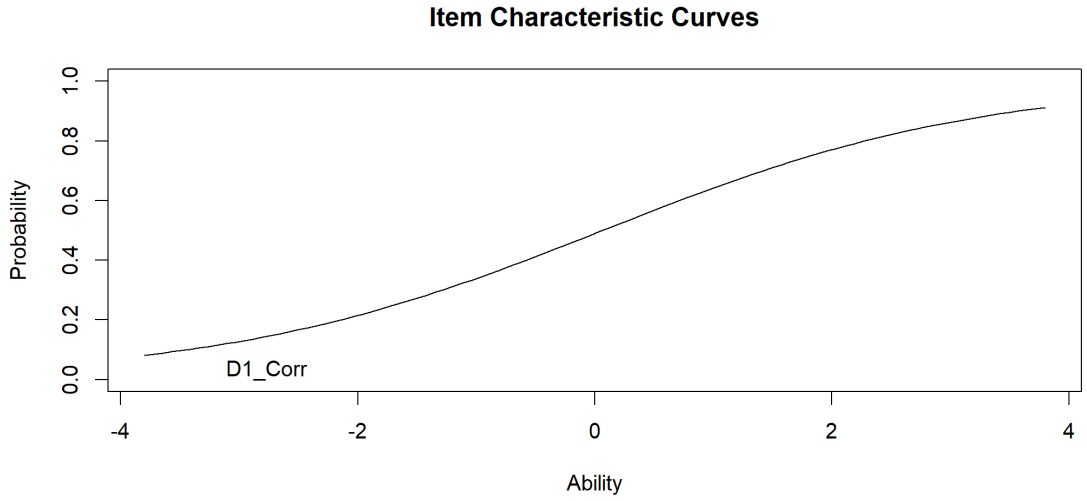


Figure D.15: Item Characteristic Curve for the item D1 included in the array's *state changes* component- *declaration*.

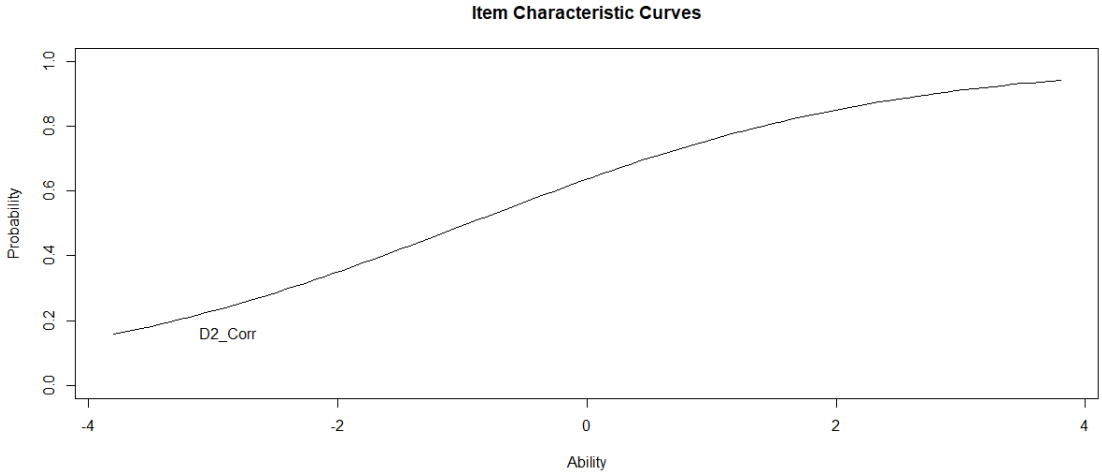


Figure D.16: Item Characteristic Curve for the item D2 included in the array's *state changes* component- *declaration*.

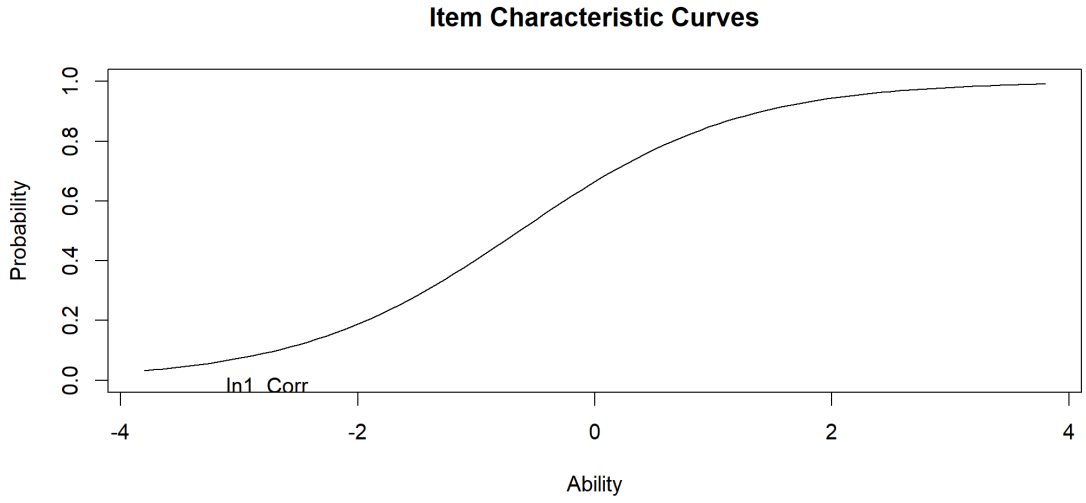


Figure D.17: Item Characteristic Curve for the item In1 included in the array's *state changes* component- *instantiation*.

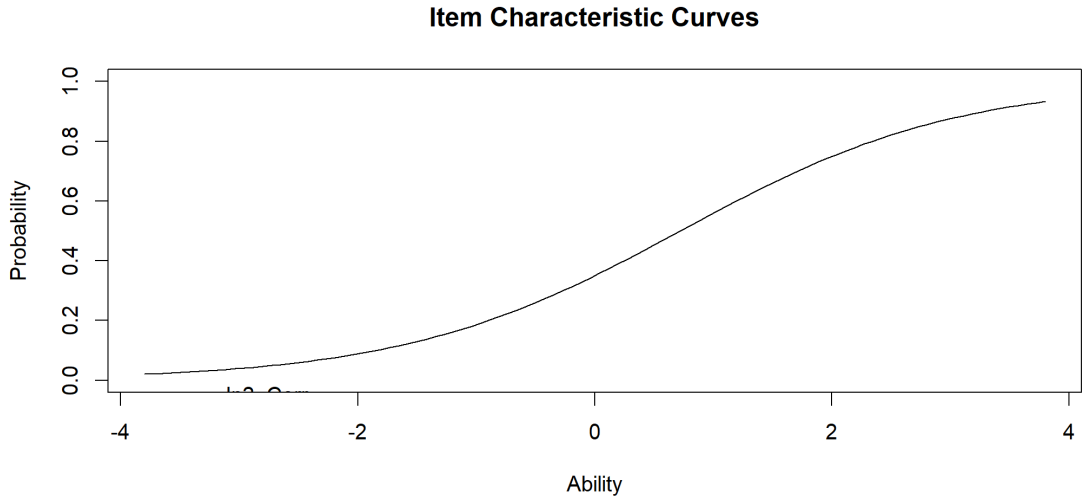


Figure D.18: Item Characteristic Curve for the item In2 included in the array's *state changes* component- *instantiation*.

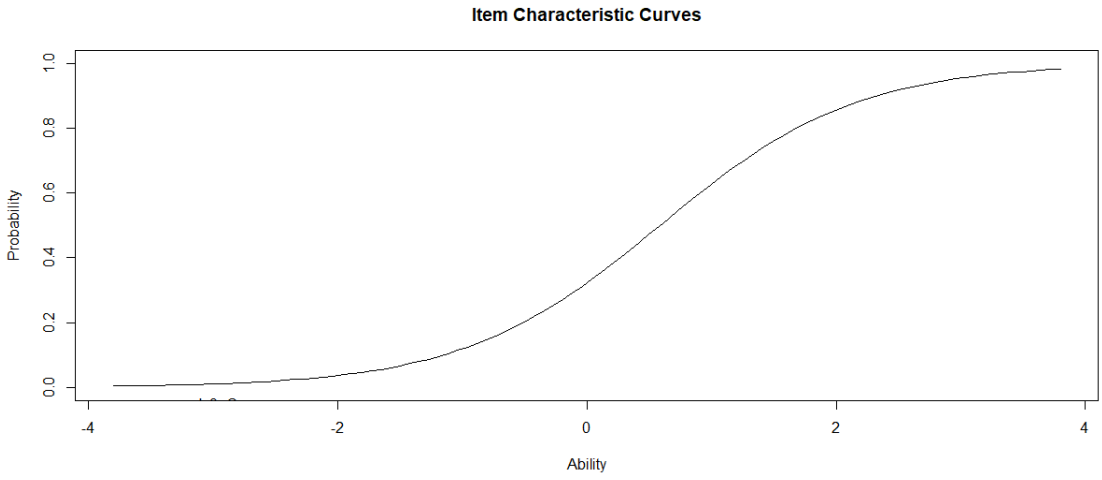


Figure D.19: Item Characteristic Curve for the item In3 included in the array's *state changes* component- *instantiation*.

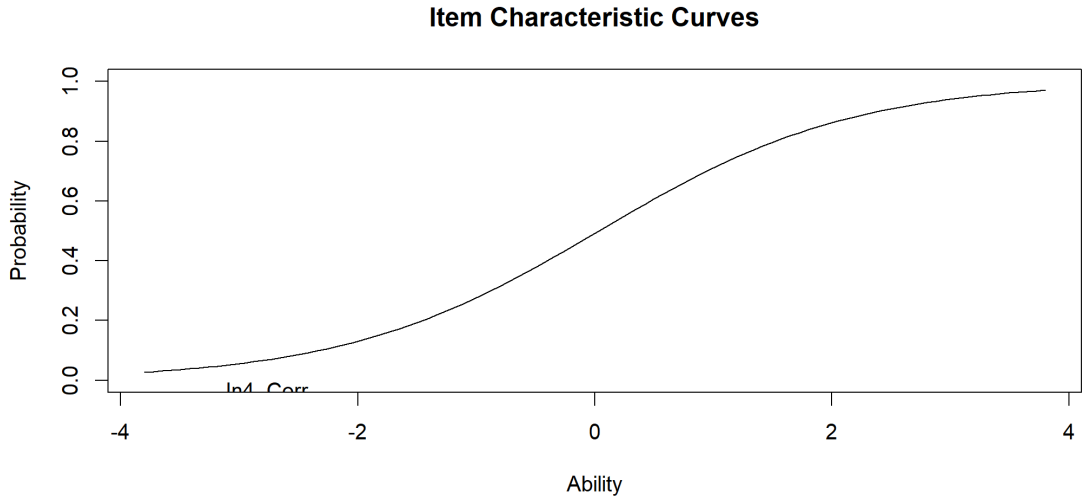


Figure D.20: Item Characteristic Curve for the item In4 included in the array's *state changes* component- *instantiation*.

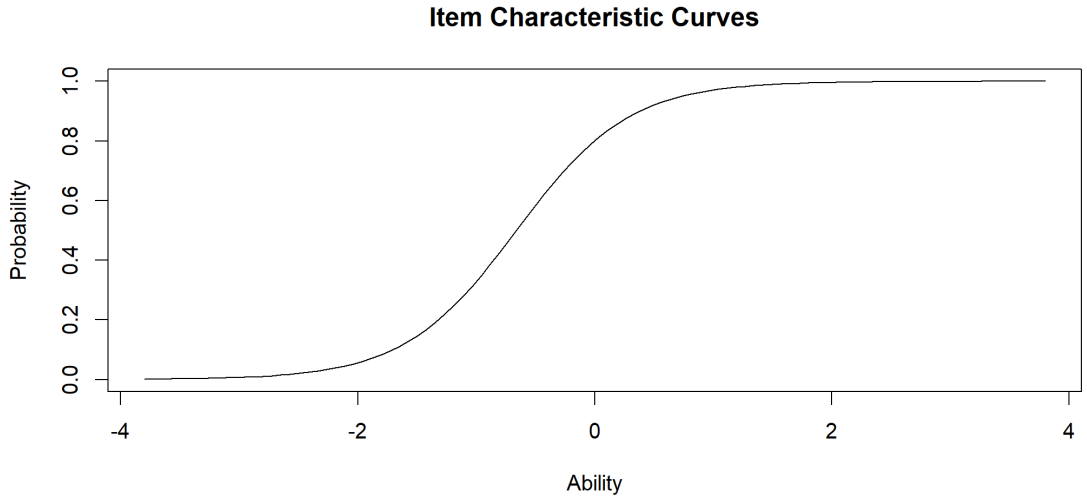


Figure D.21: Item Characteristic Curve for the item AE1 included in the array's *state changes* component- *assigning elements*.

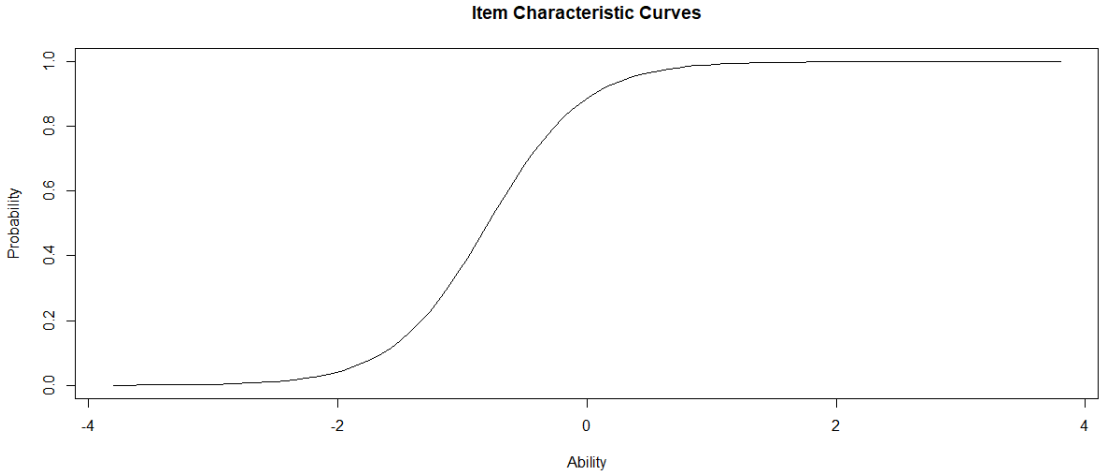


Figure D.22: Item Characteristic Curve for the item AE2 included in the array's *state changes* component- *assigning elements*.

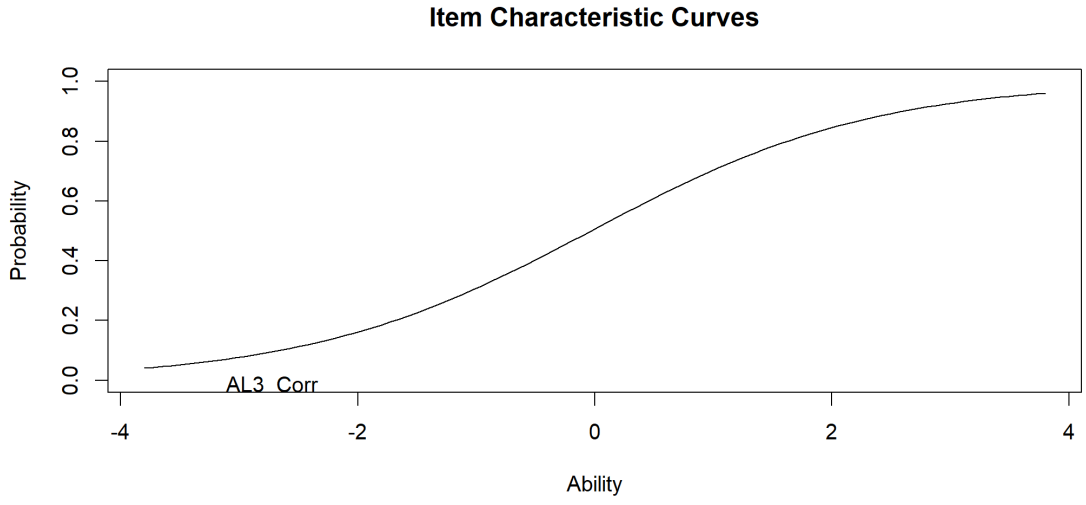


Figure D.23: Item Characteristic Curve for the item AE3 included in the array's *state changes* component- *assigning elements*.

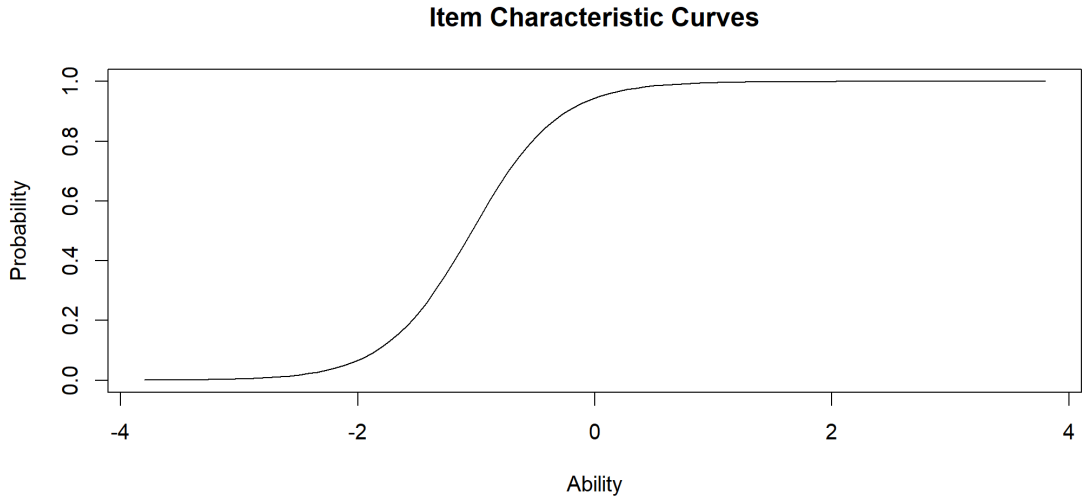


Figure D.24: Item Characteristic Curve for the item AE4 included in the array's *state changes* component- *assigning elements*.

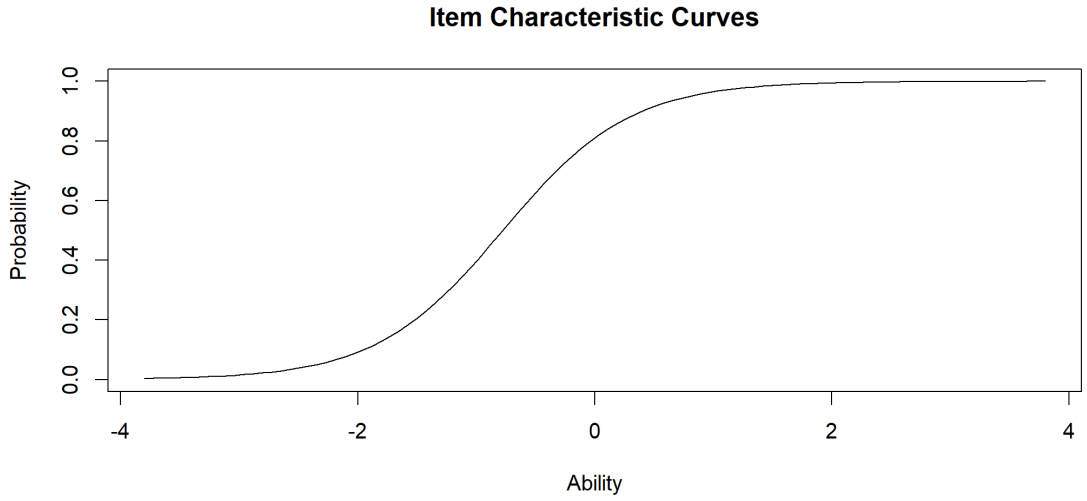


Figure D.25: Item Characteristic Curve for the item AE5 included in the array's *state changes* component- *assigning elements*.

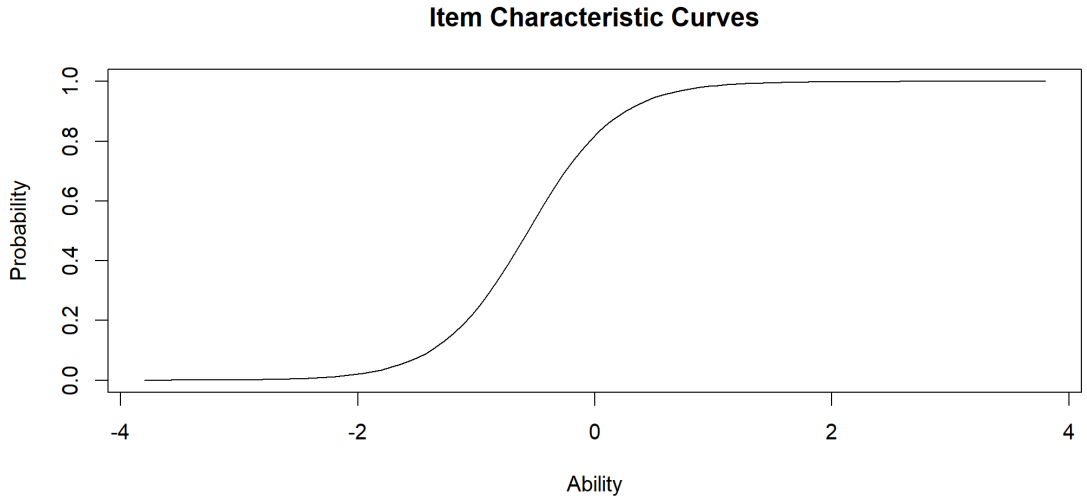


Figure D.26: Item Characteristic Curve for the item AE6 included in the array's *state changes* component- *assigning elements*.

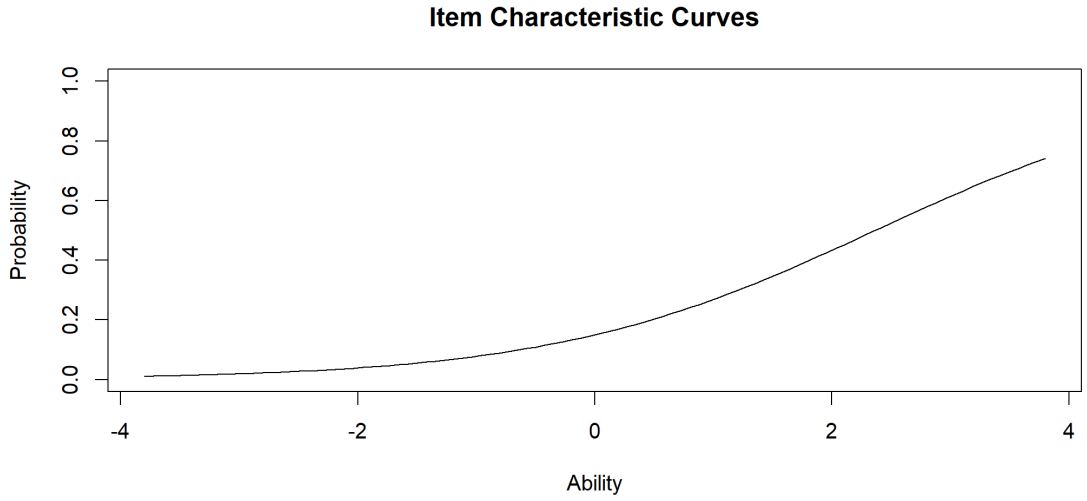


Figure D.27: Item Characteristic Curve for the item A1 included in the array's *state changes* component- *assignment*.

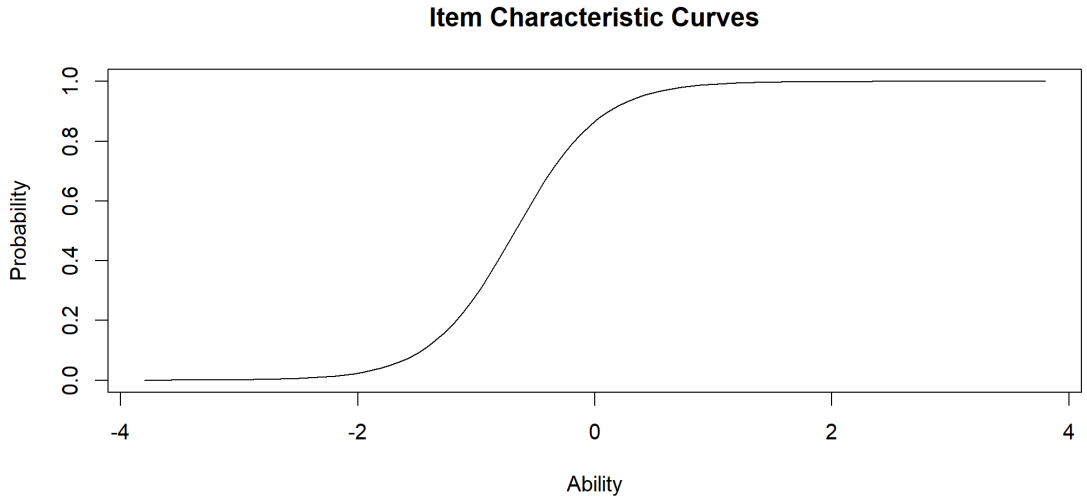


Figure D.28: Item Characteristic Curve for the item A2 included in the array's *state changes* component- *assignment*.

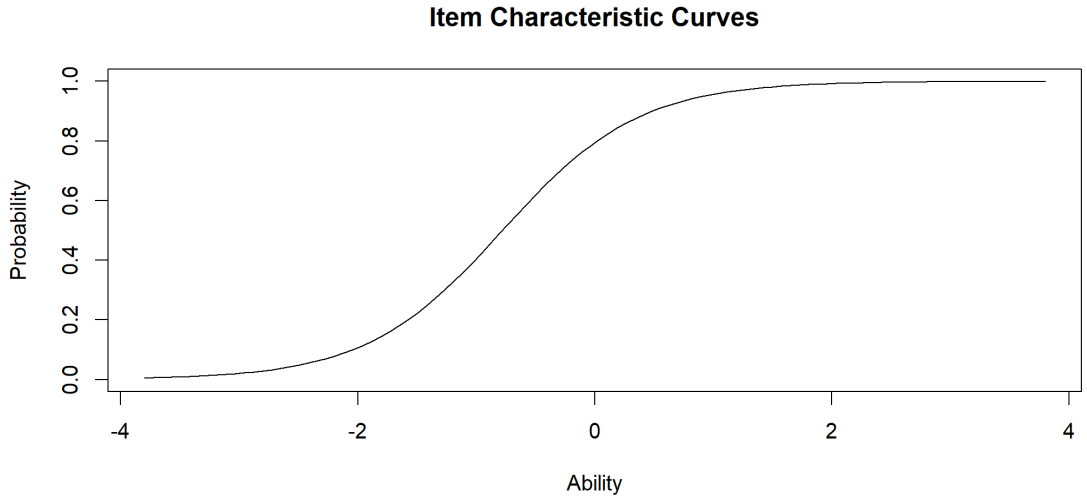


Figure D.29: Item Characteristic Curve for the item A3 included in the array's *state changes* component- *assignment*.

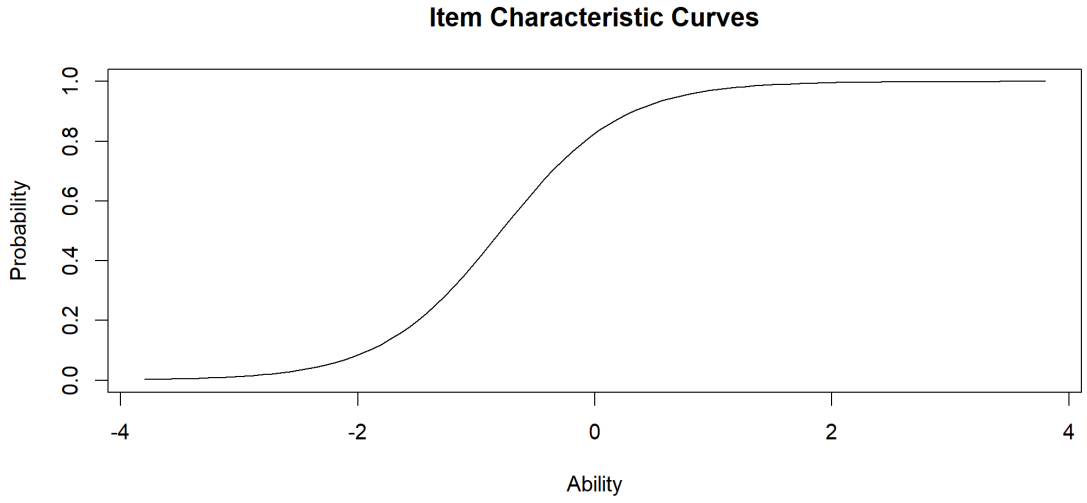


Figure D.30: Item Characteristic Curve for the item A4 included in the array's *state changes* component- *assignment*.

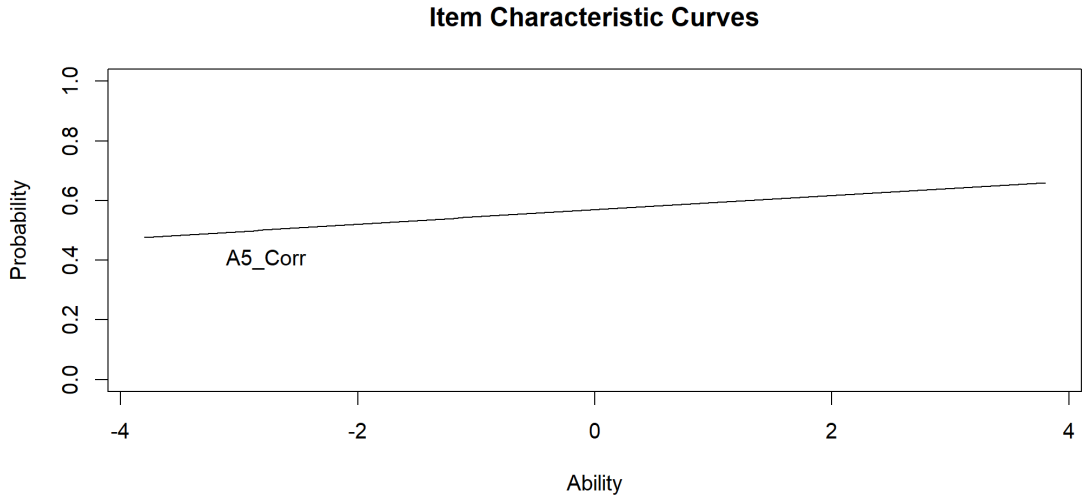


Figure D.31: Item Characteristic Curve for the item A5 included in the array's *state changes* component- *assignment*.

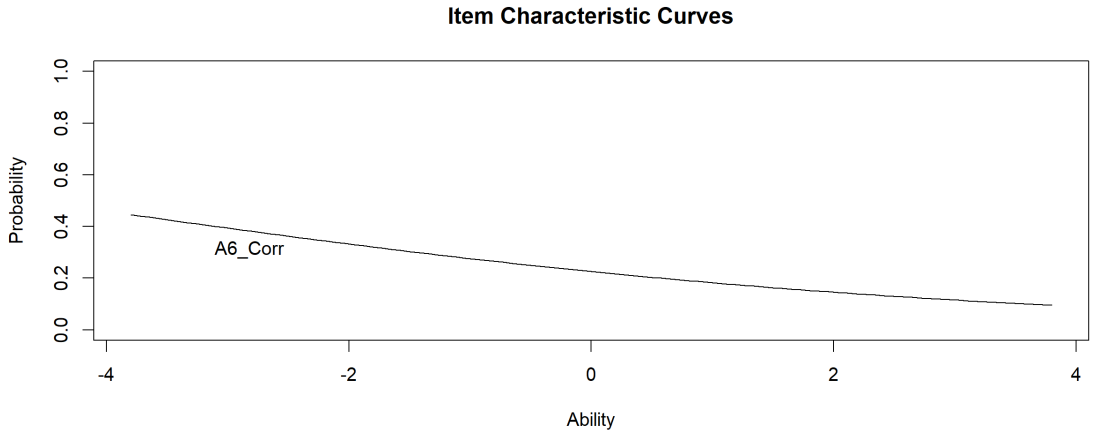


Figure D.32: Item Characteristic Curve for the item A6 included in the array's *state changes* component- *assignment*.