



**UNIVERSIDADE FEDERAL DE CAMPINA GRANDE**  
**CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA**  
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA**  
**COMPUTAÇÃO**

**JUCELIO SOARES DOS SANTOS**

**MEASURING AND FOSTERING COGNITIVE PROGRAMMING**  
**SKILLS IN BEGINNERS**

**CAMPINA GRANDE - PB**

**2023**

**JUCELIO SOARES DOS SANTOS**

**MEASURING AND FOSTERING COGNITIVE PROGRAMMING  
SKILLS IN BEGINNERS**

Tese apresentada ao Programa do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I, pertencente à linha de pesquisa Educação em Ciência da Computação e área de concentração Ciência da Computação, como requisito para obtenção do grau de Doutor em Ciência da Computação.

Orientadores: Prof. Dr. Wilkerson de Lucena Andrade | Prof. Dr. João Arthur Brunet Monteiro.

**CAMPINA GRANDE - PB**

**2023**

S237m Santos, Jucelio Soares dos.  
Measuring and fostering cognitive programming skills in beginners /  
Jucelio Soares dos Santos. – Campina Grande, 2023.  
260 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal  
de Campina Grande, Centro de Engenharia Elétrica e Informática, 2023.  
"Orientação: Prof. Dr. Wilkerson de Lucena Andrade, Prof. Dr. João  
Arthur Brunet Monteiro".  
Referências.

1. Cognitive Programming Skills. 2. Fostering. 3. Bloom's Revised  
Taxonomy. 4. Measurement. 5. Measuring Theories. I. Andrade,  
Wilkerson de Lucena. II. Monteiro, João Arthur Brunet. III. Título.

CDU 004.43(043)



MINISTÉRIO DA EDUCAÇÃO  
**UNIVERSIDADE FEDERAL DE CAMPINA GRANDE**  
POS-GRADUACAO CIENCIAS DA COMPUTACAO  
Rua Aprigio Veloso, 882, - Bairro Universitario, Campina Grande/PB, CEP 58429-900

## FOLHA DE ASSINATURA PARA TESES E DISSERTAÇÕES

JUCÉLIO SOARES DOS SANTOS

MEASURING AND FOSTERING COGNITIVE PROGRAMMING SKILLS IN BEGINNERS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação como pré-requisito para obtenção do título de Doutor em Ciência da Computação.

Aprovada em: 16/01/2023

Prof. Dr. WILKERSON DE LUCENA ANDRADE, Orientador, UFG

Prof. Dr. JORGE CESAR ABRANTES DE FIGUEIREDO, Examinador Interno, UFG

Profa. Dra. ELIANE CRISTINA DE ARAÚJO, Examinadora Interna, UFG

Prof. Dr. SEAN WOLFGAND MATSUI SIQUEIRA, Examinador Externo, UNIRIO

Profa. Dra. PATRICIA CABRAL DE AZEVEDO RESTELLI TEDESCO, Examinadora Externa, UFPE



Documento assinado eletronicamente por **WILKERSON DE LUCENA ANDRADE, PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 17/01/2023, às 09:53, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da [Portaria SEI nº 002, de 25 de outubro de 2018](#).



Documento assinado eletronicamente por **JORGE CESAR ABRANTES DE FIGUEIREDO, PROFESSOR 3 GRAU**, em 17/01/2023, às 12:15, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da [Portaria SEI nº 002, de 25 de outubro de 2018](#).



Documento assinado eletronicamente por **Sean Wolfgang Matsui Siqueira, Usuário Externo**, em 17/01/2023, às 15:16, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da [Portaria SEI nº 002, de 25 de outubro de 2018](#).





Documento assinado eletronicamente por **ELIANE CRISTINA DE ARAUJO, PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 19/01/2023, às 10:18, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da [Portaria SEI nº 002, de 25 de outubro de 2018](#).

---



A autenticidade deste documento pode ser conferida no site <https://sei.ufcg.edu.br/autenticidade>, informando o código verificador **3030263** e o código CRC **A1EB55D7**.

---

Referência: Processo nº 23096.000907/2023-57

SEI nº 3030263

## Resumo

As instruções atuais para ensinar habilidades cognitivas de programação apresentam lacunas em identificá-las, estruturá-las e sequenciá-las. Os novatos no Curso de Introdução à Programação (CS1) geralmente têm níveis diferentes de conhecimento prévio e habilidades de resolução de problemas amplamente variadas. Alunos que possuem algum contato prévio com programação em estágios anteriores ao CS1 podem apresentar mais facilidade de assimilar o conteúdo. Por outro lado, alunos que não tiveram essa experiência podem apresentar dificuldades no aprendizado e devem receber mais atenção dos educadores. Em geral, esse aprendizado por parte de alunos com diversos níveis de conhecimento é impactado por variados níveis cognitivos até então pouco explorados. A falta de uma correta compreensão desses níveis e a escassez de instrumentos confiáveis e válidos para um atendimento personalizado podem ter sérias implicações no ambiente de ensino em CS1. Cerca de um terço dos alunos matriculados no CS1 geralmente acabam reprovando ou desistindo. Este fato induz a desmotivação nos alunos, e a desconfiança destes cursos superiores pela comunidade acadêmica. Assim, é essencial preencher as lacunas de conhecimento sobre a identificação/segmentação de quais habilidades cognitivas estão envolvidas no aprendizado de programação, bem como, propor instrumentos confiáveis para medi-la e fomentá-las. Desta forma, esta pesquisa tem como objetivo identificar, medir e fomentar habilidades cognitivas em iniciantes em programação por meio de um instrumento confiável, adaptativo e empiricamente válido. Este instrumento determina o nível de desafio apropriado de acordo com o nível de habilidade do aluno. Para tanto, identificamos habilidades cognitivas de programação e as abordagens para promover/medir tais habilidades. Assumimos que sequenciar as habilidades cognitivas envolvidas no aprendizado da programação por meio do Domínio Cognitivo da Taxonomia Revisada de Bloom determina o nível de desafio correto em um instrumento de avaliação. Por meio desta abordagem, criamos um banco de itens e analisamos os conteúdo e semântica desses itens. Por meio das Teorias de Mensuração, calibramos o banco de itens e avaliamos a consistência interna do instrumento. Além disso, analisamos a relação entre as habilidades cognitivas de programação e a capacidade do participante em produzir código. Integramos a seleção adaptativa ao instrumento criado a fim de melhorar a seleção dos itens e a estimativa

das habilidades dos participantes. Por fim, investigamos se a promoção de habilidades cognitivas de programação melhora o desempenho da escrita de código para iniciantes. Como resultados, fornecemos uma abordagem para sequenciar as habilidades cognitivas de programação a fim de promovê-las e medi-las. Oferecemos instrumentos confiáveis e adaptativos, que fomentam e medem habilidades cognitivas em novatos em programação de forma incremental. Além disso, obtivemos evidências empíricas sobre a influência das habilidades cognitivas promovidas pelos instrumentos sobre o desempenho dos participantes na escrita de código. Os resultados nos dão indícios que o aprendizado pode ocorrer de forma linear à medida que os alunos avançam nos níveis cognitivos na Taxonomia Revisada de Bloom. Concluímos que o desempenho cognitivo de iniciantes em programação bem sucedidos, em tarefas de escrita de código, pode estar interligada com as habilidades fomentadas pelo instrumento. Além disso, a escolha de itens administrados pelo instrumento adaptativo determina uma redução na quantidade de itens e uma sequência adequada para determinar o nível de habilidade do sujeito. Os resultados desta pesquisa podem contribuir para a prática de ensino e aprendizagem em CS1, por fornecer um instrumento que permitirá aos professores de programação coletar evidências de dificuldades iniciais de programação de forma eficiente. Os professores podem fornecer atendimento personalizado indicando questões de acordo com o nível de habilidade do aluno para compensar suas dificuldades.

**Palavras-chave:** Habilidades Cognitivas de Programação. Fomentação. Mensuração. Taxonomia Revisada de Bloom. Teoria de Mensuração.

## **Abstract**

The current instructions to teach cognitive programming skills have gaps in identifying, structuring and sequencing them. Novices in the Introduction to Programming Course (CS1) often have different levels of prior knowledge and widely varying problem-solving skills. Students who have some previous contact with programming in stages prior to CS1 may find it easier to assimilate the content. On the other hand, students who have not had this experience may have learning difficulties and should receive more attention from educators. In general, the learning of students with different levels of knowledge is impacted by several cognitive levels, which, until now, has been little explored. A lack of the correct understanding of these levels and the need for more reliable and valid instruments for personal assistance can have serious implications in the teaching environment in CS1. About one-third of the students enrolled in CS1 usually end up failing or dropping out. This fact leads to students' demotivation and distrust of these higher education courses by the academic community. Thus, it is essential to fill in the gaps in the knowledge about the identification/segmentation of which cognitive skills are involved in programming learning and propose reliable instruments to measure and foster them. Thus, this research aims to identify, measure, and foster cognitive skills in novices in programming through a reliable, adaptive, and empirically valid instrument. This instrument determines the appropriate challenge level according to the student's skill level. To do so, we identify cognitive programming skills and approaches to foster/measure such skills. We assume that sequencing the cognitive skills involved in programming learning through the Cognitive Domain of Bloom's Revised Taxonomy determines the correct challenge level in an assessment instrument. Through this approach, we created an items bank and analyzed the content and semantics of these items. Using Measurement Theories, we calibrated the items bank and assessed the instrument's internal consistency. In addition, we analyzed the relationship between cognitive programming skills and the participant's ability to produce code. We integrated adaptive selection into the created instrument to improve the item selection and the estimation of participants' abilities. Finally, we investigated whether fostering cognitive programming skills improves code writing performance for novices. As a result, we provide an approach to sequencing cognitive programming skills

to foster and measure them. We offer reliable and adaptive instruments that incrementally foster and measure cognitive skills in programming novices. In addition, we obtained empirical evidence on the influence of the cognitive skills fostered by the instruments on the participants' performance in writing code. The results indicate that learning can occur linearly as students advance in cognitive levels in Bloom's Revised Taxonomy. The cognitive performance of successful novice programmers in code writing tasks may be interconnected with the skills fostered by the instrument. In addition, the item choice managed by the adaptive instrument determines a reduction in the number of items and an adequate sequence to determine the subject's skill level. These research results can contribute to teaching and learning practice in CS1 by providing an instrument that will allow programming educators to collect evidence of initial programming difficulties efficiently. Educators can assist by assigning questions according to the student's skill level to compensate for difficulties.

**Keywords:** Cognitive Programming Skills. Fostering. Measurement. Bloom's Revised Taxonomy. Measurement Theories.

*For our present troubles are small and won't last very long. Yet they produce for us a glory that vastly outweighs them and will last forever! So we don't look at the troubles we can see now; rather, we fix our gaze on things that cannot be seen. For the things we see now will soon be gone, but the things we cannot see will last forever.*

**2 Corinthians 4:17-18.**

## **Acknowledgements**

To God, for allowing me to be who I am, the way I am.

To my professors Wilkerson de Lucena Andrade, João Arthur Brunet Monteiro and Monilly Ramos Araujo Melo for their guidance and availability throughout this work, contributing to the success of my academic and personal training.

I would like to thank the Software Practice Laboratory, the State University of Paraiba and the Federal University of Campina Grande for the logistical conditions for carrying out this study. I also want to thank my PhD colleagues at the Federal University of Campina Grande for their contribution and instructions throughout this work.

To all, my deepest thanks.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contextualization . . . . .	1
1.2	Problem . . . . .	2
1.3	Objectives . . . . .	5
1.4	Methodology . . . . .	6
1.5	Related Works . . . . .	9
1.6	Contributions . . . . .	11
1.7	Thesis' Outline . . . . .	12
<b>2</b>	<b>Background</b>	<b>15</b>
2.1	Competency Model . . . . .	15
2.2	Taxonomies . . . . .	17
2.2.1	Bloom's Revised Taxonomy . . . . .	17
2.2.2	SOLO Taxonomy . . . . .	22
2.3	Measurement Theory . . . . .	23
2.3.1	Classical Test Theory . . . . .	24
2.3.2	Item Response Theory . . . . .	27
2.4	Computerized Adaptive Testing . . . . .	35
2.4.1	Step 1—Instrument Definition . . . . .	36
2.4.2	Step 2—Item Bank Preparation . . . . .	37
2.4.3	Step 3—Item Bank Calibration . . . . .	39
2.4.4	Step 4—Algorithm Elaboration . . . . .	41
2.4.5	Step 5—Instrument Accuracy and Validity Analysis . . . . .	43
2.4.6	Step 6—Instrument Implementation . . . . .	45



---

2.4.7	Step 7—Instrument Application . . . . .	46
2.4.8	Step 8—Instrument Maintenance . . . . .	47
<b>3</b>	<b>Cognitive Programming Skills</b>	<b>49</b>
3.1	Initial Considerations . . . . .	49
3.2	Study Design . . . . .	50
3.2.1	Search Strategy . . . . .	50
3.2.2	Study Selection Strategy . . . . .	51
3.2.3	Data Extraction . . . . .	52
3.2.4	Threats to Validity . . . . .	53
3.3	Results . . . . .	54
3.3.1	What are the Cognitive Programming Skills? . . . . .	54
3.3.2	How to Measure Cognitive Programming Skills? . . . . .	54
3.3.3	How to Foster Cognitive Programming Skills? . . . . .	55
3.4	Discussion . . . . .	56
3.4.1	Cognitive Programming Skills . . . . .	57
3.4.2	Measuring Cognitive Programming Skills . . . . .	63
3.4.3	Fostering Cognitive Programming Skills . . . . .	63
3.5	Related Works . . . . .	69
3.6	Final Considerations . . . . .	70
<b>4</b>	<b>A Cognitive Domain Adaptation to Bloom’s Revised Taxonomy</b>	<b>71</b>
4.1	Initial Considerations . . . . .	71
4.2	Related Works . . . . .	72
4.3	Adapting the Cognitive Domain of Bloom’s Revised Taxonomy . . . . .	74
4.3.1	Remember . . . . .	75
4.3.2	Understand . . . . .	75
4.3.3	Apply . . . . .	75
4.3.4	Analyse . . . . .	76
4.3.5	Evaluate . . . . .	77
4.3.6	Create . . . . .	77
4.4	Design . . . . .	78

---

4.4.1	Participants . . . . .	78
4.4.2	Techniques and Metrics . . . . .	79
4.4.3	Data Analysis . . . . .	79
4.4.4	Methodological Process . . . . .	81
4.4.5	Threats to Validity . . . . .	82
4.5	Results . . . . .	82
4.6	Discussions . . . . .	84
4.6.1	Remember . . . . .	84
4.6.2	Understand . . . . .	85
4.6.3	Apply . . . . .	87
4.6.4	Analyse . . . . .	87
4.6.5	Evaluate . . . . .	88
4.6.6	Create . . . . .	89
4.7	Final Considerations . . . . .	90
<b>5</b>	<b>An Instrument to Measure and Foster Cognitive Programming Skills</b>	<b>91</b>
5.1	Initial Considerations . . . . .	91
5.2	Instrument Development . . . . .	92
5.2.1	Instrument Definition . . . . .	92
5.2.2	Item Bank Building . . . . .	93
5.3	Study Design . . . . .	110
5.3.1	Participants . . . . .	110
5.3.2	Techniques and Metrics . . . . .	110
5.3.3	Data Analysis . . . . .	111
5.3.4	Methodological Process . . . . .	111
5.3.5	Threats to Validity . . . . .	112
5.4	Results . . . . .	112
5.5	Discussions . . . . .	114
5.6	Related Works . . . . .	115
5.7	Final Considerations . . . . .	116

---

<b>6</b>	<b>Assessing the Instrument's Reliability</b>	<b>117</b>
6.1	Initial Considerations . . . . .	117
6.2	Design . . . . .	119
6.2.1	Participants . . . . .	119
6.2.2	Data Analysis . . . . .	119
6.2.3	Methodological Process . . . . .	119
6.2.4	Threats to Validity . . . . .	121
6.3	Results . . . . .	121
6.3.1	Items' Psychometric Properties . . . . .	121
6.3.2	Internal Consistency . . . . .	123
6.3.3	Factor Correlation Analysis . . . . .	124
6.4	Discussions . . . . .	126
6.4.1	Does the Instrument Have Items With Appropriate Psychometric Properties? . . . . .	126
6.4.2	Does the Instrument Have Appropriate Internal Consistency? . . . . .	127
6.4.3	Which of the Cognitive Programming Skills Present in the Instrument Has a Strong Correlation With the Producing Skill? . . . . .	127
6.5	Related Works . . . . .	128
6.6	Final Considerations . . . . .	129
<b>7</b>	<b>Integrating Adaptive Selection to the Instrument</b>	<b>130</b>
7.1	Initial Considerations . . . . .	130
7.2	Adaptive Algorithms . . . . .	132
7.2.1	Adaptive Selection Algorithm Z1 . . . . .	137
7.2.2	Adaptive Selection Algorithm Z2 . . . . .	139
7.2.3	Adaptive Selection Algorithm Z3 . . . . .	141
7.3	Design . . . . .	143
7.3.1	Dependent and Independent Variables . . . . .	143
7.3.2	Data Analysis . . . . .	143
7.3.3	Methodological Process . . . . .	143
7.3.4	Threats to Validity . . . . .	144

---

7.4	Results . . . . .	145
7.5	Discussions . . . . .	147
7.6	Related Works . . . . .	147
7.7	Final Considerations . . . . .	148
<b>8</b>	<b>Assessing the Instrument Validity</b>	<b>149</b>
8.1	Initial Considerations . . . . .	149
8.2	Design . . . . .	150
8.2.1	Participants . . . . .	150
8.2.2	Dependent and Independent Variables . . . . .	152
8.2.3	Data Analysis . . . . .	152
8.2.4	Methodological Process . . . . .	152
8.2.5	Validity Threats . . . . .	154
8.3	Results . . . . .	155
8.4	Discussions . . . . .	156
8.5	Related Works . . . . .	157
8.6	Final Considerations . . . . .	158
<b>9</b>	<b>Conclusions and Future Work</b>	<b>159</b>
9.1	Conclusions . . . . .	159
9.2	Future Work . . . . .	161
<b>A</b>	<b>Consent Term—SPLab</b>	<b>179</b>
<b>B</b>	<b>Consent Term—LabNEUROKIT</b>	<b>181</b>
<b>C</b>	<b>Partnership Term</b>	<b>183</b>
<b>D</b>	<b>Researchers' Commitment Agreement Term</b>	<b>185</b>
<b>E</b>	<b>Results Disclosure Commitment' Term</b>	<b>187</b>
<b>F</b>	<b>Bloom's Revised Taxonomy for Measuring and Fostering Cognitive Programming Skills</b>	<b>189</b>

---

<b>G</b>	<b>Inspection of Bloom's Revised Taxonomy for Measuring and Fostering Cognitive Programming Skills</b>	<b>205</b>
<b>H</b>	<b>Informed Consent Form</b>	<b>207</b>
<b>I</b>	<b>Item Bank Inspection</b>	<b>209</b>
<b>J</b>	<b>Informed Consent Form</b>	<b>211</b>
<b>K</b>	<b>Items Psychometric Properties</b>	<b>213</b>
<b>L</b>	<b>Items Characteristic Curves</b>	<b>233</b>
<b>M</b>	<b>Items Information Function</b>	<b>244</b>
<b>N</b>	<b>Script for Adaptive Item Selection and Skill Estimation</b>	<b>255</b>
<b>O</b>	<b>Trust in Python</b>	<b>259</b>

# List of Symbols

1PL—*One-Parameter Logistic Model*

2PL—*Two-Parameter Logistic Model*

3PL—*Three-Parameter Logistic Model*

a—*Slope*

b—*Threshold*

c—*Asymptote*

CAT—*Computerized Adaptive Testing*

CS—*Computer Science*

CS1—*Introduction to Programming Course*

CTT—*Classical Test Theory*

EAP—*Expected A Posteriori*

ICC—*Item Characteristic Curve*

ICF—*Informed Consent Term*

IIF—*Item Information Function*

IRT—*Item Response Theory*

K-S-D—*Knowledge-Skill-Disposition*

LabNEUROGIT—*Laboratório de Neuropsicologia Cognitiva e Inovação Tecnológica*

RQ—*Research Question*

SLM—*Systematic Literature Mapping*

SOLO—*Structure of the Observed Learning Outcome*

SPLab—*Software Practices Laboratory*

UEPB—*State University of Paraíba*

UFCG—*Federal University of Campina Grande*

# List of Figures

1.1	Thesis' Outline. . . . .	13
2.1	Conceptual Structure of the CC2020 Competency Model [24]. . . . .	16
2.2	ICC's Example. . . . .	30
2.3	IIF's Example. . . . .	34
2.4	Computerized Adaptive Testing steps. . . . .	36
2.5	Instrument Definition Step. . . . .	37
2.6	Item Bank Preparation Step. . . . .	38
2.7	Item Bank Calibration Step. . . . .	39
2.8	Algorithm Elaboration Step. . . . .	42
2.9	Accuracy and Validity Analysis Step. . . . .	44
2.10	Instrument Implementation Step. . . . .	45
2.11	Instrument Application Step. . . . .	46
2.12	Instrument Maintenance Step. . . . .	47
4.1	Adapting the Cognitive Domain of Bloom's Revised Taxonomy. . . . .	74
4.2	Methodological Process of Adaptation Stage of the Cognitive Domain of Bloom's Revised Taxonomy. . . . .	81
4.3	Degree of Agreement Between the Judges in the Skills Adaptation. . . . .	83
4.4	Degree of Agreement Between the Judges in the Skills Association. . . . .	84
5.1	Example of the Item Recognizing Skill. . . . .	95
5.2	Example of the Item Recalling Skill. . . . .	96
5.3	Example of the Item Interpreting Skill. . . . .	97
5.4	Example of the Item Exemplifying Skill. . . . .	98

---

5.5	Example of the Item Classifying Skill. . . . .	99
5.6	Example of the Item Summarizing Skill. . . . .	99
5.7	Example of the Item Inferring Skill. . . . .	100
5.8	Example of the Item Comparing Skill. . . . .	101
5.9	Example of the Item Explaining Skill. . . . .	101
5.10	Example of the Item Executing Skill. . . . .	102
5.11	Example of the Item Implementing Skill. . . . .	103
5.12	Example of the Item Differentiating Skill. . . . .	104
5.13	Example of the Item Organizing Skill. . . . .	105
5.14	Example of the Item Attributing Skill. . . . .	105
5.15	Example of the Item Checking Skill. . . . .	106
5.16	Example of the Item Critiquing Skill. . . . .	107
5.17	Example of the Item Generating Skill. . . . .	108
5.18	Example of the Item Planning Skill. . . . .	109
5.19	Example of the Item Producing Skill. . . . .	109
5.20	Methodological Process of the Item Bank Semantic and Content Evaluation Stage. . . . .	111
6.1	Methodological Process of Instrument Reliability Assessment. . . . .	120
6.2	Recognizing Skill ICC's With Ten First Items Calibrated. . . . .	122
6.3	Recognizing Skill IIF's With Ten First Items Calibrated. . . . .	123
6.4	Boxplot in the Pre-test With the Estimate of the Subjects' Skills. . . . .	124
6.5	Correlation Heat Map Between the Skills. . . . .	125
7.1	Criteria for Elaborating Algorithms Targeting the Selection of Adaptive Items. . . . .	132
7.2	Item Frequency Boxplot Applied Across All Skills for the Z3. . . . .	139
7.3	Methodological Process of Analyzing the Integration of Adaptive Selection to the Instrument. . . . .	144
8.1	Boxplot Between Groups of How Many Users Master the Python Language. . . . .	151
8.2	Methodological Process of Assessing the Instrument Validity. . . . .	153
8.3	Boxplot of Groups' Performance in the Ability to Produce Codes. . . . .	155



# List of Tables

2.1	Bloom’s Taxonomy’s Three Domains’ Features. . . . .	18
2.2	Verbs Used in Code Comprehension Tasks. . . . .	21
3.1	Terms, Keywords, and Synonyms Used to Create the Search Query. . . . .	51
3.2	Description of Extracting Data From SLM. . . . .	52
3.3	Details of Study Search and Selection by the Database. . . . .	53
3.4	Search Phase Study Details. . . . .	53
3.5	Cognitive Programming Skills Identified in the Literature. . . . .	54
3.6	Instruments That Measure Cognitive Programming Skills. . . . .	55
3.7	Methods, Approaches, and Tools That Foster Cognitive Programming Skills. . . . .	56
4.1	Adaptation of the Level Remember and Cognitive Programming Skills Association. . . . .	75
4.2	Adaptation of the Level Understand and Cognitive Programming Skills Association. . . . .	76
4.3	Adaptation of the Level Apply and Cognitive Programming Skills Association. . . . .	76
4.4	Adaptation of the Level Analyse and Cognitive Programming Skills Association. . . . .	77
4.5	Adaptation of the Level Evaluate and Cognitive Programming Skills Association. . . . .	78
4.6	Adaptation of the Level Create and Cognitive Programming Skills Association. . . . .	78
4.7	Intraclass correlation coefficient in adapting of skills. . . . .	83
4.8	Intraclass Correlation Coefficient in the Association of Skills. . . . .	84
5.1	Skills Investigated and Proposed in the Instrument. . . . .	93

---

5.2	Total Items Contained in the Instrument. . . . .	94
5.3	Reliability Analysis of Item Content Among Judge. . . . .	113
5.4	Reliability Analysis of Item Semantics Among Judges. . . . .	113
6.1	Recognizing Skill—Ten First Items Calibrated. . . . .	122
6.2	Internal Consistency of the Instrument. . . . .	123
6.3	Exploratory Data Analysis of the Correlation Between the Skills. . . . .	125
7.1	Minimum and the Maximum Number of Items for All Skills. . . . .	139
7.2	Descriptive Statistics of Abilities' Estimation Among Algorithms. . . . .	145
7.3	Normality Test of Skills Estimates Between Algorithms. . . . .	146
7.4	Correlation of Skills Estimation Between Eap and Adaptive Algorithms. . .	146
8.1	Cross-Group Analysis of How Many Users Master the Python Language. .	151
8.2	Analysis of the Group's Performance in the Ability to Produce Codes. . . .	156

# Chapter 1

## Introduction

In this chapter, we present the Introduction to Programming Course (CS1) as the main research topic. In Section 1.1, we provide a brief background on measuring and fostering cognitive skills in CS1. In Section 1.2, we discuss the search problem. In Section 1.3, we present the objectives. In Section 1.4, we describe the research methodology, presenting the research phases and questions. In Section 1.5, we present the related work. In Section 1.6, we present the contribution of this doctoral research. Finally, Section 1.7 presents an overview of this thesis's organization.

### 1.1 Contextualization

Computer Science (CS) programs in Higher Education have programming courses as their central focus. One of the main courses is called CS1. CS1 can be challenging for novices because it conventionally covers problem-solving skills, basic programming concepts, syntax, and semantics of a programming language to formulate solutions [3].

Despite advanced teaching and learning practices in CS1, some instructors still adopt traditional teaching practices. In this model, teaching in CS1 consists of theoretical classes, pseudocode examples, and subsequent resolution of difficult-to-assimilate problems. The instructor focuses on a specific series of commands in a programming language. The instructor selects exercises as examples, often presenting them in a non-practical context. This practice may be challenging to create images and mental models that help students build algorithms [85].

In addition to this teaching model, learners start CS1 with a wide range of skill levels, particularly their prior knowledge of programming and problem-solving skills [109]. In the early stages of a programming course, low-achieving students find programming difficult [125]. Generally, learners need help during problem analysis, planning, and solutions design [121].

Students need help to analyze the flow of a program. In addition, they need help to correct errors and debug programs, often relying on help from other colleagues to solve problems and correct errors. The most critical activity for learners is understanding how to create their programs while learning how to program [109]. Therefore, students write programs after learning syntax rules and some examples. However, educators need to emphasize precursor skills for writing code.

Even after going through CS1, studies show that learners still have serious problems applying some programming concepts, considering it a complex task [77]. Studies also show that most learners have their programming knowledge consolidated only at the end of the second programming course [77, 109]. For this reason, programming teaching is considered one of the most significant challenges in Computing Education [77, 85].

## 1.2 Problem

Learning a programming language requires a basic understanding of constructs and syntax and synthesizing to apply those constructs in new ways. These components tend to be treated indiscriminately by educators [125]. For example, to draw a parallel with another field, imagine being given basic instructions on using scissors and, simultaneously, being asked to use the scissors to construct a dress skillfully. For many learners, acquiring and applying knowledge in rapid succession presents a different challenge than those found in less applied courses [109].

Students who have previous contact with the CS field before entering a CS1 course may find it easier to assimilate the content. For students with no previous contact with CS, the effort of the new programming language syntax and developing algorithms for unknown problems should be given more attention by educators. Therefore, students need to master programming construction and problem-solving knowledge. The educator's challenge is to

provide enough support for inexperienced students in one or both domains to reduce learning difficulties [109, 121].

In programming learning, a significant factor affecting learning comes from adopting new programming constructs [138]. Introductory languages tend to have similar sets of functional constructs to learn, including (but not limited to) variables and arithmetic, assignment, selection, repetition, functions, and data structures. Each of these concepts will involve learning syntax and usage. Unfortunately, the syntax presents complexities that take time for students to master, for example, the difficulties with the variables' declaration. A student must include a keyword representing the variable's data type before providing the variable name during the declaration in many introductory programming languages, such as C or Java. In later uses, adding the data type will reallocate the memory associated with the variable. It is a pattern shift that the learner must understand before they can write code freely. In addition to the syntactic rules, the learner must understand each new construction's functionalities, restrictions, and resources [109, 125].

Individually, syntax and usage will contribute to student learning. In combination, the potential to overwhelm a student is high. Syntax and functionality exist for learners in other contexts—mathematical expression or composition; however, the novice programmer must take extra care. Learners often use new programming constructs with new problems soon after the educator presents them. The task can be daunting if the learner cannot handle the syntax and programming functionality while involved in the problem-solving process [125]. Students with previous programming and problem-solving experience may not feel the CS1 impact. Their experience might help them overcome obstacles in learning the syntax and semantics of the programming language. Novices can only sometimes overcome these initial challenges.

In addition to basic knowledge of program constructs, programming requires procedural skills to perform tasks with these constructs. Learning to program also requires learning to apply multiple skills, and CS1 instruction lacked adequate instruction in these skills [137]. In traditional teaching, educators focus on writing code and superficially promote other skills.

Several research studies support the premise that people would learn to program more effectively and efficiently if they spent more time decoding code (reading, tracing, and

debugging) than writing code [31, 51, 71, 72, 115, 120, 137]. The learning progression <sup>1</sup> starts with low-risk deconstruction activities like exploring, identifying, comparing, and debugging before activities that require writing code [51].

In addition, another problem found in the traditional teaching approach is a possible consequence of the need for more knowledge about the diversity of cognitive levels. The traditional approach forces students to follow a learning pace imposed by educators. This condition negatively influences those who cannot fully master the content and those who already know how to program and need to wait for the rest of the class. On the other hand, when it does not allow learners with difficulties to have time to master the content, it increases their differences from more advanced colleagues [88].

To address these needs and make learners adopt more effective study methodologies, programming educators must make decisions that guide their pedagogical practices. The educators must organize instruction and learning activities that help students obtain the necessary knowledge for the course (going beyond the content exposition in the classroom). It is necessary to continuously use specific educational tools that give example concepts and bring dynamic explanations of the theory when the student is studying without the educator's supervision [10].

Adopting these practices and offering learning conditions to students according to their pace, learners must periodically ask questions indicated by the educator in the learning activity. Depending on the subject's limitations, the tutor/monitor proposes questions to reinforce learning. However, the questions' sequence/number may not suit the learner's skill level, so educators need to revise them. In the assessment, educators measure the learner's performance using instruments that may contain questions that need to be more discriminating between learners with low/high ability [88].

Indicating and proposing a question is a complex task, which may lead to several problems related to the motivation or demotivation of students. It is necessary to consider the learner's skill level to address tasks properly. If the item is too complex, instead of fostering/reinforcing learning/support, it may discourage learners with low ability. If the

---

<sup>1</sup>Learning progressions describe how the skills might be demonstrated, both in their early forms and increasingly advanced forms. Educators must be able to identify the behaviors that relate to these skills if they are to intervene at the appropriate levels in the learning [99].

item is too easy for her level, it may be boring for high-ability learners. Suppose the item offers little information to the skill estimates. In that case, it may influence the learner to give up practicing the exercises. The exact process occurs for assessment activities. A small/very discriminating item can overestimate or underestimate the individual's ability, directly influencing the subject's failure rate [8, 14, 101].

Given this scenario, we have the following problems: how do we identify cognitive skills in programming? Is it possible to sequence them? How to measure them? Moreover, how to foster them? Cognitive skills in programming are those introduced in developing a more structured skill, that is, skills that must be measured and fostered in CS1 to make the subject skilled in writing code. To achieve this goal, we need to know which cognitive skills are involved in this process and how to organize them sequentially. The sequence of cognitive programming skills involves a process so that knowledge of each skill can be demonstrated and built upon knowledge of previous skills. So, this sequence directly affects instructional design in CS1. Finally, once we know what cognitive skills in programming are and how to sequence them, we must conduct an instructional design to measure and foster them in CS1.

However, we need a consensus on how institutions should work on cognitive skills in CS1. We still need a solution to identify and sequence the cognitive skills in writing code. We need to expand the development of reliable instruments to foster and measure such skills to identify students' limitations in CS1 [78, 85]. Reliable instruments present items that are well understood and suitable for measuring the desired skill [8]. Therefore, there is a need for research that provides a better understanding of cognitive programming skills and an investigation into approaches to foster and measure these skills, allowing educators to identify early failures of novices in CS1.

## 1.3 Objectives

This research aims to identify, measure, and foster novice programmers' cognitive skills through a reliable, adaptive, and empirically valid instrument. This instrument determines the appropriate challenge level according to the learner's skill level. We defined the following specific objectives to accomplish this general research objective:

- **SO1.** To identify cognitive programming skills;

- **SO2.** To identify approaches to measure cognitive programming skills;
- **SO3.** To identify approaches to foster cognitive programming skills;
- **SO4.** To sequence cognitive programming skills to determine the appropriate challenge level in an assessment instrument;
- **SO5.** To develop an item bank with content and semantics analysis that include cognitive programming skills' indicators;
- **SO6.** To calibrate the item bank in the information terms they provide regarding the specific psychological construct assessed;
- **SO7.** To evaluate the internal consistency of the item bank through measurement theories;
- **SO8.** To investigate which cognitive programming skills are relevant to the participant's ability to program;
- **SO9.** To integrate adaptive selection through algorithms that improve performance in estimating students' abilities;
- **SO10.** To investigate whether fostering cognitive programming skills improves novice code-writing performance.

## 1.4 Methodology

This research is a partnership between the Software Practices Laboratory (SPLab) (Appendix A) and the Laboratório de Neuropsicologia Cognitiva e Inovação Tecnológica (LabNEUROKIT) (Appendix B). This research (Appendices C, D, E) proposes to automate approaches from an instrument that raises awareness of introductory programming difficulties and aims to operationalize the knowledge of work rules and expert reasoning.

The research is **applied**, as it involves local interests [48]. This research aimed to generate knowledge for practical application and aimed at solving specific problems. As for the objective, this research is **explanatory** [118], as we described and identified skills that determine or contribute to a better performance in novice programmers. As an approach, this



research is **quantitative** [79] because we measured/classified/analyzed programming skills and, therefore, it was necessary to use statistical resources and techniques.

At the end of this doctoral research, we investigated whether improving current practices in measuring and fostering cognitive skills in novice programmers is possible. We divided this research into six phases:

- **Phase 1.** Through a Systematic Literature Mapping (SLM), we identified the cognitive programming skills and the existing strategies to measure and foster these skills. This SLM included studies on cognitive programming skills and survey/categorization that measure and foster such skills. At this phase, we have achieved the following specific objectives: **SO1**, **SO2**, and **SO3**. We investigated the following research questions (RQs):
  - **RQ1.** What are the cognitive programming skills?
  - **RQ2.** How to measure cognitive programming skills?
  - **RQ3.** How to foster cognitive programming skills?
- **Phase 2.** We adapted Bloom's Revised Taxonomy cognitive domain for programming teaching, sequencing the cognitive programming skills in this adaptation. Then, we developed and applied a **survey** to an expert group in CS1 to theoretically evaluate this adaptation. At this phase, we have achieved the following specific objective: **SO4**. We investigated the following research question:
  - **RQ4.** How to sequence the cognitive skills involved in introductory programming learning to determine the appropriate challenge level in an assessment instrument?
- **Phase 3.** We designed an instrument based on the adaptation Cognitive Domain of Bloom's Revised Taxonomy for programming teaching. Then, we designed and applied a **survey** to advanced programmers, educators, and CS1 experts to assess the item's semantics and content in the instrument. We have achieved this phase's specific objective: **SO5**. We investigated the following research questions:
  - **RQ5.** Does the instrument have items with appropriate semantic analysis?

- **RQ6.** Does the instrument have items with appropriate content analysis?
- **Phase 4.** We applied the instrument to a novice programmer's sample. Moreover, we calibrated the items bank present in the instrument to obtain information about their psychometric properties. The item bank calibration consists of applying the items, collecting data, choosing the response model and the calibration method, and preparing and interpreting the scale. We evaluated the instrument's internal consistency through the Measurement Theories. Finally, we investigated the relationship between cognitive programming skills and the participant's ability to program. We have achieved the following specific objectives at this phase: **SO6**, **SO7**, and **SO8**. We investigated the following research questions:
  - **RQ7.** Does the instrument have items with appropriate psychometric properties?
  - **RQ8.** Does the instrument have appropriate internal consistency?
  - **RQ9.** Which of the cognitive programming skills present in the instrument has a strong correlation with the participant's ability to program?
- **Phase 5.** We presented the adaptive selection's integration into the assessment instrument to improve the measurement of cognitive programming skills. We have achieved this phase's specific objective: **SO9**. We investigated the following research question:
  - **RQ10.** Which adaptive algorithm has better accuracy in estimating cognitive programming skills?
- **Phase 6.** We presented an exploratory investigation into the instrument's validity for measuring introductory programming skills. We have achieved this phase's specific objective: **SO10**. We investigated the following research question:
  - **RQ11.** Does the instrument that fosters cognitive programming skills improve the code-writing performance of novices?

## 1.5 Related Works

In this section, we report primary studies that analyze patterns of relationship/grouping between different cognitive skills investigated to improve the teaching and learning process in CS1. Among the main works, the following approaches/theories stand out: Seven steps [56], Cognitive abilities to improve CS1 outcome [4], and instructional theory for introductory programming skills [137].

Seven steps foster structure and guidance on how to approach a problem. The first four steps focus on creating an English algorithm, and then the remaining steps are to translate that algorithm into code, test the algorithm and debug failed test cases. This method gives students a way to solve problems and ideas about what to do if they get stuck during the process. Furthermore, it provides a way for instructors to work with examples in class that focus on the code-writing process. Educators show how to create the code rather than show an example of the code [56].

Other authors summarized their findings and listed cognitive skills in programming [4]. Cognitive skills in programming can translate into the existence or not of underlying cognitive skills, which may be responsible for different perceptions about the difficulty of programming phases. The skills highlighted by the authors are: i) reading and understanding the problem; ii) resolving an instance of the problem manually; iii) generalizing the solution; iv) developing an algorithm that solves the problem; v) algorithm simulation; vi) translating the algorithm into a programming language; vii) compiling; and viii) test.

Xie et al. [137] created an instructional theory for introductory programming skills. This theory proposes the development of specific and incremental skills to avoid overloading students. These skills include: reading semantics, writing semantics, reading models, and writing models. First, students develop language-related skills by tracking and writing the correct syntax. Then, students start using the models, understanding common patterns in the code, and writing programs that use those models. The most notable part of this theory is different from the four specific skills. However, the emphasis is on being rigorous and explicit on how skills are defined and ordered in instructions.

- *Reading Semantics*: refers to the ability to precisely control code and predict the effect of syntax on program behavior. Reading semantics requires the student to trace the

code and does not require other skills. This practice consists of examining hard-coded questions in which the student determines the intermediate and final program states for a predefined piece of code;

- *Writing Semantics*: after practicing reading semantics and receiving feedback from the correct solution and an explanation, students begin to learn to write the correct syntax. Writing semantics refers to translating unambiguous natural language descriptions of language constructs into syntax that will compile and run as expected;
- *Reading Templates* after learning to read and write semantics for a new construct, a student then transitions to using standard code usage pattern models to apply knowledge of this construct. The reading model identifies reusable abstractions of programming knowledge (which we will refer to as models) and maps them to a goal. The theory taught four models: variable swapping, digit processing, floating equality, and max./min. Instructions for reading the template usually start with an example or visualization to make the template objects and steps more concrete;
- *Writing Templates* requires the student to begin with an ambiguous description of the problem. Then identify a model they can use to solve the problem and implement each component of the model in code. In parallel, the write semantics instruction specifies rules to avoid syntactic errors. In this way, the write model statement specifies rules to avoid logical errors that would be syntactically correct but would result in code that would not work with the model specification.

Based on this theory, the authors [137] have created learning materials for a subset of programming concepts. The materials included instructional content, hands-on exercises with feedback, and a post-test covering the four skills. Then they conducted an exploratory mixed-method assessment of this curriculum with two groups of novice programmers to explore the theory's validity more broadly. They compared exercises' completion rate, error rates, ability to explain code, and involvement in learning the program. The study indicated that teaching skills improved the practical exercises' completion rate, decreased errors, and post-test comprehension.

**The works listed above have some limitations. Although these works research a method that helps students solve problems systematically, they do not provide tools to**

**foster cognitive skills in CS1 programming.** In addition to identifying and sequencing cognitive programming skills, our research intends to develop instruments that measure and foster these skills to improve programming teaching in CS1. Although Xie et al. has developed instruments to improve the instructional design in CS1. They still need to analyze the items' psychometric properties of the Practical Instrument and Assessment Instrument to determine the construct's validity [137]. However, they could have measured students' results with different prior knowledge, learning contexts, and motivations. That is, they did not measure the instrument's validity. Other cognitive skills may be involved in programming learning in addition to those listed in other works.

## 1.6 Contributions

Although related work has identified specific skills that novices learn, current theories of CS1 instruction have limitations in sequencing them. Thus, as this research's first contribution, we proposed adapting the cognitive domain of Bloom's Revised Taxonomy. This proposal identifies and sequences different introductory programming skills that novices can learn sequentially not to overwhelm students. This proposal emphasizes the adaptation of cognitive skills for teaching programming and the explicit rigor of how skills are defined and ordered in instruction. In such a way, this rigor allows the adaptation to help design instruments and learning materials in CS1.

The second contribution of this research is the instrument that we developed based on the proposed adaptation of the cognitive domain of Bloom's Revised Taxonomy. To provide empirical evidence for this proposal, we created an instrument and assessed its semantic and content validity. The instrument comprises 750 items that foster and sequentially measure each of the 19 skills for the introductory concepts in CS1. Then, through an empirical study, we obtained the instrument's internal consistency, the items' psychometric properties, and which skills were strongly related to the ability to write code.

The third contribution of this research is integrating adaptive selection into the instrument. This integration improved the items' administration and the participants' ability estimation. Computerized adaptive instruments do not need to select all items present in the bank. Consequently, the instrument application time is reduced and maintains the same level

of confidence as a conventional instrument, significantly reducing the fatigue of long tests. Another advantage is selecting items according to the subject's skill level, and this advantage thus avoids selecting an item that brings little information for the subject's evaluation.

The fourth and final contribution of this research is the instrument's assessment. We seek to understand the impact of explicit instruction and the practice of introductory programming skills in novices. We developed hypotheses based on our proposal that predicted that students who received practice in each of the 19 skills fostered by the instrument would be able to perform better in code writing. To evaluate this hypothesis, we experimented with novice programmers, where the experimental group learned with the instrument that reflected our proposal, offering practice for all skills. In contrast, the control group learned from material that provided practice only for the writing-related skill. We found evidence that those who received practice in all skills performed better in writing code, producing responses that reflected a greater depth of understanding.

The implications of this research extend to academia and industry. As a more diverse group of students begin CS1, there is a need to make instruction more effective. As a result, a greater need to address academia and industry about how people learn to code. However, current CS1 instructions fail, starting with "hello world" or another similar command. Later, students write code they cannot understand. These limitations include a need for more structure around knowledge related to different programming skills and how to sequence them. This gap between research and practice makes CS1 instruction difficult.

Specific learning taxonomies, such as the one proposed in this research, can help support instructional design in CS1. Previous approaches/theories could be more precise for developing instruments and learning materials. We can solve this gap with our proposal that incrementally fosters and measures the progression of cognitive skills in programming and avoids overloaded students.

## **1.7 Thesis' Outline**

In Figure 1.1, we present how we organize the remainder of this thesis.

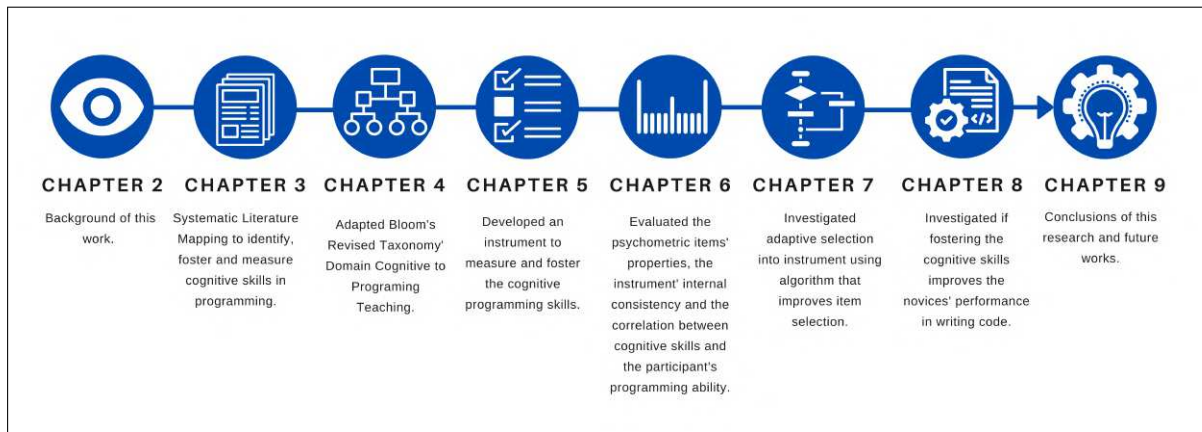


Figure 1.1: Thesis' Outline.

- In **Chapter 2**, we report the research background of this work, namely: competency model, taxonomies, measurement theory, and computerized adaptive testing;
- In **Chapter 3**, we describe the cognitive programming skills found in the literature and the approaches and instruments that measure and foster such skills;
- In **Chapter 4**, we present a proposal for adapting Bloom's Revised Taxonomy cognitive domain for the context of programming teaching. In addition, we present the cognitive skills' association found in the literature within the adapted taxonomy. Finally, we theoretically analyze this proposal through a survey applied to an expert group in CS1;
- In **Chapter 5**, we report an instrument created based on Bloom's Revised Taxonomy cognitive domain and adapted for programming teaching. Then, we analyzed the item's semantics and content present in this instrument;
- In **Chapter 6**, we analyzed the psychometric properties of the item present in the instrument and their internal consistency through measurement theories. We performed a correlation test on the cognitive programming skills (present in the instrument) and the participant's ability to program;
- In **Chapter 7**, we present the adaptive selection integration into an assessment instrument to improve the measurement of cognitive programming skills;

- In **Chapter 8**, we present an exploratory investigation of the instrument's validity for measuring introductory programming skills.
- Finally in **Chapter 9**, we present the conclusions and future work.



# Chapter 2

## Background

In this chapter, we report the research background covering the following topics for the understanding of this thesis: Competency Model (Section 2.1), Taxonomies (Section 2.2), Measurement Theory (Section 2.3), and Computerized Adaptive Testing (Section 2.4). Such theories, taxonomies, and models serve as theoretical support for the conclusion of this thesis.

### 2.1 Competency Model

ACM Computing Curricula 2020 (CC2020) presents a competence idea as a practical educational objective that enhances the Knowledge-Skill-Disposition (K-S-D) framework, which was popularized in the IT2017 report<sup>1</sup> [24]. Although different CS curricula have explored computing knowledge extensively, skill and disposition received significantly less focus [46, 126, 131].

The K-S-D dimensions compose the competence observed when a person performs a task [24]. A competency specification lists the knowledge, skills, and dispositions visible in performing a task with a purpose in a work context. Figure 2.1 illustrates the conceptual competence framework.

In the K-S-D framework, **knowledge** is the “know-what” dimension of competence, being a fact-based understanding. This dimension reflects the topics the educators list

---

<sup>1</sup>IT2017 report is the second edition of the ACM/IEEE-CS Curriculum Guidelines for Undergraduate Programs in Information Technology [105].

as courses in their syllabuses. Academic departments divide and balance content across courses and develop an academic program. Knowledge designates a core concept essential to competence. However, a concept is static and inert; it must be performed with the skill to become a behavior.

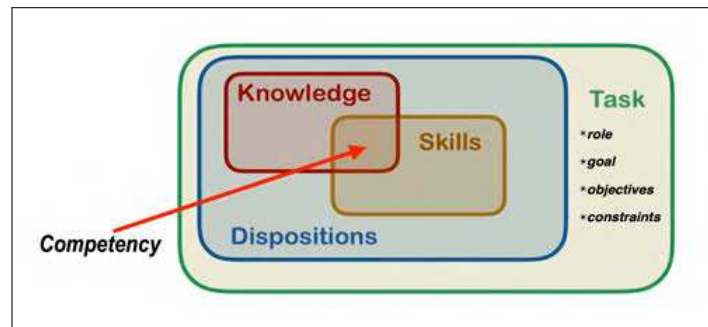


Figure 2.1: Conceptual Structure of the CC2020 Competency Model [24].

**Skills** introduce the ability to apply knowledge to accomplish a task actively. Therefore, a skill expresses a knowledge element as performed with proficiency to define the “know-how” dimension of competence. Skills need time and practice to develop. Therefore, improving skills generally requires engaging in a hierarchical sequence of higher-level cognitive processes. CC2020 adopted Bloom levels of a cognitive process to specify the skill required to perform any task successfully [24].

The evaluation process is primarily indirect and observes the competence dimension through the result. The activation of “know-what” by “know-how” combines knowledge and skills. As such, it only makes sense to use any knowledge item in a competency statement when applied at a specified or observed level of ability—such as levels in Bloom’s cognitive process. Therefore, each knowledge element and the required skill level work together with the competency specification.

**Dispositions** frame the “know-why” of competence and prescribe a character temperament in task performance. The provisions regulate the behavior of using the “know-what,” which becomes the “know-how.” Dispositions regulate knowledge and skill, linking their “optimal” or “proper” application to context.

Dispositions are habitual inclinations involving socio-emotional tendencies, predilections, and attitudes. Dispositions control whether and how an individual is inclined to use their abilities. The provisions can denote the values and motivations that

guide knowledge application, in which they designate the quality of knowledge that is an indicator of a standard of professional performance.

The **task** uses knowledge and provisions for practical application. The task provides the setting for people to demonstrate their desires in situations where they moderate their choices, actions, and efforts to achieve success efficiently and effectively. The task involves the purposeful context of the competence, showing the knowledge, skills, and integral dispositions. A task definition is a pragmatic rule that reflects professional practice relevant to the specific view of graduate programs. In this way, task explanations provide a context for the program to create a methodology that allows trainees to demonstrate capability as Information Technology professionals.

## 2.2 Taxonomies

We can use Learning Taxonomies in various contexts and with distinct purposes, such as to increase students' awareness, improve understanding and study skills [27], or improve critical thinking [94]. In addition, Learning Taxonomy is an essential tool for evaluating student performance. There are numerous taxonomies related to CS teaching: Bloom's Cognitive Domain, Unified Domain Taxonomy, and Structure of the Observed Learning Outcome (SOLO). In this work, we will apply Bloom's Revised Taxonomy as a reference, as argued by [70], that a professional programmer must be able to work at all cognitive levels.

### 2.2.1 Bloom's Revised Taxonomy

Bloom's Taxonomy is a conceptual framework designed to help define learning objectives. It is a model for educators to adopt in the practice and understanding of any content, helping them plan and evaluate. Although this taxonomy is suitable for Higher Education, few educators use it because they need to learn how to use it [43, 91].

However, Bloom's Taxonomy is a framework for classifying statements under which they help students learn through instruction [68]. The definition of educational goals guides the development of practices and activities that progressively foster students' cognitive learning. The educational goals are divided by taxonomy according to the cognitive, affective, and psychomotor specific domain development [80]. Table 2.1 presents a summary of each

domain's characteristics.

<b>Domain</b>	<b>Features</b>
Cognitive	Related to learning, mastering knowledge. It involves acquiring new knowledge, intellectual development, skills, and attitudes. Include knowledge of specific facts, standard procedures, and concepts that constantly stimulate intellectual development. The goals are grouped into six categories and presented in a hierarchy of complexity and dependency (categories), from the simplest to the most complex. The categories of this domain are <b>Knowledge; Understanding; Application; Analyze; Synthesis; and Evaluation.</b>
Affective	Related to feelings and postures. It involves categories linked to the development of the emotional and affective areas, which include behavior, attitude, responsibility, respect, emotion, and values. The categories of this domain are <b>Responsiveness; Reply; Appreciation; Organization; and Characterization.</b>
Psycho-motor	Related to specific physical abilities. Bloom and his team did not define a taxonomy for the psychomotor area. However, others did and arrived at six categories that include ideas related to reflexes, perception, physical abilities, improved movements, and non-verbal communication. The categories of this domain are <b>Imitation; Manipulation; Articulation; and Naturalization.</b>

Table 2.1: Bloom's Taxonomy's Three Domains' Features.

To advance to a new category, students need to perform adequately in the previous category, as each level uses the skills acquired in the previous one to improve. The cognitive, affective, and psychomotor domains are widely discussed and disseminated at different times by researchers. However, the cognitive domain is the most popular and widely used. Thus, we will address only this domain in this thesis. The cognitive domain involves knowledge and intellectual skills development. Many educators rely on this domain's theoretical assumptions to define their educational plans, objectives, strategies, and assessment systems.

Since its creation, the taxonomy has changed to constantly innovate and keep up with the teaching and learning process evolution. After 45 years, the need for updates to its conceptual structure arose. Krathwohl and colleagues revised Bloom's Taxonomy's original version for

the cognitive domain [68]. After the review, the taxonomy changed from a structure with only one dimension to one with two dimensions: Knowledge and Cognitive Processes.

Although the original taxonomy is still the most used, several authors criticized it [31, 71, 72]. They pointed out difficulties in its use, reporting that the categories are only sometimes easy to apply, as there is a significant overlap between them. The overlap causes debates on the order in which the categories analysis, synthesis, and evaluation appear in the hierarchy. Many works originated from the first release of Bloom's Taxonomy in the cognitive domain. However, with the new publications and technologies incorporated into the educational system, it was necessary to reassess and reread the theoretical assumptions that supported the original research to assess the need for adaptations.

Among the many revised versions, the most used is that of David Krathwohl [68], who even participated in developing the original taxonomy in 1956. He was the one who supervised the expert group (psychologists, educators, curriculum specialists, tests, and evaluation) and published the taxonomy review report in 2001. This group tried to balance what existed, the original taxonomy's structuring, and the new developments incorporated into education in the forty-odd years of its existence.

The original taxonomy changed from one-dimensional to two-dimensional, separating nouns and verbs, knowledge, and cognitive aspects. One of the two-dimensional structures was named the Knowledge Dimension, and the other was the Cognitive Process Dimension.

The knowledge dimension has four types:

- **Factual**, relates to the essential elements that students must know to become familiar with the topic to solve problems in it;
- **Conceptual**, consists of knowing the interrelationships between fundamental elements of a larger structure that allows them to work together;
- **Procedural** is the knowledge of how to do something, questioning methods; criteria for using skills, algorithms, techniques, and methods;
- **Metacognitive**, relates to cognition recognition in general and breadth awareness and knowledge depth acquired from a given content.

As for the cognitive learning goals, according to Krathwohl [68], it covers the six

categories of the original taxonomy; however, renamed to their verbal forms. The Knowledge category became **Remember**; understanding became **Understand**; synthesis became **Create** (and was promoted to the highest rank in the hierarchy); application, Analysis, and Evaluation became, respectively, **Apply**, **Analyze**, and **Evaluate**. Next, we present the different levels of Bloom's Taxonomy with their behavior. Remember is the lowest, and Create is the highest in the Bloom Cognitive Domain.

- **Remember** refers to learning the material presented; all that is needed is to recall the appropriate information;
- **Understand** allows demonstrating work understanding based on knowledge of it.
- **Apply** uses data, principles, and learned theory to answer a question in a new environment;
- **Analyze** breaks the material down into its constituent parts so that its organizational structure can be understood;
- **Evaluate** shows the ability to judge the material value for a given purpose based on definite criteria and fundamentals;
- **Create** recombines the parts created during analysis to create a new entity different from the original.

Bloom's Taxonomy helps the educator decide which action or behavior the student should perform for a particular activity. Furthermore, Bloom's Taxonomy helps reach the proposed objective and organize content coherently in face-to-face or distance learning. Thus, even keeping the original structure part, the revised taxonomy is better suited to support new learning methods and take advantage of better educational goals [127].

There are verbs associated with the taxonomy levels. These verbs help to classify an assessment question into the taxonomy levels. Table 2.2 shows the verb applied in the programmer's assessment of code comprehension tasks [63].

Level	Verbs
Remember	collect, copy, define, describe, enumerate, examine, identify, label, list, name, quote, read, recall, retell, record, repeat, reproduce, select, state, tell
Understand	associate, cite, compare, contrast, convert, differentiate, discuss, distinguish, elaborate, estimate, explain, extend, generalize, give, group, illustrate, interact, interpret, observe, order, paraphrase, review, restate, rewrite, subtract, trace
Apply	administer, apply, calculate, capture, change, classify, complete, compute, construct, demonstrate, derive, determine, discover, draw, establish, experiment, illustrate, investigate, manipulate, modify, operate, practice, prepare, process, produce, protect, relate, report, show, simulate, solve, use
Analyze	analyze, arrange, breakdown, classify, compare, connect, contrast, correlate, detect, diagram, discriminate, distinguish, divide, explain, identify, illustrate, infer, layout, outline, point, out, assess, select, separate, subdivide
Evaluate	appraise, assess, conclude, criticize, convince, decide, defend, discriminate, evaluate, explain, grade, judge, justify, measure, rank, recommend, reframe, support, test, validate, verify
Create	adapt, combine, compile, compose, construct, correspond, create, depict, design, devise, express, format, formulate, facilitate, improve, integrate, invent, plan, propose, rearrange, reconstruct, refer, relate, reorganize, revise, specify, speculate, substitute

Table 2.2: Verbs Used in Code Comprehension Tasks.

Educators can work on some activities at each level present in Bloom's Taxonomy:

- For example, the Remember level consists of remembering information, knowing the requirements to write a program, or understanding the language's syntax. The Educators can ask the learners to learn about the selection structure; at this level, the learner should know three selection structure options;
- The second level is Understand and deals with remembering facts and interpreting them. Learners understand how each selection design works and their behavior at this stage;

- The third level is Apply. This competence level consists of applying and using the knowledge they have learned. The Educators can ask the learners to apply the correct design to a given problem; the learner can modify the program to add a selection or modify the existing selection structure in the program;
- The fourth level is Analyze. This level of competence is about seeing patterns and using them to analyze a problem. The Educators can ask the learners to solve the problem by applying nested selection or debugging the program selection design;
- The fifth level is Evaluate. This competence level concerns the learner's ability to judge the validity or idea quality. The Educators can ask the learners to justify using the switch statement instead of the if statement.
- The sixth level is Create. This competence level concerns the student's capability to extract knowledge from multiple subjects and synthesize this information before concluding. At this level, the learner should be able to apply selection structures in appropriate situations.

### 2.2.2 SOLO Taxonomy

This taxonomy focuses on the students' characteristics and observable results rather than the cognitive activity required to produce those results. Thus, the SOLO Taxonomy can be an excellent strategy for programming assessments to verify novice programmers' learning effects. CS research recently studied the SOLO taxonomy about reading and writing code [49, 61, 89, 102, 117, 135, 137].

The educator can evaluate the student's code writing according to the following SOLO taxonomy hierarchy [18]:

- **Prestructural [P]** substantially lacks programming constructs' knowledge or is unrelated to the question;
- **Unistructural [U]** represents a direct specification translation. The code will be in the specifications sequence;



- **Multi-structural [M]** represents a translation close to a direct translation. The student may have reordered the code to make a valid solution;
- **Relational [R]** provides an excellent well-structured program that removes all redundancy and has a clear, logical structure. The students integrated the specifications to form a logical whole;
- **Extended Abstract [A]** used constructs and concepts beyond those required in the exercise to provide an improved solution.

The SOLO Taxonomy is suitable for measuring different types of learning outcomes. However, it has some limitations. One of them is that the more refined categorization of SOLO levels still needs to eradicate the problem of its conceptual ambiguity [22]. Educators may find it difficult to sort student responses into a category [76]. Classifying students' responses is also a manual task and requires the vision of many specialists involved in the process, thus making large-scale applications difficult [61].

## 2.3 Measurement Theory

We aim to collect data and assess students' cognitive skills related to our scientific topics in Education and CS research. Therefore, we use the Measurement Theory, also known as Psychometric Theory [104], which provides the foundation for evaluating educational tests and their uses and interpretations. Numbers make it possible to quantify natural phenomena in scientific studies, and measurement instruments and techniques carry out the quantification that favors understanding these phenomena when linked to scientific methods.

Measurement Theory has two main statistical approaches to describe an individual's characteristics and analyze one's abilities, namely: Classical Test Theory (CTT) and Item Response Theory (IRT) [39, 101]. This section provides information about the essential properties of theories. It determines the psychometric measurement process with details and compares theories' models. The earliest measurement theory, CTT, and this theory-enhanced application, IRT models, are examined from familiar and different viewpoints. This section emphasizes the importance of constructing, measuring, evaluating, and correctly interpreting the educational measurement process.

Validity and reliability are the most fundamental concepts of Measurement Theory. These concepts provide the basis for judging the technical quality and use appropriateness and educational interpretations of test results. There is a broad consensus among measurement experts that while reliability is essential, validity is crucial in evaluating the uses and inferences of test results. In addition, test results comparability for different participants or from one occasion to another is also essential [74].

### 2.3.1 Classical Test Theory

CTT considers the total score of an instrument as a measure to assess a person's performance. The total score consists of the person's actual sum of scores added to the possible measurement errors made during the instrument application. The correct score would measure skill in the perfect situation, in which the person will respond to the instrument based only on their knowledge. That is, without the possibility of random hits and any external or internal distractor [8, 12].

However, every measurement includes errors during the process, and we need to know a person's actual score as there is no strategy to measure it. Measurement errors happen when we cannot control all factors influencing the measurement process. In addition, system factors can cause measurement errors [101]. In practice, we minimize errors and produce reliable items. Therefore, in the CTT, we can consider the answers' correct number on the instrument as the total score.

CCT uses norms to interpret an instrument's score. These norms constitute a reference for us to interpret and classify the scores, for example, situating the subject position in the construct measured by the instrument or making comparisons [101, 103]. It is possible to construct these norms from a large enough population sample. In addition, sampling techniques, such as the stratified one, serve as a reference for the construction standards. Thus, the sample is a reference to compare and classify the subjects who will respond to the instrument [13].

We can use some measures to assess the items' quality and the instrument through the score, such as the Biserial Point Coefficient and Cronbach's Alpha Coefficient [14]. In the following subsection, we present the main application concepts.

### Biserial Point Coefficient

It is possible to calculate the correlation coefficient between two variables in traditional tests, one numerical and the other nominal categorical. In this case, the categorical variable has only two possible values: right and wrong; this variable is called dichotomous. Then, to calculate the correlation between this categorical variable and a numeric variable, we can proceed with Pearson's coefficient, given the sample's normality. This strategy is called Biserial Point Coefficient [12].

Biserial Point Coefficient consists of a Pearson's correlation between dichotomous variables and the instrument's score. We used the Biserial Point Coefficient to discriminate the item instrument's result. It indicates an item's ability to differentiate people with weak and strong abilities in the tested task [8]. Equation 2.1 defines the Biserial Point Coefficient:

$$Ppb = \frac{\bar{X}_A - \bar{X}_T}{S_T} \sqrt{\frac{p}{1-p}} \quad (2.1)$$

Where,

$\bar{X}_A$  represents the global average of the respondent's scores who reached the item right;

$\bar{X}_T$  represents the global average of the instrument's scores;

$S_T$  represents the instrument's standard deviation;

$p$  represents the respondents' proportion who reached the item right.

Biserial Point Coefficient estimates the items that most impact the estimated skill. If an evaluated individual gets this item right, he will have a higher chance of passing the exam. The biserial Point Coefficient varies between -1 and 1. The closer to 1, the more discriminating the item, the higher the coefficient value, and the stronger the item's correlation with the total score. This value will show that the item is essential for the instrument. For example, if item 4 has a coefficient of 0.84 and item 5 has a coefficient of 0.43, then item 4 is more discriminative than item 5. Individuals who get item 4 right are likely to get better instrument results.

As an example, to estimate the item's biserial correlation with the instrument's score, the following measures must be taken: the use of the *ltm* package with *biserial.color()* function, passing as parameters the score *rowSums(data.items)*, an item *data.items[[1]]* and informing that the question is dichotomous *level=2*, according to an example of R code. In the end,

the item's biserial correlation with the score was 0.85, indicating that this item strongly correlates with the test result by the CTT.

```
> biserial.cor(rowSums(data.items), data.items[[1]],level=2)
[1] 0.8596172
```

### Cronbach's Alpha Coefficient

Checking reliability while building an instrument is the most time-consuming and cost-efficient approach. Internal consistency measures the instrument's reliability. The Internal consistency examines the items' homogeneity that composes the instrument, verifying the relationship magnitude between the items and the total score. Calculating Internal Consistency from the instrument's overall score and each item's score is possible. Equation 2.2 defines Cronbach's Alpha Coefficient:

$$\alpha = \frac{n}{n-1} \left( 1 - \frac{\sum S_i^2}{S_T^2} \right) \quad (2.2)$$

Where,

$n$  represents the items' number;

$\sum S_i^2$  represents the variances sum for  $n$  items;

$S_T^2$  represents the global range of test scores.

Internal consistency using Cronbach's Alpha Coefficient ranges from 0 to 1. Values closer to 1 indicate that the instrument has appropriate internal consistency [6]. Values between 0.70 and 0.80 are considered acceptable but with caveats. When the values are below 0.70, the instrument's items need to be reassessed by the researcher [8].

It is possible to calculate the Instrument's internal consistency using *ltm* package with *cronbach.alpha()* function. The following code snippet shows the function *cronbach.alpha()* passing as a parameter the items *data.items*. We depict a syntax example with the output generated by the function. The result indicates that the alpha value was 0.919. This value is considered acceptable for the Instrument's Internal Consistency.

```
> cronbach.alpha(data.items)
Cronbach's alpha for the data.items data-set
Items: 20
Sample units: 100
alpha: 0.919
```

### 2.3.2 Item Response Theory

IRT is a statistical theory used by psychometrics and the educational area for constructing, evaluating, and validating instruments [8]. The mathematical models depend on the adopted logistical model and the instrument dimension. In this section, we present the three-parameter one-dimensional logistic model. We emphasize, through computational resources, as packages in R, that we can perform all the IRT's statistical analyses.

We may use IRT to design educational assessment tests, item calibration (characterizing items by numerical parameter values), and other test development processes. We can also fit the data into a model. This way, different subjects or the same subject can compare their abilities at different times, using parameters that are measured statistically, regardless of the sample used [96].

IRT considers the subject's response to the set of items to provide estimates for the assessed skill. The ability estimate, called theta ( $\theta$ ), is related to the subject's probability of responding correctly to the items, considering one or more parameters [13]. For this reason, IRT is also known as Latent Trace Theory. This theory treats constructs as composed of dimensions with different magnitudes properties that can be measured [101].

A set of hypothetical factors or variables can predict the individual's behavior toward an item. Furthermore, the dependency between behavior and skill may be related to a growing monotonous mathematical function. This function sketches a graph called the Item Characteristic Curve (ICC) [34] [101].

#### Item Characteristic Curve

Theta skill is related to the item hitting probability through mathematical functions. These mathematics functions need to be estimated and are called ICC, which provide information

about each subject's probability of getting the item right [101]. ICC is a graph representing the relationship between the estimated ability and performance on items [13]. Three properties describe the ICC:

- **Slope:** item or “parameter a” is estimated together with the other parameters using maximum likelihood estimation or some point characteristic of the parameters' posterior distribution. The steeper the curve, the greater the item discrimination. This parameter describes how many individuals with different abilities differ in the item correct probability, that is, the power to specify subjects with similar magnitudes in the latent trait to which it refers, ranging from 0 (not at all discriminative) to 4 (extremely discriminative);
- **Threshold:** item or “parameter b” refers to the skill needed by an individual, calculated from the probability of getting the item right. This parameter ranges from -4 (easy items) to 4 (difficult items), passing through the value 0 (average items). An item is easy when the model estimates the difficulty value at lower skill levels, and the ICC is more to the left on the graph. As for more complex items, the difficulty value has higher skill levels, and the ICC is positioned more to the right on the graph.
- **Asymptote:** item or “parameter c” refers to the subject probability with a low ability to give a correct answer to a problematic item: the chance of an individual hitting the item with a guess, ranging between 0 and 0.5. Values above 40% of guess probability are critical, and the researcher should review the item. The model represents this parameter at the curve's origin axis, and its extension is proportional to this point's deviation value of 0.

Distinct mathematical models can be used depending on the number involved in the parameters, dimensionality, or items present in the instrument. Equation 2.3 describes the three-parameter one-dimensional logistic model. The equation determines the subject probability with Theta skill getting item j right depending on the slope (a), threshold (b), and asymptote (c).

$$P(\theta) = c_j + (1 - c_j) \frac{1}{1 + e^{-a_j(\theta - b_1)}} \quad (2.3)$$

Where,

$\theta$  represents an individual's latent trait/skill;

$a_j$  represents the item's slope parameter;

$b_j$  represents the item's threshold parameter;

$c_j$  represents the item's asymptote parameter.

Equation 2.3 is a three-parameter one-dimensional logistic model (3PL) because it uses the cumulative distribution function of the logistic distribution. IRT uses other distributions in the standard function, such as the normal or continuous cumulative distribution functions. The model assesses the slope and threshold parameters in the 2-parameter logistic model (2PL). In this case, in Equation 2.3, the value asymptote is considered zero [13, 14]. Finally, the 1-parameter logistic model (1PL) only assesses item threshold, also known as the Rasch model. In the latter case, the equivalent threshold value starts at 1. Choosing one of these models depends on the data collected from the real world to the model [101].

We can use the *ltm* package and *tpm()* function to estimate the parameters' values of the items of the instrument. The following code snippet depicts the calling of *tpm()* function, passing the responses of the given items *data.items* and *control* through a control value list with optimized elements by a string '*nlminb*'. Next, we show the function's output for the first five items.

```
> data.tpm <- tpm(data = data.items, control = list(optimizer = "nlminb"))
```

*Coefficients:*

	<i>Gussng</i>	<i>Dffclt</i>	<i>Dscrmn</i>
<i>i1</i>	0.244	1.982	0.635
<i>i2</i>	0.191	0.254	1.896
<i>i3</i>	0.156	1.114	1.718
<i>i4</i>	0.140	0.246	1.078
<i>i5</i>	0.022	-2.765	0.935

The first column corresponds to the instrument items listed in the order of the source file in the function's output. In this case, we only print the first five items. *Dscrmn* column corresponds to **slope** parameter. *Dffclt* column shows the **threshold** parameter. Finally,

the *Gussng* column corresponds to the **asymptote** parameter. Analyzing the output, we can observe that item i1 is the most difficult and with the most chances of getting it right by chance. Item i5 is easier, less discriminating, and less likely to be guessed. Item i2 is discriminating.

The graph in Figure 2.2 shows five ICC's, each corresponding to an item, modeled from Equation 2.3, which we will use as examples to illustrate the parameter's behavior. The skill value (Theta  $\theta$ ) can assume any actual number [12]. Generally, the x-axis representing the skill scale (Theta  $\theta$ ) ranges from -4 to 4. The y-axis represents the item's probability of a correct answer ranging from 0 to 1. The lines indicate difficulty level when the correct answer's probability is 50%, as the threshold parameter and skill (Theta  $\theta$ ) are on the same scale. In the example in Figure 2.2, item "3c" is the most discriminating, item "3b" is the most difficult, and item "1" is the easiest and least discriminatory.

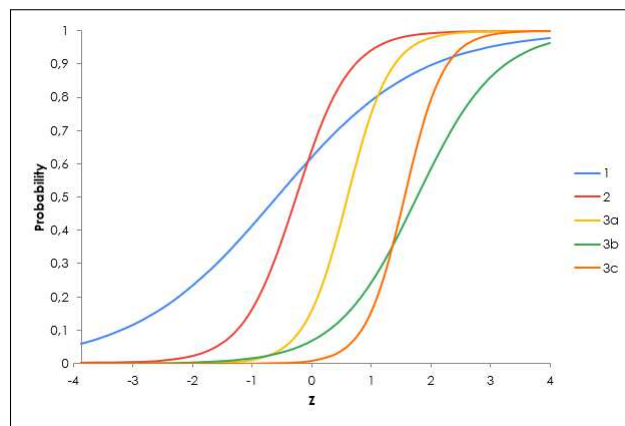


Figure 2.2: ICC's Example.

We can elaborate the instrument's ICC's through the *plot()* function of the *ltm* package. The following code shows the *plot()* function call, passing the item parameters.

```
> plot(data.tpm, legend=T)
```

We can obtain item parameters through the multiplicity of specialized computer programs. These programs use nonlinear mathematical functions like the logarithmic functions that produce ICC. However, the graphical representations of mathematical functions that relate the item's response probability to the latent trait's level or skill [8].



### Latent Variable Estimate

Regardless of the model choice, the IRT generates a standardized scale (mean=0 and standard deviation=1) called trait or latent ability that can vary, according to threshold parameters, between -4 and 4. Thus, the IRT estimates the scores using an estimation method [45].

**Maximum likelihood** procedure estimates the examinee's ability. It is an iterative procedure, as in estimating the item's parameter. It starts with some *a priori* value for the examinee's skill and the known values of the item's parameters. The procedure used this data to calculate each item's correct answer probability. Then, the procedure obtains an adjustment for the skill's estimate that improves according to the probabilities calculated from the test item response. The procedure repeats this process until the adjustment becomes small enough that the change in estimated ability is negligible, resulting in an examinee's ability estimate. This process is repeated separately for each examinee [13]. Equation 2.4 defines the latent trait estimation.

$$\theta_{s+1} = \theta_s + \frac{\sum_{i=1}^n a_i [u_i - P_i \theta_s]}{\sum_{i=1}^n a_i^2 P_i(\theta_s) Q_i(\theta_s)} \quad (2.4)$$

Where,

$\theta_s$  represents the examinee's estimated ability within s iterations;

$a_i$  represents the slope parameter of item i,  $i = 1, 2, \dots, N$ ;

$u_i$  is the answer made by the examinee to the item (1 when right and 0 when wrong);

$P_i(\theta_s)$  is the correct answer probability to item i, under the ICC model at skill level  $\theta$ ;

$Q_i(\theta_s)$  is the incorrect answer probability to item i under the ICC model at skill level  $\theta$ .

Unfortunately, there is no way to know the examinee's actual skill. In this case, the best procedure is to estimate it. However, one should obtain a standard error of the estimated skill that indicates accuracy. Applicants can administer the instrument and estimate the examinee's  $\theta$  capacity as often as necessary. With each application, the examinee will have different estimates for his skill. **Standard error** measures the  $\theta$  values variability around the subject's value for the unknown ability. This error is estimated using Equation 2.5.

$$SE(\theta) = \frac{1}{\sqrt{\sum_{i=1}^n a_i^2 P_i(\theta_s) Q_i(\theta_s)}} \quad (2.5)$$

The term under the square root sign is precisely the denominator of Equation 2.4. As a result, the estimated standard error results from the by-product of the examinee's skill estimate.

Another way to estimate an examinee's ability is through the **Expected A Posteriori** (EAP). EAP derives from Bayesian statistical principles. The term "a posteriori" derives from the Bayesian concept of posterior probability. This context refers to a posterior probability distribution of latent trait scores. Specifically, the scores' predicted distribution for a given case, given (a) that case's response pattern and (b) the model's estimated parameters [101]. The term "expected" derives from the concept of an expected value. Thus, an EAP estimate refers to the expected value of the posterior probability distribution of the latent trait scores for a given case. Equation 2.6 defines the latent trait estimation.

$$\theta_{s+1} = \theta_s + \frac{\sum_{i=1}^n a_i [u_i - P_i \theta_s]}{\sum_{i=1}^n a_i^2 P_i(\theta_s) Q_i(\theta_s)} \quad (2.6)$$

Where,

$\theta_s$  represents the examinee's estimated ability within s iterations;

$a_i$  represents the slope parameter of item i,  $i = 1, 2, \dots, N$ ;

Through the *irtoys* package and *eap()* function, we can estimate the abilities of those examined by a posteriori expectation. This estimator depends on the prior average of all those examined, guaranteeing the exact estimate. The following code shows *eap()* function call, passing: the items' responses *data.items*, the items' parameters already estimated previously *data.items.par\$est* and the choice of a quadrature object produced in normal *qu=normal.qu()*. We show the extract function's output of the first six examined outputs using the *head(hab)* function:

The result above shows the values for the first six students. The *est* column represents the skill estimate for each student. The *sem* column shows the estimate's standard error. Finally, column *n* shows the number of items in the instrument, in this case, 18 items.

```

>hab <- eap(data.itens, data.itens.par, qu=normal.qu())
>head(hab)
           est           sem           n
[1,] 0.61104042    0.4771454         18
[2,] 1.05883705    0.3089617         18
[3,] 1.92623806    0.4680334         18
[4,] -1.71143568    0.6046471         18
[5,] 0.47840008    0.3540985         18
[6,] -0.79040561    0.4448174         18

```

### Item Information Function

In IRT, the goal is to estimate the examinee's skill. During this process, the model calculates the standard deviation of the examinee's estimate. If this term is squared, it becomes a variance, which measures how accurately the model estimates a given skill level. The Item Information Function (IIF) at a given skill level is reciprocal of this variance [101].

If the amount of information is significant, the model can estimate an examinee whose true capability is at that level more accurately. All estimates will be reasonably close to the true value. If the amount of information is small, the model cannot accurately estimate the capability, and the estimates will spread widely over the true capability. We can use Equation 2.7 to calculate IIF.

$$I(\theta) = a^2 \frac{Q_i(\theta)(P_i(\theta) - c)^2}{P_i(\theta)(1 - c)^2} \quad (2.7)$$

Where,

$a$  represents the item's slope parameter;

$c$  represents the item's asymptote parameter;

$P_i(\theta)$  is the correct answer probability to item  $i$ , under the ICC model at skill level  $\theta$ ;

$Q_i(\theta)$  is the incorrect answer probability to item  $i$  under the ICC model at skill level  $\theta$ .

The IRT gets the IFF for each item in a test. The amount of information for each item is relatively small. Typically, an examinee's skill with a single item is not estimated. The

information amount at a capability level, and the test information function is of primary interest. The IRT obtains test information by summing the item information at a specific capability level. The mathematical definition of the amount of item information depends exclusively on the ICC model used by the team that develops the instrument/test. Therefore, examining these definitions in each model is necessary [13].

The IIF is a powerful tool for item analysis, allowing us to know how much information an item accumulates at a given  $\theta$  value and at what  $\theta$  value of the most crucial information of the item. IIF has been the most used item analysis' method by test builders today [12].

The graph in Figure 2.3 shows five IIFs, each corresponding to an item, modeled from Equation 2.7, which we will use as examples to illustrate the item's behavior. The skill value ( $\theta$ ) can assume any actual number [12]. Generally, the x-axis representing the skill scale ( $\theta$ ) varies between -4 and 4. The y-axis represents the amount of information for this item in the skill. The lines indicate each item's information in each latent trait region. In the example in Figure 2.3, item "3a" gives less information when the skill is at -2. However, this item provides more information on the average skill (when theta is close to 0).

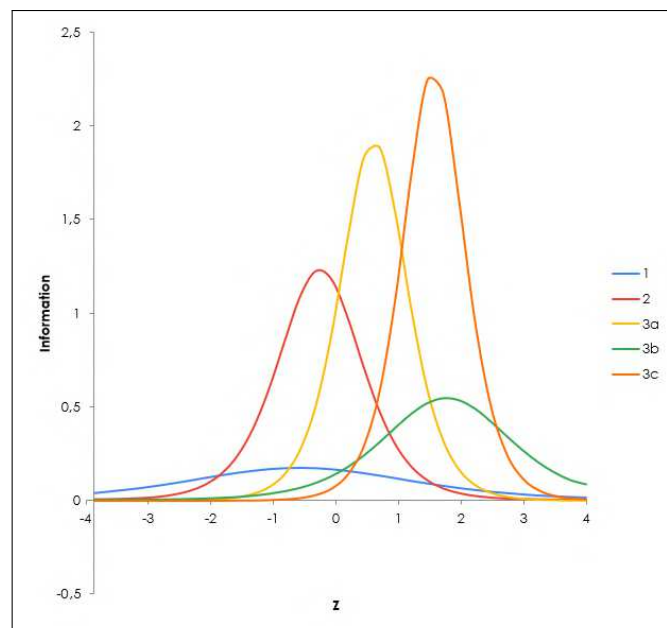


Figure 2.3: IIF's Example.

We can use the function `plot()` to plot graphs of the instrument's IFFs. The following code shows the calling of the `plot()` function, passing the item's parameters (`data.tpm`) and the ICC function' type.

```
> plot(data.tpm, type = 'IIC', legend=T)
```

Most of the instruments developed at IRT are on paper and pencil, and the subjects answered the same items in these instruments in the same order. Due to the nature of the administration of these instruments, educators can estimate the student's ability imprecisely. To overcome this limitation and improve the reliability of the assessment instruments, the IRT has a great ally: Computerized Adaptive Testing (CAT).

## 2.4 Computerized Adaptive Testing

CAT seeks to adapt the instrument's items to the skill level of each examined subject. CAT chooses from the items the same way an expert would intuitively [25]. As a result, these tests offer several advantages over traditional tests, such as:

- Reduction by one test size. The number of items in the adaptive tests is lower than in a traditional test for the same level of accuracy. This reduction is only possible due to the information reduction about the skill estimate in each administered item;
- Flexibility in test batteries. As with traditional tests, the participation of all subjects at the same time in the test application is unnecessary;
- Greater control of test rules;
- Reduces errors that can occur in optical correction processes;
- Computerized tests better motivate individuals because they use multimedia features that make them more attractive than traditional tests.

Given the limitations, these tests require more financial and human resources when compared to traditional paper-and-pencil versions. Such resources aim to build the items' database and information security that require investments in hardware and software for their creation and application.

The instruments' construction is a multi-step process, requiring multidisciplinary team support to guide all steps. Experts in skill analysis, statistics, and psychometrics compose

the team to build the instrument [101]. When the instrument is a CAT, we also need IT professionals. This section presents the steps involved in producing an instrument based on the IRT. In Figure 2.4, we detail the steps in developing a computerized instrument.

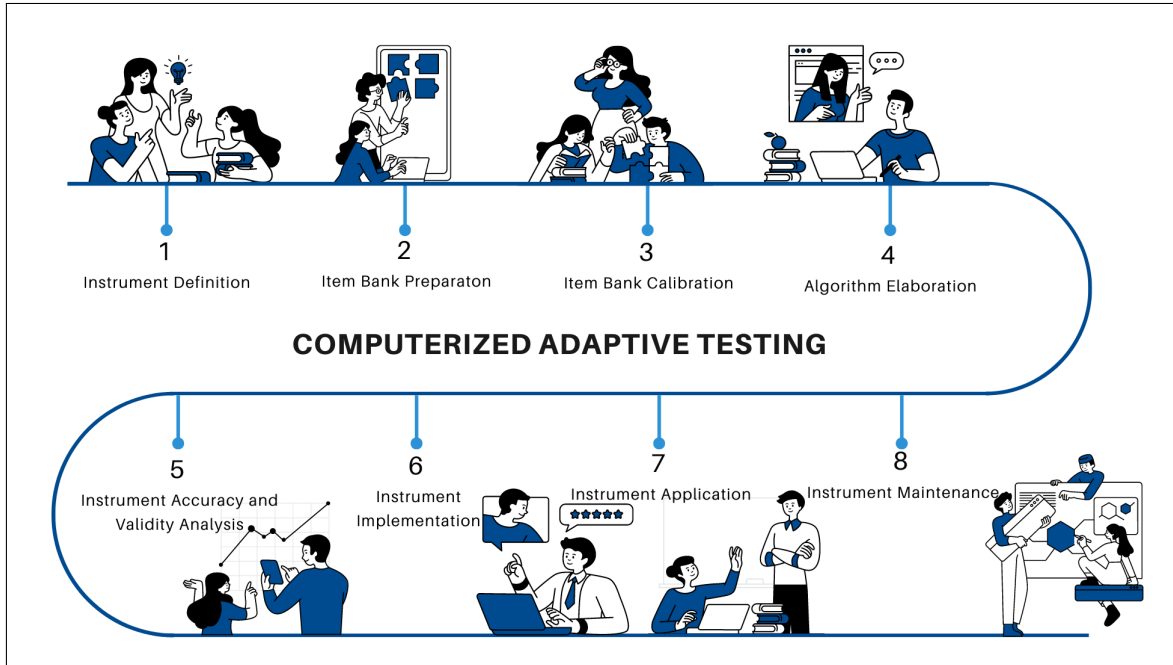


Figure 2.4: Computerized Adaptive Testing steps.

### 2.4.1 Step 1—Instrument Definition

At this step, verifying the instrument's previous existence and defining its objective, skill, and dimension is necessary. The instrument's goal indicates its general purpose: to estimate a grade or classify individuals. Skill is the characteristic to be evaluated: which the instrument aims to measure. The instrument's dimension indicates the number of skills assessed. Figure 2.5 presents the Instrument Definition Step details.

The first step is to verify the instrument's previous existence. If the instrument does not exist, the team must build it, define its objective, skills, and dimension, then go to Step 2. Suppose the instrument the team wants to build already exists, even in a non-adaptive default version. In this case, the team should check whether the instrument has psychometric validity. Otherwise, the team must disregard it and create a new instrument. If the instrument is valid, its adaptation to a CAT based on existing historical data is verified (item bank and examinees' answers) [101].

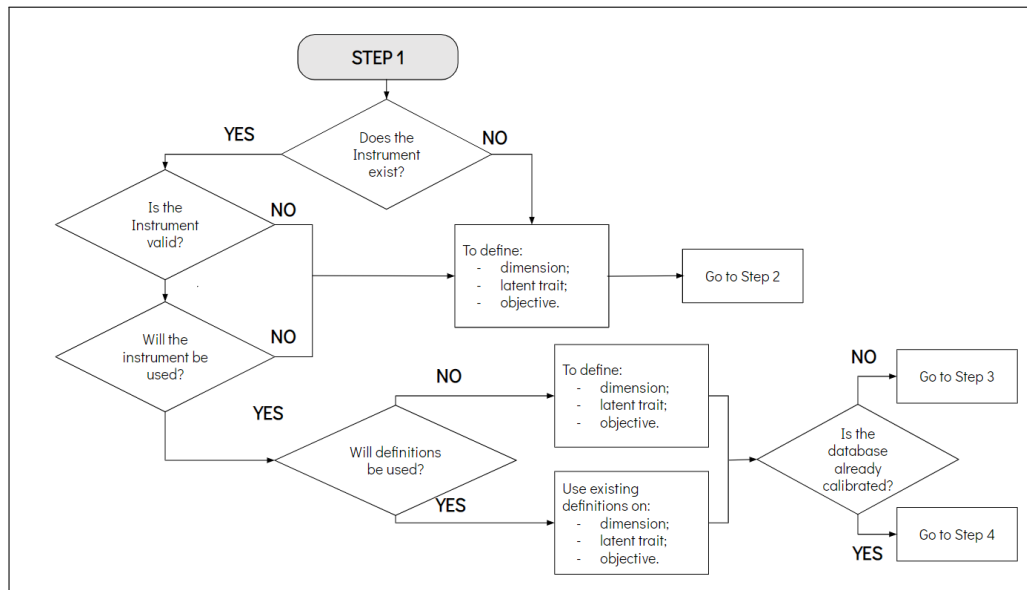


Figure 2.5: Instrument Definition Step.

If the team uses an existing instrument, it is necessary to verify the objective, skills, and dimensions. Otherwise, the team should define these aspects as discussed above before proceeding. Once the team decides to use or define all three aspects, the next step is to verify the item bank calibration and observe whether the instrument uses IRT in its standard version [12]. If the instrument already uses IRT, item calibration will not be necessary, and the next step will be the construction of the CAT algorithm; go to Step 4. If the instrument did not use IRT, the scoring uses the CTT; the team must calibrate items and go to Step 3.

### 2.4.2 Step 2—Item Bank Preparation

When moving to the design stage of the items, the team must master the estimated skill theory through the item. Furthermore, the team must design all items so the examinee can express skill by answering them. In multiple-choice items, an item consists of a description, its alternatives, and the correct alternative. This step is the most laborious because developing items that assess only one skill is a self-reflection and a practical knowledge task [8]. Figure 2.6 presents the Item Bank Preparation Step details.

The item bank development also involves defining the item's type and quantity to compose the instrument. The item's definition type depends on the instrument's objective [13]. For example, when we aim to measure proficiency, we should use a

multiple-choice answer format. If the instrument is digital, the item comprises multimedia resources such as images, animations, and sounds.

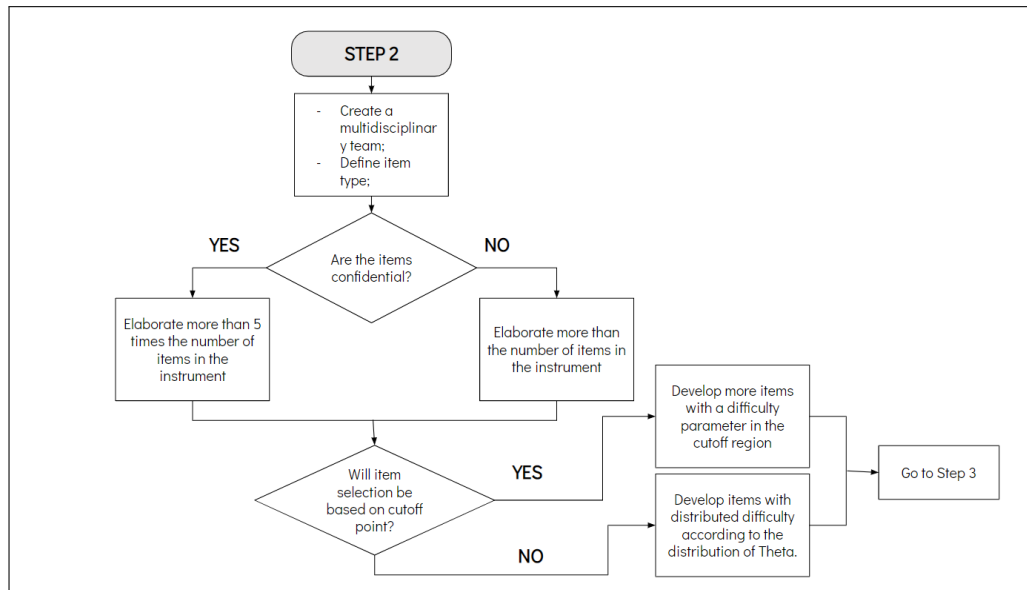


Figure 2.6: Item Bank Preparation Step.

In the item development stage, the team must consider the number of items managed by the instrument, as this information depends exclusively on the item's type. For example, suppose the items belong to the proficiency instrument. In that case, it is necessary to control their exposure, so the team must design many items. The number of items must be at least five times the number of items the instrument will have. Due to the calibration process, some items from the instrument are likely to be dropped [103]. The greater the number of items in the bank, the better the instrument, as there will be more suited items to a certain skill level.

Suppose CAT selects items based on a cutoff point. In that case, the team should design more items close to the cutoff region with the difficulty parameter. Otherwise, the team must create items with difficulties distributed according to  $\Theta$  and go to step 3.

After the item development process, the team should evaluate them. Judges can carry out theoretical analyses. That is, specialists in the researched area carry out theoretical analyses. The judges check if the items are well understood (semantic analysis) and if the items adequately measure the desired skill (content analysis) [101].

A semantic analysis checks whether the items are intelligible to all subjects, and the items need to be easy to understand by everyone, even subjects with lower skill traits. The semantic



analysis ensures that the difficulty in understanding the items should be a manageable factor that could interfere with the subject's response [7].

Content analysis ensures that the items refer to the skill we aim to estimate. The number of judges may vary, but the literature recommends that there should be at least three of them [58]. An agreement of 80% among judges can be a reference to decide if the item refers to skill and to include it in the instrument [8, 101]. If there is less than 80% agreement, we must exclude the item from the instrument. In the case of having three judges, all three must agree on including the item in the instrument. If there are only two judges, the agreement level is 66.6%, not reaching the required 80%. However, the judge's qualification is more important than the agreement quantity in the research-specific area [8].

### 2.4.3 Step 3—Item Bank Calibration

Item Bank Calibration consists of applying the items, collecting data, choosing the response model, choosing the calibration method, and designing and interpreting the scale. The items have already gone through the judges' semantic and content analysis and will go to an item bank. Figure 2.7 shows the Item Bank Calibration Step details.

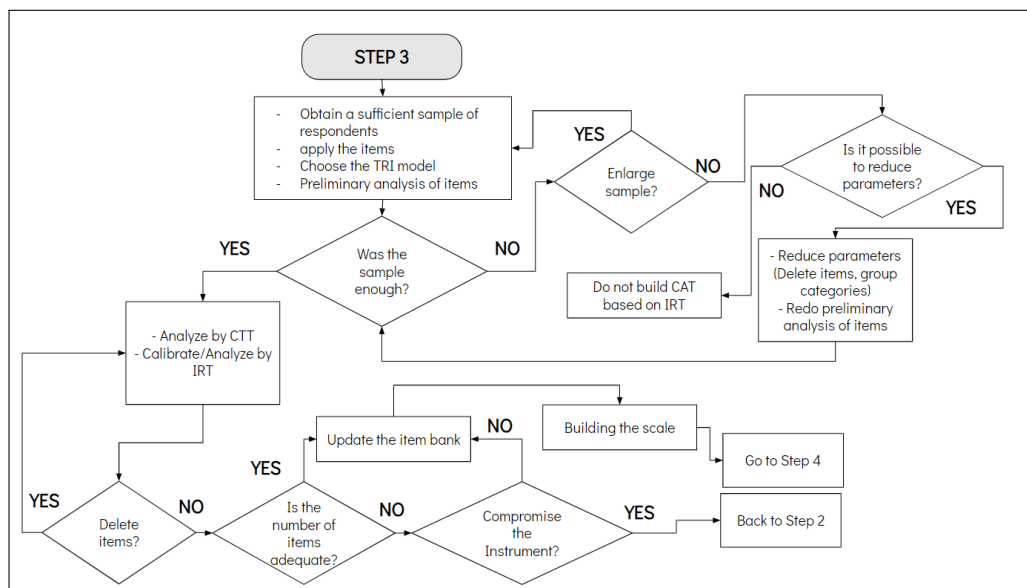


Figure 2.7: Item Bank Calibration Step.

When applying the items, the team needs a sufficient respondent sample. The sample size depends on the number of items in the bank, so the more items, the larger the sample.

For example, in a 20-item database, the sample must consist of at least 200 respondents (ten subjects for each item) [14]. The team can apply the items in the traditional paper-and-pencil format or a computerized version to obtain the sample. All subjects must answer all items in this step.

If the sample is sufficient, the team should analyze the items using the CTT criteria and calibrate and analyze them using the IRT. These analyzes will help eliminate inappropriate items. After the Item Bank Calibration, the team must verify whether the item's quantity is adequate and whether it covers the latent trace. If the sample is sufficient, the team can adopt some modifications to reduce the number of parameters. For example, the team can eliminate underperforming items or reduce the number of item categories [101]. After these modifications, the team must redo the preliminary analysis and continue the analysis depending on the results' consistency, in case the sample is sufficient. Suppose that the result needs to be correctly calibrated by the team after all the efforts needed to take this sample. In this case, the team cannot use IRT to calibrate the items and build the instrument.

The chosen response model consists in verifying which IRT model fits the instrument. The team must verify that the fit was adequate and, if necessary, replace the adjusted model. A poorly tuned model will not provide constant parameters for items and abilities. Suppose the estimation of the item parameters through the IRT is inconsistent, for example, with absurd values or high standard error. In that case, this may be due to inadequate sample size [14].

Typically, the item bank is too large for all subjects to respond to all items. One way to solve this is to use balanced incomplete block techniques and equalization methods [15]. Suppose there is a divergence in the instrument's dimensionality constructions. In this case, the team can choose a response model that considers this dimensionality and exclude items that contribute to undesirable constructs or produce more items that consider the desired constructs. However, the team can collect more samples from these new items' calibration.

The Item Bank Calibration method uses the CTT and IRT criteria to estimate item parameters. Analysis by CTT indexes helps to eliminate inappropriate items. The team carries out the item calibration process through IRT and parameter analysis, and this analysis will help in the decision to exclude inappropriate items. The team estimates the parameters by the maximum likelihood using the ltm R package. Alternatively, it uses a Bayesian

approach using the *bairt* R package. The team must verify whether the deletion will not affect the other items. Then the team must rerun the calibration process to verify whether the remaining items are adequate and not affected by deleting the other items.

The team must analyze the parameters to know if an item is suitable. Estimates with critical slope, threshold, and asymptote parameters imply removing the item from the bank. A slope index below 0.30 is inadequate for an item to have the power to differentiate subjects with different skill estimates. Threshold ratings below -2.95 or above 2.95 are also inappropriate, as the skill scale ranges from -3 to 3 in practice. Lastly, a random hit probability above 0.40 is also a critical value. In all these cases, we recommend that the item is excluded from the item bank so that the skill estimate is not compromised [129].

After the final Item Bank Calibration, the team must verify whether the number of items in the bank is sufficient for the instrument's application. According to the instrument's objective, the team must verify whether the items cover all the content [39]. In addition, the team must verify if the items provide adequate information in all extensions of the evaluated latent trait: easy, medium, and complex items.

Suppose the preparation of new items is required. In this case, the team must verify whether it can start the instrument application without significant compromise. Thus, these items can be later added and calibrated gradually according to the maintenance in Step 8. Suppose the number of items remains compromised for the instrument's validity. In that case, it is necessary to elaborate on more items (go back to Step 2) and perform the pre-test and calibration. After completing the calibration process and updating the item bank, the team should build and interpret the scale and proceed to Step 4.

#### **2.4.4 Step 4—Algorithm Elaboration**

For computerized instruments, we can select items using an algorithm. The algorithm has as a requirement the existence of a bank of calibrated items. The algorithm has the definition of an initial selection criterion for the item, the selection method for the other items, the stopping criterion, and the exposure rate of the item [106]. Figure 2.8 presents the details of the Algorithm elaboration step.

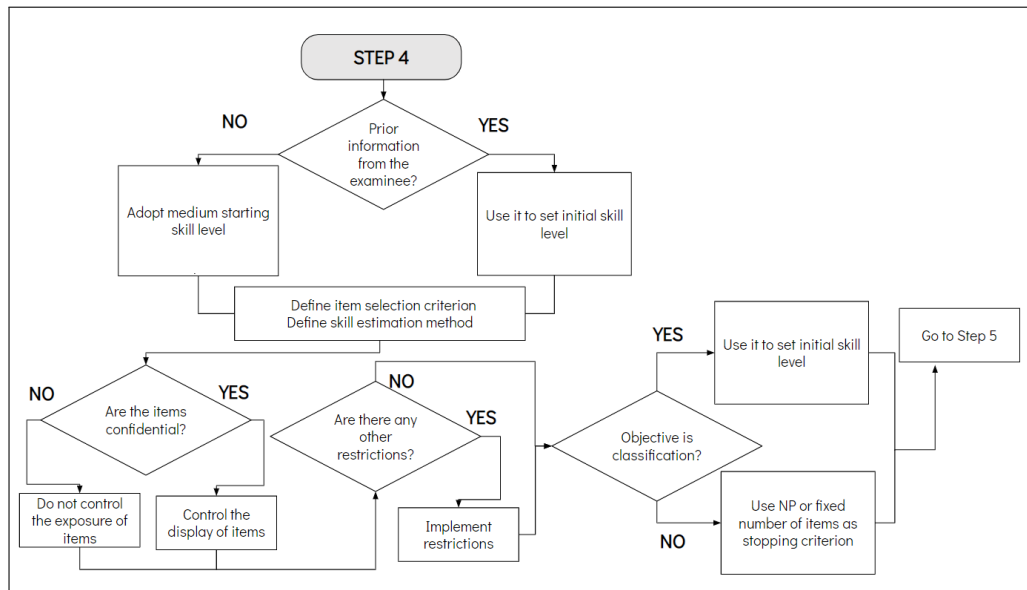


Figure 2.8: Algorithm Elaboration Step.

The initial item selection criteria determine the examinee's degree of prior information. The instrument selects the first item using the appropriate selection criteria. For example, the instrument may use the ability estimated from previous applications, results from other aptitude tests, or other variables related to the measured characteristic. An initial medium skill level is usually adopted when this information is unavailable: fixed, centered on the average, or a random value within the median range. If the instrument uses a fixed value, the first item will always be the same, and its exposure rate will be high. So, this technique is adopted if there is no item exposure rate. If the selection is a random median value, it controls the items' exposure better, reducing the instrument's efficiency.

Suppose we need to get information about the examinee. The team can adopt a fixed mean initial skill estimation level. For example, the skill estimate level on average 0 or a random value within a range, such as between -1 and 1 [14].

The algorithm selects the items using the item selection criteria and the skill estimation method. Item selection can use the most recent skill estimate—such as Fisher's maximum information [101]. Alternatively, it can use the maximum probability when the ability estimate occurs after the individual has responded to an item.

Fisher's Maximum Information selects items aiming to maximize the information in the current skill estimation and Bayesian procedures that select items, minimizing the posterior

variance. The maximum likelihood procedure is an iterative process used to estimate the examinee skill [15]. The process begins with some a priori value for the examinee's skill and the known values of the item's parameters. The team uses these values to calculate the probability of the correct answer for each item. Then the team gets an adjustment for the skill estimate, which improves according to the calculated probabilities of the item's response. The team repeats the process until the adjustment becomes small enough that the change in estimated skill is negligible, resulting in an estimate of the examinee's skill. The algorithm repeats this process for each subject to be tested [14].

The team must check if there is a need to control the items' exposure rate. If the instrument's items are confidential (for example, proficiency instrument), they need to control the item's exposure so that they do not become known and compromise the instrument's reliability. Controlling the item's exposure and using content balancing or any other restriction will restrict the instruments' capacity to reach their best performance [101].

Inevitably, the current item will not be the best (the one with the most information for the given skill level) due to the restrictions defined in the instrument's algorithm. It will result in an instrument accuracy loss since accuracy is related to item information, i.e., the more information the item provides, the greater the accuracy in estimating the skill. For this loss to be insignificant, the item bank must have an appropriate amount of quality items and items with an appropriate amount of information distributed along the scale [39]. In addition, the lower the value set for the item's exposure rate, the greater the number of items in the bank.

Then, the stopping criterion must be defined, which depends on the instrument's objective. Suppose the objective is to estimate a value for the latent trait. In that case, the literature recommends that the estimated skill reach a minimum precision level, a specific minimum standard error [13]. Suppose the objective is to classify or divide respondents into two or more groups (pass/fail). In that case, the literature recommends comparing the estimated skill with one or more predetermined cutoff points.

### **2.4.5 Step 5—Instrument Accuracy and Validity Analysis**

The Instrument's Accuracy and Validity Analysis evaluates the algorithm through some psychometric quality control. This analysis takes place through empirical data or simulations. Figure 2.9 presents the Accuracy and Validity Analysis Step details.

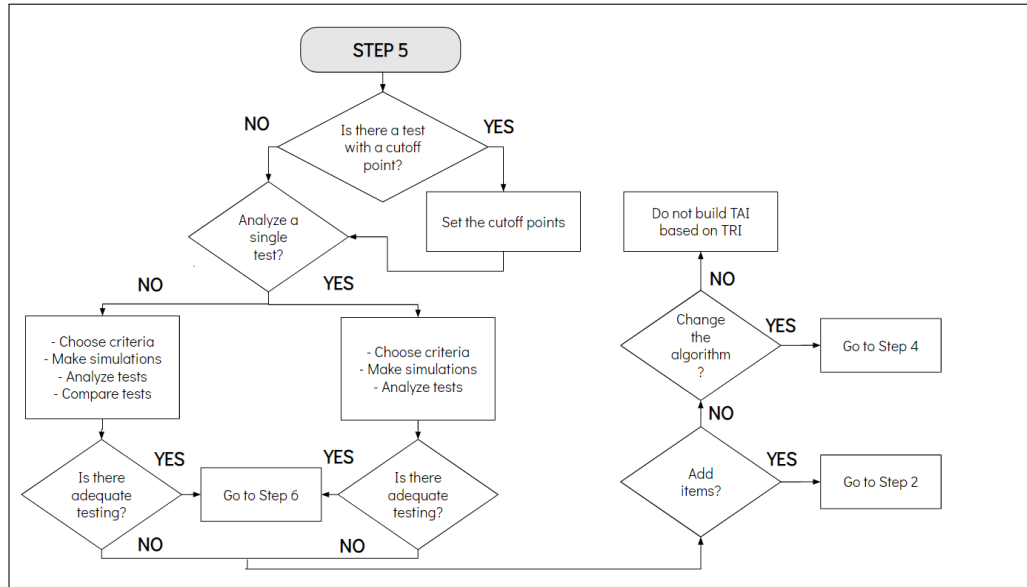


Figure 2.9: Accuracy and Validity Analysis Step.

In this step, the team designs multiple algorithms for the same instrument, combining items, item selection criteria, latent feature estimation methods, stopping criteria, and constraints [44]. For example, the team designs an Items Bank with acceptable parameters and another stricter bank selecting only items that perform above the acceptable level. In cases where the number of items is low, it is possible to assemble banks of different sizes. However, the team tests them through simulations to verify how much the less suitable items affect the instrument's quality.

If the instrument has a cutoff point, the team should define those cutoff points necessary for classification purposes. Regardless of this cutoff point, analyzing the sequential selection algorithm's accuracy is necessary. Suppose the instrument's purpose is to estimate the latent trait using a precision level as a stopping criterion. In that case, it is necessary to consider some criteria for this analysis. Regarding accuracy and validity, we have: the mean; estimated standard error; square root means; square error; empirical deviation means; efficiency, and correlations between simulated skill and estimated skill, among other CTT procedures [14]. It is necessary to analyze the results by observing how much the tested algorithm could recover from the foster skill if the simulated and estimated skills were similar. Then analyze the results according to the criteria that were selected earlier.

The team compares the algorithms trying to observe which one performed better.

After the comparison, the team checks how many instruments performed adequately and significantly better than the others. Suppose the analyzes show an adequate instrument with a performance superior to the others [8]. In that case, it will be implemented (Step 6) and used (Step 7). Otherwise, the team must reformulate the instrument.

Finally, applying a remote hypothesis to the instrument is necessary, verifying the adequate performance for a CAT. If necessary, one needs to go back to Step 2, design and incorporate more items, and calibrate them. If it is possible to change the algorithm, one needs to go back to Step 4. If it is possible to add more items or change the algorithm, building the CAT will also be possible.

### 2.4.6 Step 6—Instrument Implementation

The sixth step includes instrument implementation, which must consider several material resources and a multidisciplinary team. Figure 2.10 presents the Instrument Implementation Step details.

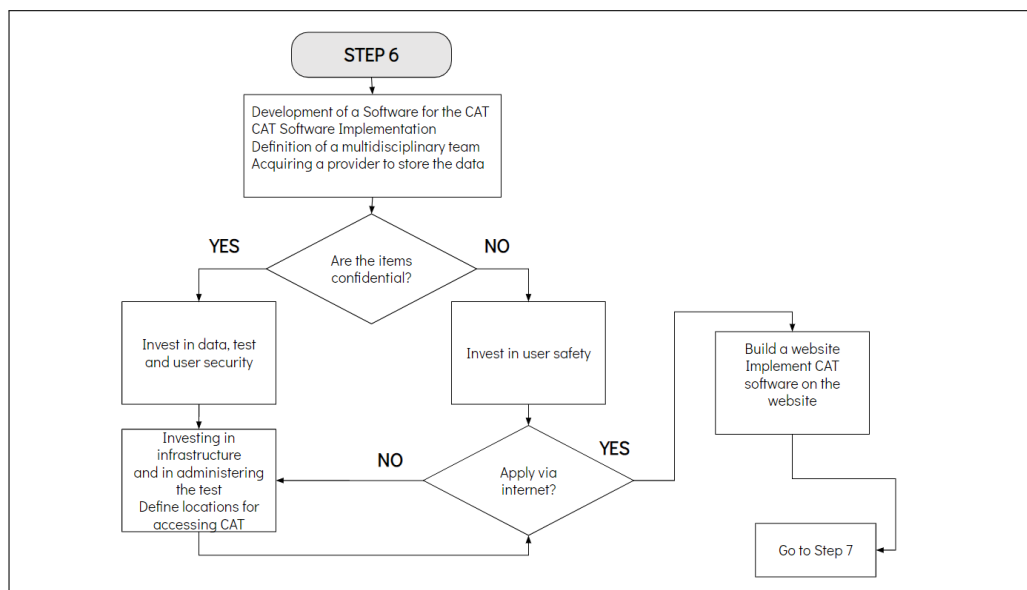


Figure 2.10: Instrument Implementation Step.

It is necessary to consider several factors, including the programming language, item bank security, the users' response security, the material resources, and the location of the application's access terminals. Whether items are confidential, it is necessary to invest in user security [12]. However, if items are already confidential, investment should be made

in better instrument infrastructure, data security, and items management. The team must consider all precautions beforehand if the instrument is offline. If not, it will require the team to build a website/application to manage the CAT and improve security [39].

### 2.4.7 Step 7—Instrument Application

In this step, the team must effectively apply the instrument and provide performance feedback to the examinee. However, the team must build a database with the examinee's responses for later analysis and instrument maintenance. Figure 2.11 shows the Instrument Application Step details.

If the instrument is on paper and pencil, it can be administered by an applicator or virtually (self-administered). The instrument application identifies the person examined a priori; this identification consists of recording the individual's information to control the responses provided by the instrument. Immediately after that, it is necessary to provide feedback to the examinee to provide them with the performance or rating with a summary report identifying the limitations. Finally, the data server stores this data (identification and responses). The team must analyze the answers later for instrument maintenance [14, 101].

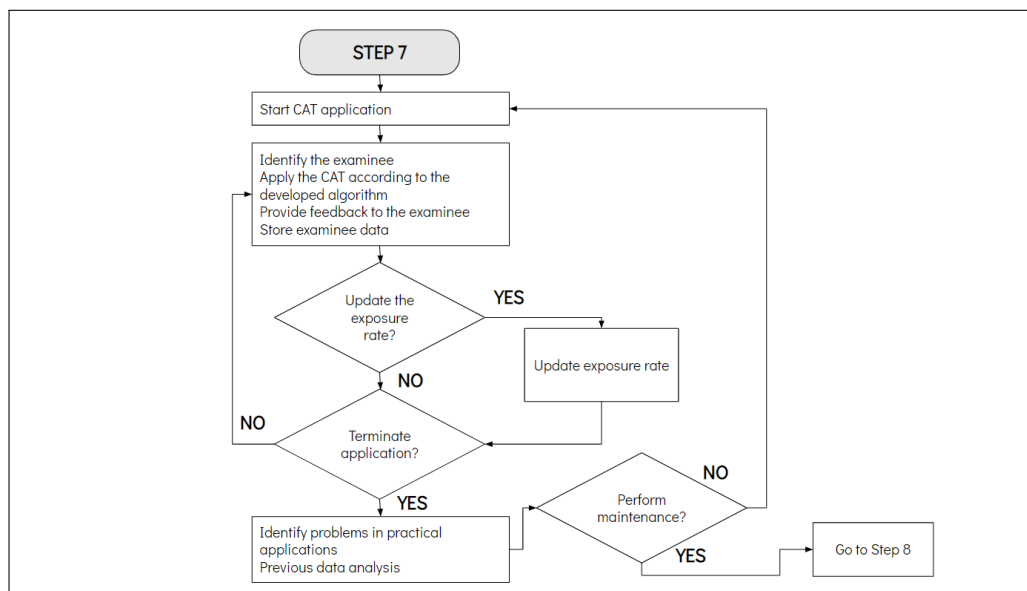


Figure 2.11: Instrument Application Step.

After applying each instrument, it is necessary to verify if it is necessary (and if possible) to update the item's exposure rate in instruments where the items are confidential. If the



instrument applies something heavily, the algorithm should update this rate.

Suppose the instrument's application is sufficient to perform the previous data analysis. In that case, this application must be temporarily closed for this verification. If the application number is still insufficient, the instruments continue to be applied.

### 2.4.8 Step 8—Instrument Maintenance

In the last step, it is necessary to carry out periodic maintenance on the instrument. Figure 2.12 presents the Instrument Maintenance Step details.

When maintenance is required, we must consider some factors related to the instrument. We need to check whether updating the exposure rate of managed items is necessary [101]. If the team designed the instrument to update this rate when applying the instruments, this rate must be updated.

Then, it is necessary to check if any new item added in the last maintenance already has enough responses to be calibrated [14]. This calibration will occur through a suitable equalization method, which uses items in common and placed in the existing scale. The team should review these new items based on the CTT and IRT criteria.

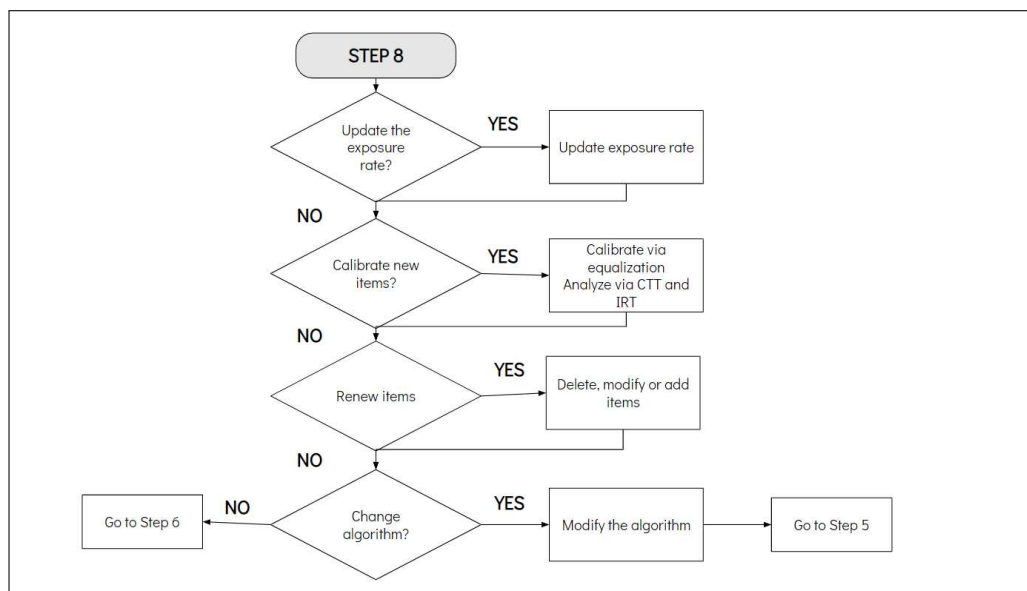


Figure 2.12: Instrument Maintenance Step.

Suppose there is a need to renew items in the bank (delete or add). It is necessary to re-estimate the item's parameters based on the responses collected during the instrument's

application. However, comparing them with the previously estimated parameters is necessary to check whether the item should remain in the database [39]. If the parameter estimates are similar, the item must remain in the bank; otherwise, excluded. These new analyses will verify whether the team can or cannot add the items to the bank.

Finally, it is essential to verify changes in the instrument's algorithm. Changing the item bank will result in differences in its structure. For example, we need to check for changes if the team designs the bank to have more items in a cutoff region or items spread across the entire latent range. Such modifications can affect the algorithm's performance in CAT, which is necessary to modify it [96]. The team can perform simulations in this sub-step, as mentioned in Step 5. Suppose there is a need to modify the algorithm. In that case, the team must perform new simulations and analyze the data according to Step 5. After the instrument's validation, the team must make changes described in Step 6 and other maintenance.

# Chapter 3

## Cognitive Programming Skills

In this Chapter, we identify cognitive programming skills and approaches for measuring and fostering those skills. We defined the SLM protocol and presented the search and its review results.

### 3.1 Initial Considerations

CS1 has low pass and retention rates, and students starting in CS1 enter with a broad range of skill levels. To deal with the differences in skill levels, educators must better understand the teaching practices. Educators can opt for easier tasks to simplify student learning and decrease failure and dropout rates [109].

Previous and recent SLM papers have addressed the students, educators, curriculum, assessment, and trends in CS1 [77]. Alternatively, focus on categorizing introductory programming challenges in Higher Education [85]. However, there is a need for a consensus on how institutions should work on cognitive skills during a CS1, as well as a clear and complete categorization.

Due to the need for an overview of this area, this Chapter has the following objectives:

- **SO1.** To identify cognitive programming skills;
- **SO2.** To identify approaches to measure cognitive programming skills;
- **SO3.** To identify approaches to foster cognitive programming skills.

Thus, we will answer the following research questions:

- **RQ1.** What are the cognitive programming skills?
- **RQ2.** How to measure cognitive programming skills?
- **RQ3.** How to foster cognitive programming skills?

To answer these research questions, we designed an SLM following the guidelines proposed by Kitchenham [64]. This Chapter presents the results as the first contribution to cognitive programming skills knowledge. We organize the remainder of this Chapter as follows. In Section 3.2, we describe the study design. In Section 3.3, we present the results. In Section 3.4, we discuss the results. In Section 3.5, we discuss the related works. Finally, in Section 3.6, we conclude the Chapter with final remarks.

## 3.2 Study Design

This section presents the SLM design to identify studies that describe cognitive programming skills and approaches to measure/foster these skills.

### 3.2.1 Search Strategy

Our search strategy consisted of an online search in the four main digital libraries with high relevance for CS, namely: IEEE Xplore [59], ACM Digital Library [1], ScienceDirect [113], and Scopus [114].

Search keywords are essential for the quality and results coverage, so they should be carefully defined to search online digital libraries. The query has a raw string composed of three terms, one standard term and the other consisting of a combination of identified keywords' synonyms. The first term (A) refers to Introduction to Programming and its synonyms, the second term (B) refers to novice programmers or CS1, and the third term (C) refers to skill and its synonyms. We presented the keywords and synonyms for each term in Table 3.1.

Term	Keywords	Synonyms
A	Introduction to Programming	Programming Course, Programming Language, Programming Learning, Learning Programming, Programming Teaching, Teaching Programming
B	Novice programmers	CS1
C	Skill	Expertise, Ability, Proficiency, Experience, Art, Technique, Facility, Talent, Intelligence, Craft, Competence, Readiness, Accomplishment, Knack, Ingenuity, Finesse, Aptitude, Dexterity, Cleverness, Quickness, Adroitness, Handedness, Skillfully

Table 3.1: Terms, Keywords, and Synonyms Used to Create the Search Query.

We performed a pilot search on the IEEE digital library to evaluate the search query. We performed a full-text search and the publications' metadata (title, keyword, and abstract). After some trial and error with a databases range, we selected a combined search string that seemed to capture the area of interest:

((“introduction to programming” OR “programming course” OR “programming language” OR “programming learning” OR “learning programming” OR “programming teaching” OR “teaching programming”) AND (“novice programmers” OR CS1) AND (skill OR expertise OR ability OR proficiency OR experience OR art OR technique OR facility OR talent OR intelligence OR craft OR competence OR readiness OR accomplishment OR knack OR ingenuity OR finesse OR aptitude OR dexterity OR cleverness OR quickness OR adroitness OR handedness OR skillfulness))

### 3.2.2 Study Selection Strategy

We explicitly set the inclusion and exclusion criteria in reviewing the protocol for the SLM. Primary studies must meet the following inclusion criteria: **(IC)**: studies that identify cognitive programming skills in CS1. We considered some exclusion criteria in the SLM, namely:

- **(EC1)**: preliminary studies, unavailable and duplicate studies (papers with the same or updated version, keeping only the most recent);

- **(EC2):** the study is a short paper or SLM;
- **(EC3):** studies are not written in English<sup>1</sup>;
- **(EC4):** studies published before the last decade;
- **(EC5):** studies that do not identify cognitive programming skills in CS1.

We organized our study selection process in four distinct phases, described below:

- **Phase 1:** as a preliminary selection, we performed the queries, applied EC4 and EC5, and defined the study group that served for the second phase;
- **Phase 2:** based on the titles, abstracts, and words of the preliminary selection studies, we determined and kept which studies were relevant;
- **Phase 3:** based on the inclusion and exclusion criteria and full reading of the study, we reviewed relevant studies from the previous phase;
- **Phase 4:** A specialist evaluated and validated the selected studies, with the studies' inclusion or exclusion criteria.

### 3.2.3 Data Extraction

We prepared a form to extract and synthesize relevant data to answer the research questions in the SLM protocol. Data extraction aims at summarizing data from primary studies. We define the items and their descriptions and present them in Table 3.2.

Common items	Descriptions
Title	Paper title
Year	Paper publication year
Skill	Cognitive programming skills described in the paper
Validation	Study's validation
Foster	Studies that foster cognitive programming skills
Measure	Studies that measure cognitive programming skills

Table 3.2: Description of Extracting Data From SLM.

<sup>1</sup>We consider English a universal language, and it is easy for the researcher to understand it, unlike other languages that could be difficult in the translation process. Although the main Brazilian researchers publish in national conferences/journals, they also publish versions of their articles in international vehicles (generally in English).

The SLM process defined in the previous section resulted in 5063 articles found in the four databases. After Phase 1, we selected 262 studies for Phase 2, in which we identified 67 studies [2, 4, 9, 11, 16, 17, 19–21, 23, 28–33, 35–38, 40–42, 47, 49, 51–53, 55–57, 60–62, 69, 71–73, 78, 82–84, 86, 87, 89, 90, 92, 93, 95, 100, 102, 108, 115–117, 119, 120, 124, 130, 133, 135–140] as relevant after a careful reading of the paper. In the works whose principal researcher had difficulties accepting or not the paper in the inclusion criteria, we consulted specialized university professors in programming teaching to solve these doubts. We show the study’s details in Table 3.3 by the library and Table 3.4 by phase.

Digital Library	Search result
IEEE Xplore	421
ACM Digital Library	1,807
Science Direct	297
Scopus	2,538
<b>Total studies</b>	<b>5,063</b>

Table 3.3: Details of Study Search and Selection by the Database.

Phase	Descriptions	Included	Excluded
Phase 1	Search results	5,063	0
Phase 2	Title and abstract selection	262	4,801
Phase 3	Full reading selection	67	195
Phase 4	Selection validated by a specialist	67	0

Table 3.4: Search Phase Study Details.

### 3.2.4 Threats to Validity

Although our research followed the protocol, there are some threats to validity. The selected studies are directly related to the search engine of virtual libraries, as each has specific characteristics. Therefore, the search we performed in libraries may have returned only some relevant studies. Furthermore, we only consider complete studies published in journals and scientific events (conferences, symposia, workshops). Relevant studies may be outside the SLM’s scope, which is introductory programming instruction. To mitigate this threat, we have chosen several high-end libraries in CS to return as much relevant work as possible.

Furthermore, human factors can also influence data extraction. The researchers may have yet to notice some relevant information or even misinterpreted it, as data extraction is a manual process. However, the authors examined the selected studies more than once to mitigate this threat.

### 3.3 Results

In this section, we present the obtained results. In the subsections, we answer each previously defined research question.

#### 3.3.1 What are the Cognitive Programming Skills?

We have identified a set of cognitive programming skills in Table 3.5. Cognitive programming skills are tracing, explaining, comprehension, reading, debugging, modifying, and writing.

Skill	Studies
Tracing	[41, 42, 51, 52, 54, 55, 108, 116]
Explaining	[11, 53, 83, 89, 116, 119, 130]
Comprehension	[28, 30, 32, 35–38, 73, 86, 92, 93, 117, 139]
Reading	[19, 20, 51, 57, 84, 90, 102, 108, 137]
Debugging	[2, 9, 16, 21, 23, 29, 35–38, 41, 42, 51, 69, 77, 83, 84, 87, 95, 108, 124, 133]
Modifying	[16, 86, 108, 116, 139]
Writing	[16, 17, 19, 20, 32, 33, 40–42, 47, 49, 51–54, 57, 60–62, 69, 82–84, 86, 89, 90, 92, 93, 100, 102, 108, 116, 117, 135–140]

Table 3.5: Cognitive Programming Skills Identified in the Literature.

#### 3.3.2 How to Measure Cognitive Programming Skills?

In Table 3.6, we present the instruments that measure cognitive programming skills.



<b>Skill</b>	<b>Instruments</b>
Tracing	Universiti Teknikal Malaysia [54] PyKinetic [41,42] University of Toronto Scarborough [52] Benchmarking Programming Performance [116]
Explaining	Uppsala University [53] E-assessment [83] Benchmarking Pacific Lutheran University [89] Programming Performance [116] University of Newcastle [119]
Comprehension	New York City College of Technology [86] Medgar Evers College [139]
Reading	Freie Universität Berlin [19] University of Washington [137]
Debugging	Kent State University [2] University of Oslo [16] PyKinetic [41, 42] E-assessment [83]
Modifying	University of Oslo [16] New York City College of Technology [86] Benchmarking Programming Performance [116] Medgar Evers College [139]
Writing	Universiti Teknikal Malaysia [54] University of Oslo [16] TUM School of Education [17] Freie Universität Berlin [19] Predicted Difficulty Index [33] PyKinetic [41,42] Tel-Aviv University [49] University of Toronto Scarborough [52] Uppsala University [53] The University of Adelaide [61] E-assessment [83] New York City College of Technology [86] Pacific Lutheran University [89] Benchmarking Programming Performance [116] AUT University [135] University of Washington [137] Walchand Institute of Technology [138] Medgar Evers College [139] University of Toronto [140]

Table 3.6: Instruments That Measure Cognitive Programming Skills.

### 3.3.3 How to Foster Cognitive Programming Skills?

In Table 3.7, we present the methods, approaches, and tools that foster cognitive programming skills.

Skill	Methods	Approaches	Tools
Tracing	-	Program Memory Traces [55]	-
Explaining	-	Self-Explanation [130]	-
Comprehension	B-learning Model [117] Graphical Language [124] Schulte's Block Model [30]	Kane's Framework [92] Slicing Technique [35, 38]	PLTutor [93] Thinkathon [28]
Reading	-	-	C-doku [57] Java Programming Learning Assistant System [90]
Debugging	-	Slicing Technique [35, 38]	Debugger Tool [95] Debugging Teaching Environment [9] Ladebug [78] Quiz Summary [133]
Modifying	-	-	-
Writing	B-learning Model [117]	Kane's Framework [92] Writing-to-learn [60]	C-doku [57] CodeSpells [40] Java Programming Learning Assistant System [90] PLTutor [93]

Table 3.7: Methods, Approaches, and Tools That Foster Cognitive Programming Skills.

### 3.4 Discussion

This section discusses the data presented in the previous section on the studies that characterize cognitive programming skills, highlighting methods, approaches, tools, and instruments that foster/measure such skills.

### 3.4.1 Cognitive Programming Skills

Programming requires many different skills. This subsection explores the different programming skills found in the literature, such as tracing, explaining, comprehension, reading, debugging, modifying, and writing.

#### Tracing

Tracing is a skill that involves hand-tracing the code by executing it mentally or with paper and pencil (for example, code trace to get a variable value) [108, 116]. The novice programmer should learn to trace code before writing it [52]. There is evidence on code tracing, code writing, and code explanation [51, 92, 93]. There is a strong positive correlation between tracking code and writing code [54].

A study involving 384 students investigated the limitations of students' ability to trace and write code. In the exam environment, participants answered two questions and were randomly assigned to perform code tracking in one question and code writing in another. Results showed that 56% of participants had almost no difference in performance between code tracking and writing skills. There was a strong negative correlation between student performance and gap size for the remaining students. They scored at least two of the eight grades. Regardless of whether the student is better at tracking or writing code, the study suggested that a significant gap is likely due to student difficulties in the course. The students needed help understanding key programming concepts [52]. These findings prove that good tracing skills also have a higher order of code writing skills.

The novice's debugging skills positively correlate with the code tracing skills. In addition, code tracing skill is also positively correlated with code modifying skill. Lastly, code modifying skill is also positively correlated with code debugging skills. All these correlations are solid and significant. They show evidence that students with less prior knowledge perform similarly in various coding activities, from debugging, tracing, and code correction [41, 42].

An experiment using a tool to improve code tracing skills presented reasonable expectations in the students. The code tracing activities exposed students' unfeasible models and, more importantly, provided the clearest starting point for discussing the correct models.

While this should have encouraged students to track down the code, they would avoid these questions and guessed the program's final state. As other studies found, students needed more support to do any tracing independently. This experiment showed that code tracing helps students understand code rather than just exploring material that fosters writing code. In addition, a teaching approach based on tracing improved the students' learning. In which students showed statistically significant improvements in grades in programming. There is evidence that the ability to track code helps students to develop viable programming concepts and increases students' course completion rate [55].

Furthermore, there is evidence that underachieving students often have a significant gap in coding skills, as they are more likely to struggle with basic programming concepts. Basic programming concepts are essential for learning to debug, trace, and correct coding skills [41,42].

### **Explaining**

A code reading or explaining question presents students with a small snippet and asks them to explain it, not line by line, but in general-purpose terms. Students spontaneously explain the content's meaning in self-explanatory studies, typically text passages or other written material, as they study a target domain. Educators can compare self-explanations to any learning gains demonstrated by students [130]. However, the students who pass CS1 seem unable to explain the simple code snippets' purpose, like a loop, to find the most significant element in an array [119].

Several recent studies have explored the relationship between novice programmers' ability to read and explain code and their ability to write code [11, 53, 83, 89, 116, 119, 130]. There is a strong correlation between the skill to answer a question about the code and explain the code in simple English questions correctly. The skill to write code indicates that the code's reasoning aspects are common to both writing and explaining code [89].

One study compared students' responses across multiple offerings from a CS1. Students expressed the purpose in their own words; in later offerings, they chose the purpose among several options in a multiple-choice question. Students performed significantly better on the multiple-choice question; on a local campus, performance was better but insignificant. Many students needed help identifying the correct purpose of small code fragments when given

that purpose and some alternatives. The students need to perform better on code explanation questions because they cannot recognize it in a different way [119].

One approach to helping students learn to program consists of self-explanation tasks, and students explain instructional materials using domain knowledge covered in the course in these assignments. The experiment showed that the experimental group that received self-explanatory tasks with supporting questions performed better on the test questions when compared to the control group. This evidence argues that incorporating self-explanatory questions into programming material benefits students [130].

There are reasoning aspects of code that are common to both writing and explaining code. There are specific reasoning skills for writing code and specific skills for explaining code. However, the results of a survey suggest that there is a possibility that common reasoning skills for both writing code and explaining code are not trivial. The results suggest other ways of teaching and learning to program in addition to writing code, given the statistical relationships between student performance in explaining and writing code [89]. These other ways focus on the common elements of reading, writing, and explaining code. Tracing, reading, and writing code can lead to a more effective and efficient process by which many novices would learn to reason about code.

### **Comprehension**

Comprehension consists of the student's skill to understand a code piece or a program [86, 117, 139]. Ways to assess this skill include: i) asking students to select the code piece from an option set that performs a specific task; ii) describing what a program does; indicating what the return value is when calling a function; iii) indicating what a program would display on the screen when running; iv) indicating the state of one or more variables after executing a program; v) identifying the part of the program that the compiler executes in a certain action.

Traditionally, educators work on code comprehension in CS1 through sample solutions, design road-maps, implementation tips, and early code, emphasizing code writing. Providing students with these resources is not a code-understanding practice; it is not the focus, and educators rarely verify this skill in students [86, 139].

Code comprehension skills are a precursor to code writing skills [32, 36, 37, 92, 93, 139]. The educator should ensure that students exercise code comprehension before asking them to

write code [86, 139]. Code comprehension skills in the early stages of programming courses can ensure good learning outcomes [30].

Understanding the programming language requires high cognitive skills (e.g., reading, writing, working memory) and information processing. This difference is evident when we compare an expert with a programming novice. One study in neuroscience compared novice programmers and experts during a program comprehension tasks series [73]. Experts exhibit greater brainwave activation than novices. These results indicate that experts have excellent skills associated with program comprehensions, such as digit coding, coarse coding, short-term memory, and subsequent memory effect.

There is also evidence that more than code comprehension skill is needed to be a forerunner in programming [28]. This finding contradicts other research studies indicating that understanding code is a significant step in the development process of novice programmers. This study analyzed a CS1 and found that students' performance on the final exam was unrelated to code comprehension practice. In the study, final exams required writing and code comprehension exercises.

### **Reading**

Programming consists of a process that connects reading and writing code. Until now, research in Computer Education has always focused on the writing part [20]. Active code reading is a particular case of active learning [51, 57]. Many instructors believe students would write better code if they read it more. Unfortunately, we need precise goals and practical evaluation mechanisms to understand the code. Many students can learn more about programming if they take the time to read code [57].

In this scenario, code reading is an essential programming skill [19] [20, 90, 102]. It is inspired by people's linearity when they read natural language text. One study designed local and global gaze-based measures to characterize linearity in reading source code. Linearity involves looking at code from left to right and top to bottom. Unlike natural language text, source code is executable and requires a specific reading approach. To validate these measures, the authors compared the eye movements of novice and expert programmers when reading and understanding small snippets of natural language text and Java programs. The results showed that novices read source code less linearly than natural language text.

Furthermore, experts read code less linearly than novices. These findings indicate differences between natural language and source code reading and suggest that nonlinear reading skills increase with experience [19].

The code reading task provides students with a short program in the appropriate language and a series of questions about the code execution. Understanding measurement in this task presents users with questions about the variables' values at various points during execution [84].

Another study showed that the explicit indication of code reading instructions and their practice improves student skills in code writing in the reinforcement and evaluation activity [137]. Students who practiced semantic reading and reading models skills performed better in code writing when compared to participants in the control group.

However, learning to write code is a complex activity [108]. The scientific community widely accepts that students would write better if they spent more time reading code. Some tools like C-doku aim to fill this need by improving code reading skills [57]. An instructor can motivate students to focus on selected aspects of a code snippet, and this tool reinforces that practicing code reading skills helps to improve code writing skills.

### **Debugging**

Debugging is an essential skill that is difficult for novice programmers to learn and challenges educators to teach [133]. Debugging is a necessary aspect of a CS course that can be difficult for novice and experienced programmers [21, 29]. Debugging involves reading code, tracing it, and mentally running it according to the programming language's rules in question [51]. This skill is usually self-taught and acquired through trial and error, perhaps with the help of educators or other experts [21].

Debugging is essential for all programmers, especially novices still learning the programming language's syntax and semantics. Therefore, novices are more likely to write incorrect code than experienced programmers [51, 78]. Novice programmers often find it challenging to perform debugging tasks effectively [9, 35–38]. Students need to understand how the program works and run the program with bugs, know the application domain and the programming language, and know about (specific) bugs and debugging methods. Moreover, students' systematic exposure to error types different from those found in their learning

environment can improve debugging skills [2, 108, 124].

However, most students believe that debugging depends on individual aptitude. Furthermore, students believe that debugging code develops through learning [87]. Most students do not use tools for debugging the environment or even know them [23, 83, 84]. Some students often apply a trial-and-error approach to programming errors. Compile-time errors represent a significant obstacle for many students. Educators are running from computer to computer trying to help [87]. Error messages help novices to find and fix errors, but compiler messages are often uninformative [29].

The debugger improves the conceptual understanding of novices, and Low-achieving students benefit most from this experience. The debugger can help minimize logical errors and improve writing skills [16, 95].

Therefore, self-sufficiency in debugging is essential and a significant challenge for learning to program. However, educators focus on heuristics for common mistakes and debugging strategies related to teaching debugging skills [69]. Educators need to conduct a systematic process for dealing with errors. Furthermore, they need to employ explicit teaching lessons on debugging. Educators need a more systematic approach to teaching debugging, as there are only vague concepts and materials [41, 42, 87].

### **Modifying**

Modifying involves using or changing the existing code [16, 86, 108, 116, 139]. The educator can foster this skill in their students with exercises, namely: i) completing a code snippet (either writing the missing code or choosing it from a set of options); ii) writing function calls to certain functions so that a specific result can be given; iii) build a program from a set of code fragments, not all which can be part of the solution; iv) reordering a scrambled program (also known as Parson's programming puzzles).

### **Writing**

Writing consists of the student's skill to write code for a particular task [16, 17, 19, 20, 32, 33, 40–42, 47, 49, 51–54, 57, 60–62, 69, 82–84, 89, 90, 92, 93, 100, 102, 108, 116, 117, 135, 136, 138, 140]. The standard assessment method provides students with a problem specification for writing a program. However, other skills are needed to write code, which educators do



not pay much attention to promoting in traditional education, such as:

- Identify programming constructs (variables, data types, expressions, function definitions, function calls, and parameters);
- Explain the code (to be able to explain what a piece of code does verbally);
- Understand the technical documentation (i.e., Error messages help novices to find and fix errors, but compiler messages are often inappropriate);
- Refactoring existing code.

Several studies support the premise that novice programmers would learn more effectively and efficiently if they spend more time deconstructing code than writing code. Deconstructing code involves reading, tracing, and debugging code skills [31, 51, 71, 72, 115, 120, 137]. Educators can adopt cognitive science principles in the learning process to solve problems. The progression starts with low-risk deconstructionist activities, such as exploring, identifying, comparing, and debugging, before activities that require writing code [51].

### 3.4.2 Measuring Cognitive Programming Skills

Several studies use the **CTT** to measure cognitive programming skills [2, 19, 32, 33, 41, 42, 49, 52–54, 83, 86, 89, 108, 116, 119, 137–140]. However, some universities have adopted other theories and taxonomy for this practice, such as **IRT** [16, 17], **SOLO Taxonomy** [49, 61, 89, 102, 117, 120, 135–137] and **Bloom's Revised Taxonomy** [31, 71, 72, 108, 115, 120]. We define these theories and taxonomies in Chapter 2.

### 3.4.3 Fostering Cognitive Programming Skills

On the one hand, the number of employees in the IT industry is continuously growing. On the other hand, there is a growing need for more IT specialists, especially in software development. This problem can be solved by increasing student interest in computer programs or decreasing the number of students who drop out of the course early. While the first solution requires the involvement of all stakeholders, influencing

students (parents, teachers, schools, friends), in the second, students have already decided to study programming. Therefore, it is “easier” to invest time and resources in designing an appropriate educational concept. Which could increase the likelihood that students will successfully graduate and have the knowledge and skills required by the job market [120].

With the advent of the internet and the popularization of information, many young people prefer to be self-taught. They choose what they want to study and are not interested in listening to lectures or watching long-term educational videos. They prefer learning anytime, anywhere, not just at university. They prefer the immediate use of applying the knowledge and skills gained. In addition, they prefer to select the knowledge and skills whose usefulness they can imagine or prove in a short time.

Innovative educational models and frameworks that minimize obstacles and meet student requirements and expectations use well-known learning approaches and develop the skills and knowledge needed for taxonomies. Programming teaching is a complicated educational process. Educational theories and taxonomies are valuable tools for developing learning objectives and assessing student performance. However, they do not apply directly to this area. Programming requires students to understand the relevant theory and apply it to solve real problems [120].

Researchers use different approaches to foster programming based on different educational theories, teaching structures, or educational approaches. Some studies have sought to simplify the acquisition of knowledge’s complexity and skills. On the other hand, other studies described and mapped the skills accurately. In the following subsection, we discuss methods, approaches, and tools to foster cognitive programming skills.

### **Tracing**

**Program Memory Traces** consists of an approach that fosters tracing skills. Program Memory Traces accurately represents the program’s memory usage to balance the abstraction required by novice students with the precision needed for advanced students. Program memory traces represent all the program’s accessible memory, dividing the tracking area into different regions representing memory’ types differently: stack, heap, and static. In a program memory trace, program execution begins with a method call. The Program memory traces present an accurate memory model that educators can use to explain programming

concepts to students [55].

### **Explaining**

**Self-explanation** is an approach that fosters explaining skills. Self-Explanation is an approach educators use for students to explain the content's meaning spontaneously. Usually, the content is text passages or other written material used to study a target domain. The resulting explanations are coded and compared to any learning gains demonstrated by students. Several studies in various fields, particularly science and mathematics, have found that high-quality self-explanations positively correlate with learning gains. In the programming context, students who use the material with self-explanatory questions perform significantly better on the questions of an explanatory exam than those without self-explanatory questions [130].

### **Comprehension**

There are several methods able to foster comprehension in novices programming, namely: B-learning Model [117], Graphical Language [124], and Schulte's Block Model [112].

**B-learning**, sometimes called blended learning, is a teaching and learning approach combining traditional face-to-face instruction and distance learning. Educators can provide novices with programs to read and explain in the web-based learning platform. The B-learning environment on program learning can result in marked improvements in pass rates and positive student evaluations [117].

B-learning provides a way to improve the learning of novice programmers. On the other hand, reading and explaining computer programs with a B-learning learning method is a challenge for many novice programmers. Roles represent programming knowledge higher than simple knowledge of a programming language. The results show significant improvements, including novice's performances on the final examination and the ability to read and write a program. These results indicate that the B-learning with roles of variables to provide scaffoldings virtually affects the learning of novice programmers [117].

**Graphical Language** is a method to improve students' understanding of programming techniques, enhancing students' problem-solving skills. The method focuses on teaching students graphical representations of program techniques because studies have shown that

humans remember pictures more efficiently than text. Students' overall feedback showed they could understand programming terminologies introduced using graphical languages. Side notes from quizzes and assignments showed that students use graphical approaches to help themselves understand problems. For example, they use tables for tracking loops [124].

**Schulte's Block Model** is a model that supports educators' understanding of how to divide the skill into parts that are more manageable for a student [112]. The model considers the code understanding at four levels:

- The atom level, which considers programs at the level of the smallest program components, may be exemplified by a single assignment statement, an input or output statement, or even the expressions contained within these components;
- The block level considers contiguous lines within a program that are bounded logically, for example, the body of a loop or a branch in a selection statement or a subprogram body;
- The relations level considers separated lines of code that are related to one another this could be relevant to variable roles;
- At the top-level, macrostructure concerns the understanding of the whole program.

Different approaches foster comprehension skills: Kane's Framework [92] and Slicing Technique [35, 38]. PLTutor [93] and Thinkathon [123] are tools that foster comprehension skills.

**Kane Framework** designs a formative assessment of program tracking, developing a compelling argument for a specific use that includes: 1) a refined scoring model to guide practice; 2) items' design to test parts of the refined model with low variation caused by confusion; 3) a coverage test project that shows samples of an item space and covers the scoring model and 4) a feasibility argument for effective informative use (can guide and improve learning). The framework contributes a new way of modeling possible conceptions of a programming language's semantics. The framework modeling predominant compositions of control flow and data flow graphs and the paths through them, a process to generate test items and principles to minimize the items' confusion [92].

**Slicing Technique** has been widely used in software testing, debugging, and assessment [35, 38]. For example, while debugging, there could be syntactic and semantic errors within the source code which could throw multiple errors without showing the code, which caused the bug. Slicing helps overcome this by modularizing the entire code into slices to debug the erroneous code easily and reduce the compilation time by improving program code. This approach also helps learners to acquire better programming skills;

**PLTutor** is an instrument that allows building the experience of using a debugger with the following features: 1) allows stepping forward and backward in time, 2) allows stepping at an instruction-level instead of line-level granularity, and 3) interleave conceptual instruction on semantics throughout the execution of the program. PLTutor also conveys constraints, functional forms, and ontology by showing natural language explanations of action instructions. PLTutor reinforces functional forms by showing an instruction form as a description filled with concrete values. PLTutor presents learning gains among CS1 students [93].

**Thinkathon** is a tool that works on the ability to comprehend code. The Thinkathon idea was born from the realization that students were not developing appropriate conceptual comprehension. This approach aims to provide an extended immersive experience [123].

## Reading

C-doku [57], and Java Programming Learning Assistant System [90] are tools that foster reading skills.

**C-doku** evaluates code-reading skills with quizzes containing four simple question types. The tool showed results that support the claim that its practice improves the ability to write code [57]. With C-doku, an instructor can motivate students to focus on selected aspects of the code pieces. The instrument has four principles: active code reading, template-based, automated oracle, and strong authoring support.

**Java Programming Learning Assistant System** provides the fill-in-blank problem to support the self-studies of novices in Java programming. The fill-in-blank problem aims to improve students' self-studies for learning grammar and basic programming skills through code reading [90].

## Debugging

Debugger Tool [95], Debugging Teaching Environment [9], Ladebug [78] and Quiz Summary [133] are tools that foster comprehension skills.

**Debugger Tool** improves the conceptual understanding of the programming language and the program's execution flow for novices. Students discover and correct logical errors when using a debugging tool for their program. Furthermore, they can understand the program's execution flow, which helps them predict the output correctly [95].

**Debugging Teaching Environment** is an extension of the Eclipse Integrated Development Environment platform. It exploits the Eclipse Modeling Framework to implement its guidance model. The Debugging Teaching Environment receives the buggy system's source code as a workspace project and the metadata for the bugs specified by the instructor. The Debugging Teaching Environment provides the contextually evaluated suggestion. In this way, the developer only focuses on the subset of source code elements with direct or indirect dependencies on the bug site [9].

**Ladebug** explicitly teaches debugging skills, finds flaws in the code, provides feedback, and encourages repetition [78]. Finally, **Quiz Summary** is a web-based software tool for assessing the debugging skills of CS students. The Quiz Summary creates debugging skills assessment questionnaires. After identifying the student's skill level, the application recommends a list of tutorials and practices to improve the debugging skill. The application tracks student activities such as assessment results and the tutorial and completion status of the exercises. This relationship allows the application to deliver skills-based material to everyone [133].

## Writing

The B-learning Model is also a method that fosters writing skills. Writing-to-learn [60] and Kane's Framework are approaches that foster writing skills. C-doku [57], CodeSpells [40], Java Programming Learning Assistant System [90] and PLTutor [93] are tools that foster writing skill.

**Writing-to-learn** is a critical reflective act. There is a correspondence between writing and learning strategies to improve the learning of the content of the course. Writing to learn

is a technique for learning programming. Thus, this technique improves student learning in CS1 [60].

**CodeSpells** is a game that engages students in introductory programming concepts similar to Scratch<sup>2</sup> but using Java. CodeSpells provides students with a magic metaphor that tries to mimic the CS culture. The game involves students in the safe writing of Java code [40].

We did not find studies addressing methods to foster tracing, explaining, reading, debugging, and modifying skills. None of the studies present approaches that foster reading and modifying skills, and we did not find any tools that foster tracing, explaining, and modifying skills. Based on this limitation, new research should emerge to increase the discussions that such skills are essential for learning programming. This research can support the need for empirically tested tools to foster them.

### 3.5 Related Works

Existing studies address instruction/assessment challenges in CS1, alternative methods, formative feedback, or mention some programming skills [77,85]. These studies are essential because they provide a fundamental overview of CS1 and indicate some skills that should be fostered and evaluated but do not categorize them. However, to our knowledge, an SLM has yet to categorize students' skills in learning to program.

The works of Luxton et al. and Medeiros et al. characterized the challenges in CS1. They highlight some fundamental skills for a novice student to learn to program and their difficulties in this process. The main teaching challenge concerns the need for adequate methods and tools for personal teaching. Problem-solving remains one of the leading learning challenges, followed by motivation, involvement, and difficulty in learning the programming languages syntax [77,85].

This Chapter introduces an SLM that categorizes cognitive programming skills and highlights approaches that foster and measure those skills. This Chapter's contributions

---

<sup>2</sup>Scratch is one of the most popular visual programming languages. Many institutions use it to develop computational thinking skills, fundamental computational concepts [107], algorithmic-level thinking skills [26], and problem-solving [66].

guide higher education institutions in formulating their curricula and assessment instruments. Such practices can foster programming skills in novices, thus reducing the students' difficulty, consequently, the dropout and failure rates in CS1.

## 3.6 Final Considerations

This Chapter has presented a cognitive programming skills categorization and approaches to foster/measure those skills. Therefore, we defined the SLM protocol and presented the search and the results. As main results, we found that, in recent years, several studies have shown that:

- **(RQ1:)** Cognitive programming skills are tracing, explaining, comprehension, reading, debugging, modifying, and writing;
- **(RQ2:)** Researchers use CTT to measure cognitive programming skills. However, some universities have adopted other theories and taxonomies for this practice, such as IRT, SOLO Taxonomy, and Bloom's Revised Taxonomy;
- **(RQ3:)** Researchers use different approaches in programming teaching based on different educational theories, teaching structures, or educational approaches.



# Chapter 4

## A Cognitive Domain Adaptation to Bloom's Revised Taxonomy

In this Chapter, we present a proposal for adapting the cognitive domain of Bloom's Revised Taxonomy to programming teaching and the cognitive skills' association raised in the literature within the Adapted Taxonomy. To validate the proposal, we analyzed it theoretically through a Survey applied to an expert group in CS1.

### 4.1 Initial Considerations

In Chapter 3, we identified the most cited cognitive programming skills in the literature and the existing approaches to fostering and measuring them. However, we did not find unanimity in the literature about which skills educators foster in a CS1. This sequence involves the construction of linear knowledge; that is, the previous skills influence the later skill. The sequence of activities involved in learning programming can be progressive, allowing the student to learn without frustration. Therefore, this sequence directly affects the instructional design in CS1. This Chapter aims:

- **SO4.** To sequence cognitive programming skills to determine the appropriate challenge level in an assessment instrument.

In this chapter, we answer the following research question:

- **RQ4.** How to sequence the cognitive skills involved in introductory programming learning to determine the appropriate challenge level in an assessment instrument?

To answer this question, in Chapter 3, we present the cognitive domain of Bloom's Revised Taxonomy to sequence learning and improve the assessment quality [5]. Each skill in the cognitive domain of Bloom's Revised Taxonomy has a goal that guides the educator to develop tests in different areas. To our knowledge, no work explores adapting these skills' goals to programming teaching. With that in mind, this Chapter presents a proposal that sequences programming skills in the cognitive domain of Bloom's Revised Taxonomy. To validate this proposal, we applied a Survey to an expert group in CS1 to theoretically evaluate this adaptation.

We organize the remainder of this Chapter as follows: In Section 4.2, we discuss related work. In Section 4.3, we present the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching, sequencing the cognitive programming skills in this adaptation. In Section 4.4, we describe the study design. In Section 4.5, we present the results of the proposal validation. Then, in Section 4.6, we discuss the results. Finally, in Section 4.7, we conclude the Chapter with final remarks.

## 4.2 Related Works

There are several learning approaches and taxonomies in the context of programming teaching; one of these approaches is called Bloom's Revised Taxonomy [120]. Other studies used the older version of this taxonomy [31, 71, 72]. In its traditional version, Bloom's Taxonomy has the following levels: knowledge, understanding, application, analysis, synthesis, and evaluation. The Revised Taxonomy aimed to correct some problems in the original taxonomy. Bloom's taxonomy revision changed the sequence of some categories and used verbs instead of nouns [5]. Another element to highlight in the review was that creativity was considered superior (in the sequence of levels) to evaluation. In addition, it included subcategories that indicate a way to assess whether the educational objective has been achieved [5].

Recently, researchers applied the cognitive domain levels of Bloom's Taxonomy to determine the appropriate challenge level of test questions in CS1 [31]. This study aimed

to expose students to well-defined assessment tests to challenge them based on different Bloom's Taxonomy domains. Furthermore, identify difficult areas for students to redesign and reorganize class activities. To this end, the authors developed an artifact to classify questions based on the cognitive domains of Bloom's Taxonomy. They then signed up to assess the three-semester tests and write a final exam. In designing the final test, the goal was to challenge students' abilities in a predetermined ratio and combination mapped to Bloom's Taxonomy domains. After each test, the authors identified the students' problem areas and adjusted the class-related activities to address these deficiencies [31].

Another research studied the current scenario of assisted assessment for practical programming, focusing on competency-based assessment. The study presented Bloom's Taxonomy used to assess skills [71]. As a result, the study exemplifies how Bloom's Taxonomy can help to guide learning. Its skills characterization identifies which skill level the student is being assessed at, which helps improve their learning. The survey cites each Bloom Taxonomy skill level and gives examples of programming tasks educators can work on [71]. Then, the authors proposed an assessment framework based on the cognitive domain of Bloom's Taxonomy to assess the students' programming skills [72].

A previous study identified tasks that could effectively promote program understanding for novices [115]. The authors identified and classified the tasks into homogeneous categories based on Bloom's Revised Taxonomy. The researchers conducted the study to classify these tasks into each Bloom's Revised Taxonomy category based on their potential effectiveness in understanding the program targeting novices [115].

Despite advances, these studies have some limitations, namely: i) they did not consider some abilities contained in the cognitive domain of Bloom's Taxonomy, whether in its traditional or revised version [31,71,72,115]; ii) they associated tasks with cognitive domains without considering the sequence of skill subcategories that indicate a way of assessing whether the educator managed to achieve the educational objective [115]; iii) they considered the cognitive domains as unique skills without investigating the set of skills present in each level [71, 72]; iv) they considered the older version of Bloom's Taxonomy instead of its revised version [31,71,72].

These works show that programming tasks within Bloom's Taxonomy's domains improve students' understanding. However, given the limitations, we need a deeper

investigation of the contributions of Bloom's Taxonomy to programming teaching in its revised version. So, in this chapter, we adapt the cognitive domain of Bloom's Revised Taxonomy for programming teaching, sequencing the cognitive programming skills in this adaptation. Then, we applied a Survey to an expert group in CS1 to empirically evaluate this adaptation.

### 4.3 Adapting the Cognitive Domain of Bloom's Revised Taxonomy

This section presents the original version of Bloom's Revised Taxonomy and its adaptation/association with programming instruction. Together with a team of two specialists in programming instruction, we developed the initial stages of the adaptation (Fig. 4.1), adaptation synthesis, and cognitive programming skills' association to the cognitive domain of Bloom's Revised Taxonomy in programming teaching. In the following subsections, we describe these processes.

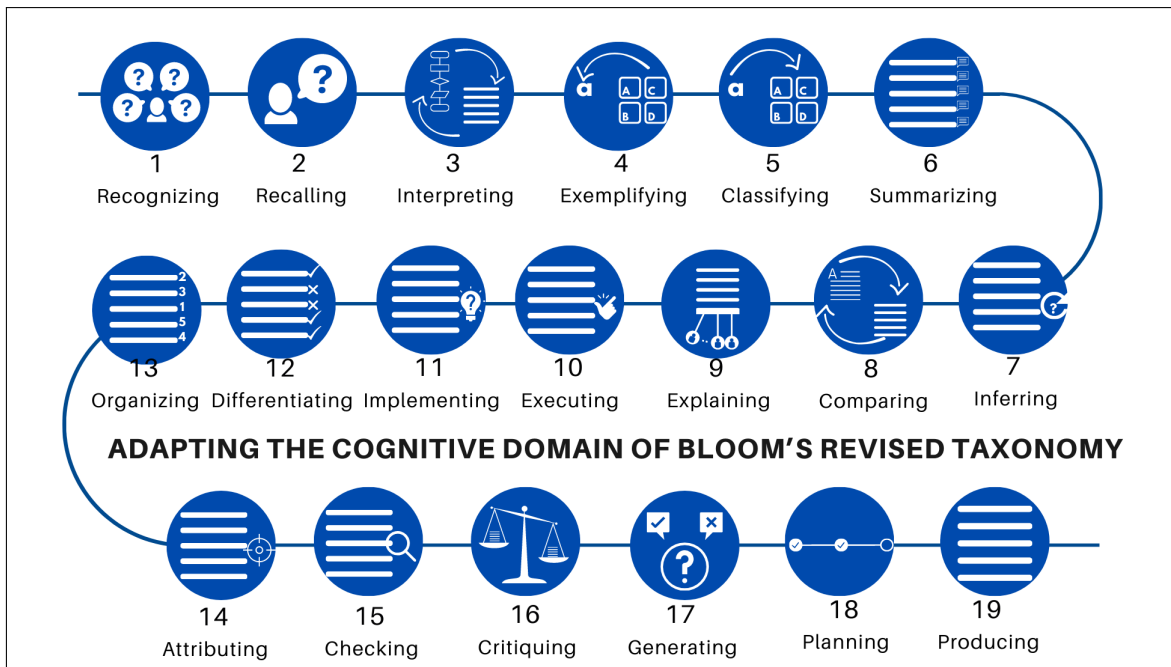


Figure 4.1: Adapting the Cognitive Domain of Bloom's Revised Taxonomy.

### 4.3.1 Remember

The first cognitive domain of Bloom's Revised Taxonomy involves remembering previously covered information and content. This domain involves remembering important information or specific facts, and this level's objective is to bring this knowledge to consciousness [5]. Table 4.1 presents the level of "remember" adaptation for programming teaching and the cognitive programming skills associated with this level.

Skill	Definition	Adaptation	Association
Recognizing	Locate knowledge in long-term memory consistent with the presented material.	Retrieve relevant knowledge from long-term memory and compare it with the information presented.	-
Recalling	Retrieve relevant knowledge from long-term memory.	Retrieve relevant knowledge from long-term memory when requested.	-

Table 4.1: Adaptation of the Level Remember and Cognitive Programming Skills Association.

### 4.3.2 Understand

The cognitive level Understand applies meaning to content. This cognitive level translates the understood content into a new form (i.e., oral, written, and diagrams) or context. There is the ability to understand the information or fact, capture its meaning, and use it in different contexts [5]. Table 4.2 presents the level of "understand" adaptation to programming teaching and the cognitive programming skills associated with this level.

### 4.3.3 Apply

The cognitive level Apply uses information, methods and content learned in new concrete situations. It includes applications of rules, methods, models, concepts, laws, and theories [5]. Table 4.3 presents the level of "apply" adaptation to programming teaching and the cognitive programming skills associated with this level.

<b>Skill</b>	<b>Definition</b>	<b>Adaptation</b>	<b>Association</b>
Interpreting	Change from one representation form to another.	Translate from one algorithm form into another.	Comprehension and Reading.
Exemplifying	Find a specific example or concept illustration, or principle.	Find an example of a particular problem.	Comprehension and Reading.
Classifying	Determine that something belongs to a category.	Determine that something belongs to a category.	Comprehension and Reading.
Summarizing	Abstract a general theme or significant point(s).	Summarize/comment on code snippets/parts.	Comprehension and Reading.
Inferring	Draw a logical conclusion from the presented information.	Draw a logical conclusion from a presented code.	Comprehension, Reading, and Tracing.
Comparing	Detect correspondences between two ideas, objects, and the like.	Detect matches between pseudocode and a code.	Comprehension and Reading.
Explaining	Construct a system's cause-and-effect model.	Construct an algorithm's cause-and-effect model.	Comprehension, Reading, and Explaining.

Table 4.2: Adaptation of the Level Understand and Cognitive Programming Skills Association.

<b>Skill</b>	<b>Definition</b>	<b>Adaptation</b>	<b>Association</b>
Executing	Apply a procedure to a familiar task.	Apply a procedure to a familiar problem.	-
Implementing	Apply a procedure to an unfamiliar task.	Apply a procedure to an unfamiliar problem.	-

Table 4.3: Adaptation of the Level Apply and Cognitive Programming Skills Association.

#### 4.3.4 Analyse

The cognitive level Analyse subdivides the content into smaller parts to understand the final structure. This level may include identifying the parts, analyzing their relationship, recognizing the organizational principles involved, and identifying parts and their interrelationships. At this point, it is necessary to understand the content and study the

object structure [5]. Table 4.4 presents the level of “analyse” adaptation to programming teaching and the cognitive programming skills associated with this level.

<b>Skill</b>	<b>Definition</b>	<b>Adaptation</b>	<b>Association</b>
Differentiating	Distinguish relevant parts from irrelevant ones or important parts from unimportant ones in the presented material.	Distinguish relevant parts from irrelevant ones presented by the algorithm.	-
Organizing	Determine how elements fit or function within a structure.	Organize code parts to achieve the program's purpose.	Modifying
Attributing	Determine a viewpoint, bias, values, or intent underlying the presented material.	Determine the code's viewpoint.	-

Table 4.4: Adaptation of the Level Analyse and Cognitive Programming Skills Association.

### 4.3.5 Evaluate

The cognitive level Evaluate involves judging material value (proposal, research, project) for a specific purpose. The criteria for this judgment can be external (relevance) or internal (organization) and can be provided or identified together [5]. Table 4.5 presents the adaptation of the level “evaluate” programming teaching and the cognitive programming skills associated with this level.

### 4.3.6 Create

The cognitive level Create aggregated parts to create a new whole. This level involves a single communication production (theme or discourse), an operation plan (research proposals), or a set of abstract relationships (scheme for classifying information). This level combines unorganized parts to form a “whole” [5]. Table 4.6 presents the adaptation of the level “create” to programming teaching and the cognitive programming skills associated with this level.

<b>Skill</b>	<b>Definition</b>	<b>Adaptation</b>	<b>Association</b>
Checking	Detect inconsistencies or fallacies within a process or product, determining whether a process or product has internal consistency.	Detect/fix flaws in an implemented program.	Debugging
Critiquing	Detect the procedure's suitability for a given problem.	Criticize procedures to solve a problem.	-

Table 4.5: Adaptation of the Level Evaluate and Cognitive Programming Skills Association.

<b>Skill</b>	<b>Definition</b>	<b>Adaptation</b>	<b>Association</b>
Generating	Come up with alternative hypotheses based on criteria.	Create alternative hypotheses based on criteria.	-
Planning	Devise a procedure for accomplishing some task.	Develop a procedure to perform a problem.	-
Producing	Create a product.	Create a program.	Writing

Table 4.6: Adaptation of the Level Create and Cognitive Programming Skills Association.

## 4.4 Design

This section presents the study design to validate the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching, sequencing the cognitive programming skills in this adaptation.

### 4.4.1 Participants

A group of five university educators who supervise CS1 evaluated the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming instructions and the association of the skills raised in the literature in this adaptation.



### 4.4.2 Techniques and Metrics

The evaluation covered the objectives of the cognitive-level skills of Bloom's Revised Taxonomy for instructional programming and cognitive programming skills' association with the adapted taxonomy. We asked the following questions:

- Indicate the degree of agreement with the objective of the skill's adaptation to programming instruction;
- Indicate the degree of agreement with the association (if any) of cognitive programming skill at the cognitive domain of Bloom's Revised Taxonomy.

We present a Likert-type scale, with the options for the judge to carry out the evaluation, with the answer for each question represented as follows:

1. Not adequate: nothing suitable, not adapted, not corresponding at all to the proposed objective for programming teaching;
2. Not very adequate: 25% adequate, adapted, corresponding very little to the proposed objective for programming teaching;
3. Moderately adequate: 50% adequate, adapted, moderately corresponding to the proposed objective for programming teaching;
4. Very adequate: 75% adequate, adapted, corresponding intensely to the proposed objective for programming teaching.
5. Completely adequate: 100% adequate, adapted, corresponding perfectly to the proposed objective for programming teaching.

### 4.4.3 Data Analysis

A judges' panel inspected the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming instruction. Then, we extracted the data contained in this study through an exploratory survey. We use the function *agree*, available in the IRR package in the R language, to evaluate the agreement's rate between the judges. We used the Intraclass Correlation Coefficient to assess inter-judge reliability.

Inter-rater agreement is how two or more raters achieve identical results under similar assessment conditions. We can use Equation 4.1 to calculate the agreement between raters.

$$agree = \frac{a}{a + b} \cdot 100 \quad (4.1)$$

Where,

$a$  = number of participants who agree;

$b$  = number of participants who disagree.

The Intraclass Correlation Coefficient assesses inter-observer or intra-observer reliability for numerical variables [67]. The Intraclass Correlation Coefficient has four questions that guide its correct use for the reliability study between evaluators:

1. Do we have the same rates for all subjects?
2. Do we have a raters' sample randomly selected from a larger population or a raters-specific sample?
3. Are we interested in the reliability of a single rater or the several raters' mean values?
4. Are we concerned about consistency or agreement?

The first two questions guide the “Model” selection, the third question guides the “Type” selection, and the last question guides the “Definition” selection [65]. Our model is **two-way mixed effects** because we use the same set of raters for all subjects and choose the sample in a specific way. The type is **average measure** because we base our measurement protocol on multiple rater averages. In the end, we adopted the definition **absolute agreement** because we wanted to assign the same score to the same subject.

We can use Equation 4.2 to calculate the Intraclass Correlation Coefficient bidirectional mixed effects, absolute agreement, and multiple rates/measurements.

$$ICC = \frac{MS_R - MS_E}{MS_R + \frac{MS_C - MS_E}{n}} \quad (4.2)$$

Where,

$MS_R$  = mean square for rows;

$MS_E$  = mean square for error;

$MS_C$  = mean square for columns;

$n$  = subject number.

Values less than 0.5 indicate low reliability; values between 0.5 and 0.75 indicate moderate reliability; values between 0.75 and 0.9 indicate appropriate reliability; and values greater than 0.90 indicate excellent reliability [65].

#### 4.4.4 Methodological Process

Figure 4.2 represents the methodological process of the adaptation stage of the cognitive domain of Bloom's Revised Taxonomy.

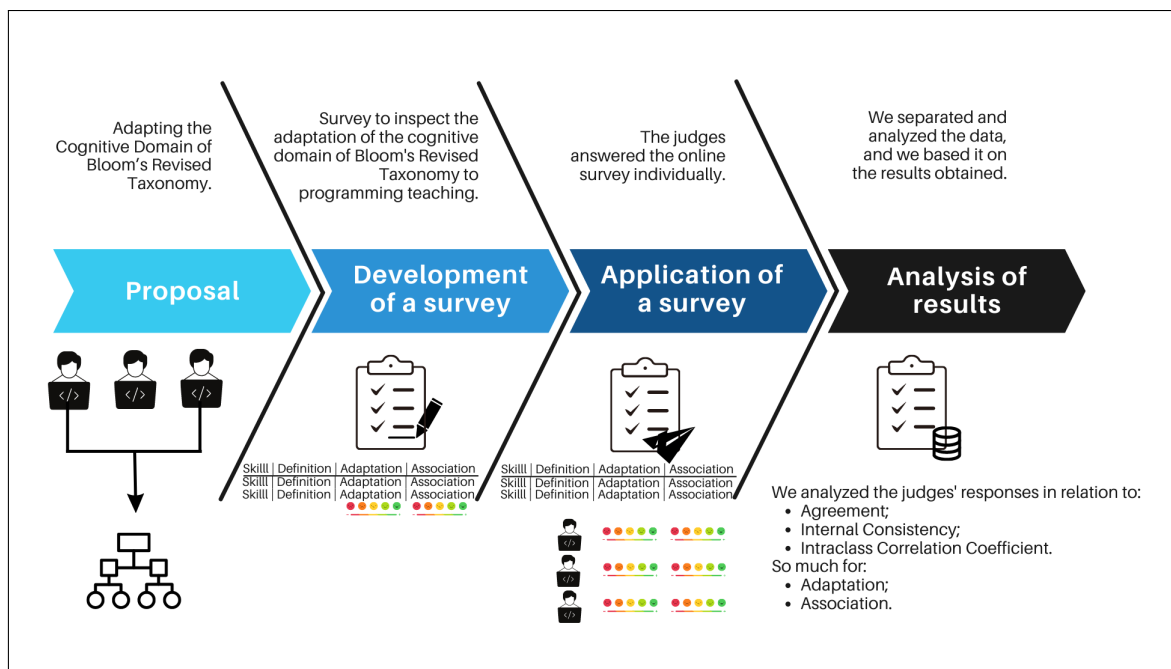


Figure 4.2: Methodological Process of Adaptation Stage of the Cognitive Domain of Bloom's Revised Taxonomy.

After completing the final version (Appendix F), we developed a survey (Appendix G) to inspect the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching.

Then, we selected a professional sample to compose the judge's panel. These judges are professionals specialized in programming teaching. We sent an invitation letter to professionals by email. The email contained the objectives and methodology of the

study, the justification for adapting the cognitive domain of Bloom's Revised Taxonomy to programming teaching, and the request to participate in the research as an evaluator.

The judges answered the online survey individually and did not have access to the opinions of the other judges participating. Each judge signed the Informed Consent Form (ICF) (Appendix H) to participate in the research. We separated and analyzed the data, and we based it on the results obtained.

#### 4.4.5 Threats to Validity

We consider some factors that can generate threats and directly influence this study's conclusions; among them:

- It is possible that this research application has been long, which may influence the specialists' answers. One way to mitigate this threat was to guarantee the judge's withdrawal at any time during the research. We applied the Ethics Committee in Research with Human Beings' guidelines of the Federal University of Campina Grande (UFCG) and the State University of Paraiba (UEPB), which approved this research (Protocols: 23933919.4.0000.5182 | 23933919.4.3001.5187). Only participants who signed the ICF participated in this study;
- The number of subjects participating in the study does not allow the inference of the results;
- The criteria cannot be clear, and the experts' answers do not reflect the response we expect for this study. Which may also disqualify the adaptation validation.

### 4.5 Results

This section presents the results of our theoretical analysis on adapting the cognitive levels of Bloom's Revised Taxonomy to programming teaching and the cognitive programming skills' association with the Adapted Taxonomy.

In Figure 4.3, we present the score distribution barplot, among the judges, within the Likert scale for each skill. We identify how much each judge agrees with adapting the definition of the skill to programming teaching.

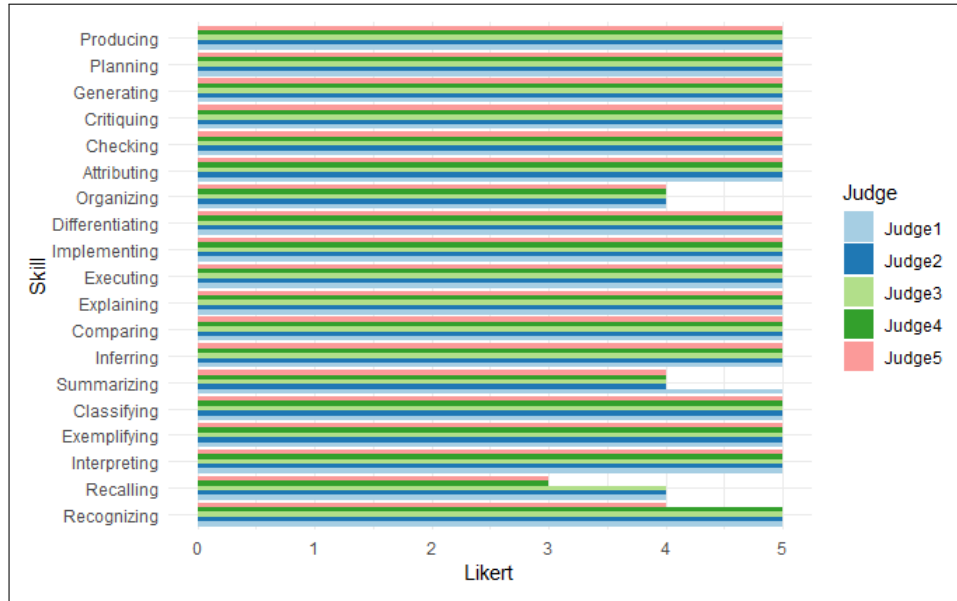


Figure 4.3: Degree of Agreement Between the Judges in the Skills Adaptation.

The judges’ responses to adapting the cognitive level skill definition of Bloom’s Revised Taxonomy to programming teaching achieved high agreement of 84.20% and high consistency, with Cronbach’s alpha reaching 0.958. In addition, the Intraclass Correlation Coefficient test 0.956 also indicated excellent reliability between the scores assigned by the evaluators [95% = 0.915 – 0.981];  $F_{(18,72)} = 24, p - value < 0.001$ , as shown in Table 4.7.

	ICC	Lower	Upper	F-test	df1	df2	P-value
Average measurements	0.956	0.915	0.981	24	18	72	< 0.001

Table 4.7: Intraclass correlation coefficient in adapting of skills.

In Figure 4.4, we present the barplot of the score distribution, among the judges, within the Likert scale for each skill. We indicate the degree of agreement with the association (if any) of cognitive programming skill in the cognitive domain of Bloom’s Revised Taxonomy.

The judges’ responses achieved a high agreement of 80.00% and a high consistency, with Cronbach’s alpha reaching 0.935. In addition, the Intraclass Correlation Coefficient test 0.940 also indicated that there was excellent reliability between the scores given by the evaluators [95% = 0.850 – 0.983];  $F_{(9,36)} = 15.483, p - value < 0.001$ , as shown in Table 4.8.

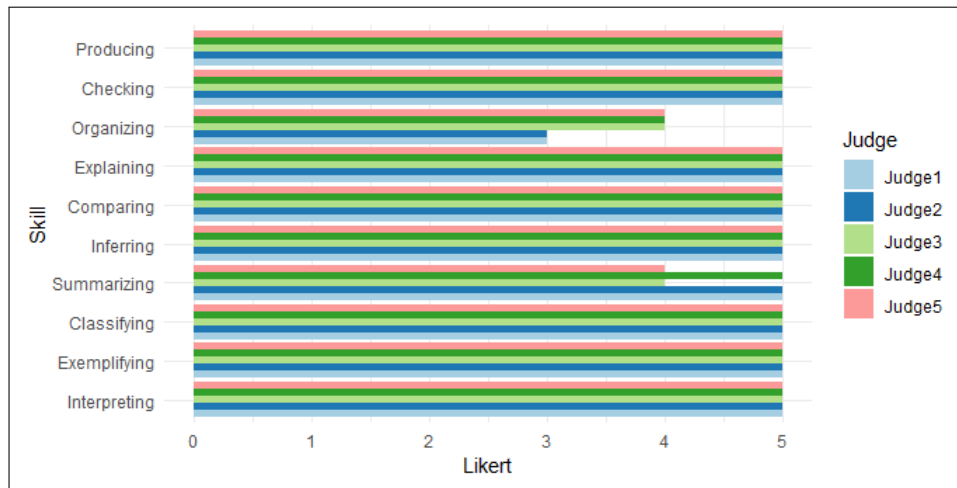


Figure 4.4: Degree of Agreement Between the Judges in the Skills Association.

	ICC	Lower	Upper	F-test	df1	df2	P-value
Average measurements	0.940	0.850	0.983	9	36	15.483	< 0.001

Table 4.8: Intraclass Correlation Coefficient in the Association of Skills.

## 4.6 Discussions

In tables 4.1 4.3 4.4 4.5 and 4.6, we can see some gaps in the association of cognitive programming skills within the taxonomy. Gaps include the need for more training in some skills in CS1.

### 4.6.1 Remember

Traditionally, the educator focuses only on mechanical learning, and teaching and assessment focus only on remembering elements or knowledge fragments, often isolated from their context. However, when educators focus on significant learning, learning is integrated with building new knowledge or solving new problems. For example, selection structure knowledge is necessary if the student wants to solve flow deviation problems. This knowledge because it allows executing one or more commands if the tested condition is “true” or executing one or more commands if it is “false.” The cognitive skills in the Remember level include:

- **Recognizing** involves retrieving relevant knowledge from long-term memory to compare it with the information presented. Recognize the student's demand for long-term memory information identical to the information presented (as represented in the working memory). When presented with the new information, the student determines whether this information corresponds to previously learned knowledge;
- **Recalling** involves retrieving relevant knowledge from long-term memory when asked to do so. The prompt is often a question. When remembering, the student searches the long-term memory for the information and processes part of the working memory information.

Let us assume that a student has learned selection structures. A memory test may involve asking the student to match each selection structure with its respective functioning in a second list (i.e., recognize). Alternatively, the student writes the corresponding functioning beside each selection structure presented in the list (i.e., recalling).

## 4.6.2 Understand

Several research studies have confused skill with "cognitive level." Comprehension is not a skill, but a synonym related to the "understand" cognitive level [28, 30, 32, 35–38, 73, 86, 92, 93, 117, 139], as well as "reading," which involves the "understand" cognitive level [4, 17, 20, 51, 84, 90, 102, 108, 137]. These works need to specify the level of cognitive ability.

Students understand when they build connections between "new" knowledge and prior knowledge. Existing cognitive structures and schemata integrate the new knowledge. Since concepts are the building blocks for these schemes and structures, conceptual knowledge provides a basis for understanding.

The cognitive skills of the Understand level include:

- **Interpreting** is the student's ability to translate an algorithm form to another form (for example, translate an algorithm to a flowchart);
- **Exemplifying** involves identifying the defined characteristics of the concept or a general principle (for example, the rules for variable names) and using these

characteristics to select or construct a specific instance (for example, being able to select which of the variable name declarations is incorrect);

- **Classifying** occurs when a student recognizes that something (e.g., a particular instance or example) belongs to a specific category (e.g., concept or principle). Classifying involves detecting relevant features or patterns that “fit” the specific instance and the concept or principle. Classifying is a complementary process to exemplifying. Whereas exemplifying begins with a general concept or principle and requires the student to find a specific instance or example, classifying begins with a specific instance or example and requires the student to find a general concept or principle;
- **Summarizing** occurs when a student suggests a single statement representing presented information or abstracts a general theme (for example, learning to summarize the subroutine’s purposes in a program. An assessment item presents a program and asks a student to write a sentence describing the subgoal that each program section accomplishes within the overall program);
- **Inferring** is the student’s ability to draw a logical conclusion from a presented algorithm or code (for example, finding out the code output) [41, 42, 51, 52, 54, 55, 93, 116];
- **Comparing** is the student’s ability to detect similarities and differences between two or more ideas and determine how well-known an event is with a less familiar one (for example, detect matches between pseudocode and a code);
- **Explaining** is the student’s ability to build and use a system’s cause-and-effect model. Several tasks can assess a student’s ability to explain, including reasoning, problem-solving, redesign, and prediction [11, 20, 28, 30, 83, 86, 89, 116, 119, 130, 139]. In reasoning tasks, a student offers a reason for a particular event. In problem-solving, a student diagnoses what could have gone wrong in a flawed system. In the redesign, a student changes the system to achieve some goal. In the prediction, the student may assess that changing one part of the system will affect another.



### 4.6.3 Apply

Apply consists of two cognitive processes: Executing and Implementing. When executing, the educator presents the student with a familiar task and a procedure to solve it. The student can provide an answer or, where appropriate, select from the possible answers set. In the implementation, the student receives an unfamiliar problem that must be solved. Therefore, most evaluation formats start with specifying the problem. Students determine the procedure required to solve the problem, use the selected procedure (making the necessary changes), or generally both.

- **Executing** is the student's ability to perform a familiar problem procedure (e.g., requesting the student to perform a procedure on the familiar problem);
- **Implementation** is the student's ability to use an unfamiliar problem procedure (e.g., requesting the student to perform a procedure on the unfamiliar problem).

### 4.6.4 Analyse

Although learning to analyze can be seen as an end, educationally, it is more defensible to consider analysis as an understanding extension or as a prelude to evaluating or creating. Many universities aim to improve students' code analysis skills in teaching programs. In introductory programming courses, educators provide code examples for students to "learn to analyze" as an important goal. For example, they may wish to develop their students' abilities: i) divide a programming task into parts; ii) organize the parts to achieve a general objective; iii) identify critical or unimportant components for development.

Understand, Analyse and Evaluate are interrelated and often used iteratively in performing cognitive tasks. At the same time, however, it is essential to maintain them as separate process categories. A person who understands a problem may need to analyze it better. Similarly, someone skillful in analyzing a problem may need to improve.

This process category includes cognitive skills:

- **Differentiating** is the student's ability to distinguish relevant parts of the presented code or algorithm (for example, distinguish irrelevant parts of code). Differentiating occurs when a student discriminates relevant/essential from irrelevant/unimportant

information. This skill is different from the cognitive skills associated with understanding because it involves structural organization and, in particular, determines how the parts fit into the overall structure or the whole;

- **Organizing** is the student's ability to order the parts to achieve the program goal (such as organizing code or algorithm snippets so that the logic is correct). In organizing, a student builds systematic and coherent connections among presented information. Organizing usually occurs in conjunction with differentiating. The student first identifies the relevant or essential elements and then determines the overall structure in which the elements fit. Organizing can also occur in conjunction with attributing, in which the focus is on determining the author's intention or viewpoint;
- **Attributing** is the student's ability to determine a given code's viewpoint (for example, the student constructs or selects a description of a given code). The attributing involves a deconstruction process in which the student determines the material objectives presented. Unlike interpreting, in which the student seeks to understand the meaning of the material presented, attribution involves an extension beyond the necessary understanding to infer the intended or viewpoint underlying the material presented.

### 4.6.5 Evaluate

The evaluate category includes cognitive verification processes and critical judgments based on external criteria. These criteria can be determined by the student or by others. For example: is this code or algorithm sufficiently compelling?

This level involves the following skills:

- **Checking** is the student's ability to detect the effectiveness of a program as the student implements it (for example, finding/correcting a logical error in a given piece of code) [2, 9, 16, 21, 23, 29, 35–38, 41, 42, 51, 69, 78, 83, 84, 87, 95, 124, 133]. CS considers checking as testing or debugging [21, 29, 133].
- **Critiquing** is the student's ability to criticize procedures for solving a problem based on coding standards (for example, judging which of two code snippets or algorithms is the best way to solve a given problem) [16, 83, 86, 116, 139].

When combined with planning (a cognitive skill in the category Create) and implementing (a cognitive skill in the category Apply), the checking determines how well the plan works. When critiquing, the student observes the positive and negative characteristics of a code or algorithm and makes a judgment based, at least partially, on those characteristics. Critiquing is at the heart of what has been called critical thinking. A critiquing example is to judge the particular solution's merits to the problem in terms of its efficiency [68].

### 4.6.6 Create

Students create new code or algorithms at this level by mentally rearranging some elements or parts in a pattern or structure that was not present. To carry out this process, students draw on previous learning experiences. Although it requires creative thinking on the student's part, it does not imply free creative expression.

Although this level includes goals that require unique production, it also refers to goals that require production that all students can and will do. When fulfilling these objectives, many students synthesize information or materials to form a new whole, such as solving a problem.

Although the Understand, Apply, and Analyze process levels may involve detecting relationships between the elements presented, Create is different because it involves building an original product. Unlike the Create level, the other levels involve working with a particular set of elements that are part of a given whole and part of a larger structure the student is trying to understand. On the other hand, the student must draw on elements from many sources and put them together in a new structure or pattern related to his previous knowledge. Creating a new program or algorithm can be observed, that is, more than the student's initial material. A task that requires Create is likely to require aspects of each of the previous cognitive processes' categories to some extent.

Thus, the creative process begins with a divergent phase in which the student thinks about various possible solutions to understand the task (generate). The student conceives a solution method and transforms it into an action plan (planning). Finally, the student executes the plan while building the solution (producing). It is not surprising that Creating is associated with three cognitive processes:

- **Generating** is the student's ability to create alternative hypotheses based on criteria (for example, hypothesizing that an algorithm's new combination will solve the problem).
- **Planning** is the student's ability to develop a procedure to perform a task (for example, planning an algorithm, process, or an alternative strategy for a problem);
- **Producing** is the student's ability to create a program (for example, build a program using invented algorithms) [16, 17, 19, 20, 32, 33, 40–42, 47, 49, 52–54, 57, 60–62, 69, 81, 83, 84, 86, 89, 90, 93, 100, 102, 116, 117, 135–140].

## 4.7 Final Considerations

Programming is a complex activity involving several concepts and structural regulations addressed by different cognitive levels operating in different stages, making it a challenge for students to learn and for educators to teach. Current introductory instructions fail to sequence the many skills involved in programming [137].

With that in mind, (**RQ4:**) we start from the assumption that sequencing the cognitive skills involved in programming learning determines the appropriate challenge level in an assessment instrument. We adapted the cognitive domain of Bloom's Revised Taxonomy to programming teaching, sequencing the cognitive programming skills in this adaptation. Then, we validate the proposal through a survey applied to a group of judges in CS1.

As a result, we conclude that the judges' responses to the adaption/association of cognitive level skill definition of Bloom's Revised Taxonomy to programming teaching achieved high agreement and high consistency. The Intraclass Correlation Coefficient test also indicated excellent reliability between the scores assigned by the judges.

# Chapter 5

## An Instrument to Measure and Foster Cognitive Programming Skills

In this Chapter, we present the instrument built on adapting the cognitive domain of Bloom's Revised Taxonomy to programming teaching. Then, we analyzed the item's semantics and content present in this instrument through a survey applied to a group of experts in CS1.

### 5.1 Initial Considerations

In Chapter 3, we present a wide variety of resources with recommendations for their application to fostering and measurement in CS1. However, not all recommendations are relevant to be applied as additional resources for educators to sequence the cognitive load involved in programming learning. Thus, the vast majority do not have specific characteristics that meet the needs of these individuals.

Based on this knowledge gap of these specific characteristics, we present the development of an instrument. The instrument contains a bank of 750 items based on the indicators in the Cognitive Domain of Bloom's Revised Taxonomy adapted to programming teaching (Subsection 4.3). The instrument aims at developing the student's ability to understand and write code. However, how do we develop a reliable instrument to measure such skills? We use Measurement Theories (Section 2.3) to answer this question. The initial steps in developing an instrument include i) Instrument definition; ii) Item Bank Preparation.

Therefore, this Chapter aims to:

- **SO5.** To develop an item bank with content and semantics analysis that include indicators of cognitive programming skills;

In this chapter, we answer the following research questions:

- **RQ5.** Does the instrument have items with appropriate semantic analysis?
- **RQ6.** Does the instrument have items with appropriate content analysis?

In this initial study, we applied a survey to a group of 38 judges of advanced programmers, educators, and experts in CS1. The judges assessed whether the items were well understood (semantic analysis) and adequate to measure the expected ability (content analysis). The IRT considers an item with appropriate semantics and content if the judges' agreement is greater than 80% [8, 14, 101].

We organized the remainder of this Chapter as follows. In Section 5.2, we describe the instrument's development. In Section 5.3, we describe the study design. In section 5.4, we present the results. In Section 5.5, we discuss the results. In section 5.6, we discuss the related works. Finally, in Section 5.7, we conclude the Chapter with final considerations.

## 5.2 Instrument Development

This section presents the instrument definition and the building process of the items' bank.

### 5.2.1 Instrument Definition

The first step in building an instrument consists of formalizing the purpose and defining the skill and dimensions necessary to respond.

- **Instrument's purpose:** indicates its general purpose, which can be, for example, to estimate a grade or classify individuals [8, 103]. In our case, *the instrument's purpose is to measure the student's performance in CS1*;
- **Skill:** is the characteristic we aim to evaluate. It is what the instrument aims to measure [8]. *We want to measure the cognitive skills that influence students' performance in CS1*;

- **Instrument’s dimension:** indicates the number of skills evaluated [8]. In our case, *the instrument has 19 skills*, as shown in Table 5.1.

Level	skill	Description
Remember	Recognizing	Retrieving relevant knowledge from long-term memory and comparing it with the information presented.
	Recalling	Retrieving relevant knowledge from long-term memory when requested.
Understand	Interpreting	Translate from one algorithm form into another.
	Exemplifying	Finding an example of a particular problem.
	Classifying	Determining that something belongs to a category.
	Summarizing	Summarize/comment on code snippets/parts.
	Inferring	Draw a logical conclusion from a presented code.
	Comparing	Detecting matches between pseudocode and a code.
Apply	Executing	Apply a procedure to a familiar problem.
	Implementing	Apply a procedure to an unfamiliar problem.
Analyse	Differentiating	Distinguish relevant parts from irrelevant ones presented by the algorithm.
	Organizing	Organizing code parts to achieve the purpose of the program.
	Attributing	Determining the code’s viewpoint.
Evaluate	Checking	Detecting/fixing flaws in an implemented program.
	Critiquing	Criticizing procedures to solve a problem.
Create	Generating	Creating alternative hypotheses based on criteria.
	Producing	Developing a procedure to perform a problem.

Table 5.1: Skills Investigated and Proposed in the Instrument.

### 5.2.2 Item Bank Building

Combined with a multidisciplinary team of experts composed of two experts in programming teaching, a statistician and a psychometric psychologist, we developed an instrument that measures cognitive programming skills. The instrument addresses the following contents: i) data input and output; ii) conditional statement; iii) iteration statement, and vi) function. The instrument fosters the 19 skills contained in the cognitive domain of Bloom’s Revised Taxonomy for each content.

Developing items that foster the “Differentiating” skill to the “data input and output” content was impossible. Such impossibility is because data input and output are the first content taught in a programming language. This content focuses on structured problem-solving without flow deviation so that the solution does not present irrelevant parts.

We developed ten items for each skill x content combination, totaling 30 items per differentiating skill and 40 for the other skills. Thus, the instrument comprises 750 items, as specified in Table 5.2.

<b>Skill</b>	<b>Data input and output</b>	<b>Conditional statement</b>	<b>Iteration statement</b>	<b>Function</b>	<b>Total</b>
Recognizing	10	10	10	10	40
Recalling	10	10	10	10	40
Interpreting	10	10	10	10	40
Exemplifying	10	10	10	10	40
Classifying	10	10	10	10	40
Summarizing	10	10	10	10	40
Inferring	10	10	10	10	40
Comparing	10	10	10	10	40
Explaining	10	10	10	10	40
Executing	10	10	10	10	40
Implementing	10	10	10	10	40
Differentiating	-	10	10	10	30
Organizing	10	10	10	10	40
Attributing	10	10	10	10	40
Checking	10	10	10	10	40
Critiquing	10	10	10	10	40
Generating	10	10	10	10	40
Planning	10	10	10	10	40
Producing	10	10	10	10	40
<b>Total</b>					<b>750</b>

Table 5.2: Total Items Contained in the Instrument.

When developing the measurement instrument, we ordered the constructions to depend only on the knowledge of the constructions learned in CS1. We chose Python programming because it is a standard introductory language that appeals to many students (including non-graduates). To write programs, python does not require more advanced programming constructs, such as methods or classes [41,42]. Below, we present example items present in the measurement instrument in CS1 (Conditional statement) within the Adaptive Cognitive Domain of Bloom's Revised Taxonomy to measure and foster cognitive programming skills (Section 4.4). Other items in their Portuguese version are available at: <https://screeningprogramming.com/>.

### **Remember**

Bloom's Taxonomy first level reinforces curriculum mastery skills: Recognize and Remember. Such skills relate to recognition or recall in conditions as the student learned



from the material.

### *Recognizing*

Figure 5.1 depicts a case of an item that assesses the student's ability to retrieve relevant knowledge from long-term memory and compare it with the information presented. In this case, we retrieve knowledge about selection commands.

<b>LEVEL:</b> REMEMBER	<b>SKILL:</b> RECOGNIZING
<b>ID.:</b> #REMREC01CS	<b>CONTENT:</b> CONDITIONAL STATEMENTS
Based on conditional statements' understanding, check among the alternatives which of the conditional statements' types allows you to create an expression with two paths: one when the condition result is True and the other when the result is False.	
a) If statement;	
b) If-else statement;	
c) Nested statement;	
d) Case statement.	

Figure 5.1: Example of the Item Recognizing Skill.

### *Recalling*

Figure 5.2 is an example of an item that assesses the student's ability to retrieve relevant knowledge from long-term memory when asked to do so.

### **Understand**

If instruction fosters retention, the focus is on skills that emphasize the cognitive level. However, if instruction promotes transfer, the focus shifts to the other seven cognitive skills at the Comprehend level. This level can be exercised intensely in CS1, as many programming activities involve, in some way, the Understand level.

<b>LEVEL:</b> REMEMBER	<b>SKILL:</b> RECALLING
<b>ID.:</b> #REMREL01CS	<b>CONTENT:</b> <b>CONDITIONAL STATEMENTS</b>

Based on conditional statements' understanding, the **if statement** allows us to make a decision. If the condition is true, the program executes the instructions' block(s) following the colon. Otherwise, the program executes the first instruction after the indented command.

- a) Correct;
- b) Incorrect.

Figure 5.2: Example of the Item Recalling Skill.

It becomes easier for students to succeed in problem-solving when they understand that they can construct meaning from instructional messages, including syntax and code examples with specific problems.

Examples to explain what the code does or even infer the code output (execute the code mentally or on paper) extend the cognitive levels of understanding a particular concept in programming. As a result, students understand the subject better and connect the “new” knowledge and their previous knowledge. The knowledge received is integrated into existing cognitive schemes and structures. Since concepts are the building blocks for these schemas and structures, conceptual knowledge provides a foundation for understanding. Skills such as interpreting, exemplifying, classifying, summarizing, inferring, comparing, and explaining are part of the second cognitive level of Bloom’s Taxonomy.

### *Interpreting*

Once the student has mastered the definitions of a given concept (in this case, selection commands), we move on to the Understand level. First, we present an example of a problem that can be solved for each selection command type so that the student can assimilate the definitions with examples in practice. We can present flowcharts with the student’s intention

to interpret through a visual image (i.e., interpreting). This skill is called interpretation and occurs when students can convert information from one representational form to another. In Figure 5.3, we present an example of an item that assesses this ability.

**LEVEL: UNDERSTAND** **SKILL: INTERPRETING**

**ID: #UNDINT01CS |** **CONTENT: CONDITIONAL STATEMENTS**

Interpret the flowchart below, and check the alternative where it translates the code pass to a simple select statement for Python.

```

graph TD
    begin([begin]) --> n[/n/]
    n --> mod{n mod 2 = 0?}
    mod -- Yes --> even([The number is even])
    mod -- No --> end([end])
  
```

a) `if (n%2!=1):`  
`print("The number is odd")`

b) `if (n%2==1):`  
`print("The number is even")`

c) `if (n%2!=0):`  
`print("end")`

d) `if (n%2==0):`  
`print("The number is even")`

Figure 5.3: Example of the Item Interpreting Skill.

We can assess the student's ability to exemplify and classify selection commands. For example, we can select among the options which of the code snippets is a composed selection command (i.e., exemplify). Alternatively, sort a code snippet among the types of selection commands (i.e., classifying). Overall, the ability to classify is complementary to exemplification. At the same time, exemplification starts with a general concept or principle, requiring the student to find a specific instance or example. The classification starts with a specific instance or example and requires the student to find a general concept or principle.

### *Exemplifying*

Figure 5.4 depicts an example of an item that assesses the student's ability to exemplify. That is, to identify the defining characteristics of the general concept or principle and to use these characteristics to select or build a specific instance.

<b>LEVEL: UNDERSTAND</b>	<b>SKILL: EXEMPLIFYING</b>
<b>ID: #UNDEXE01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

In Python, which of the code snippets is an if statement example.

a)

```
if (average>=7):  
    print("Approved")  
else:  
    print("Failed")
```

b)

```
if (average>=7):  
    print("Approved")
```

c)

```
if (average>=7):  
    print("Approved")  
elif (average<7):  
    print("Failed")
```

d)

```
if (average>=7):  
    print("Approved")  
elif (average>=4):  
    print("Final")  
else:  
    print("Failed")
```

Figure 5.4: Example of the Item Exemplifying Skill.

### *Classifying*

Figure 5.5 depicts an example of an item that assesses the student's ability to classify, that is, to recognize that something belongs to a specific category.

### *Summarizing*

In this process, the student can foster the ability to summarize the code and understand the various subroutines' purpose in a program. In Figure 5.6, we present an assessment item with a code piece. The item asks the student to write a sentence describing the subgoal that each section of the program accomplishes within the general program.

### *Inferring*

In addition to explaining the code, we can exercise the student's ability to infer and compare on this cognitive level of Bloom's Taxonomy. We can provide items that assess the student's ability to draw logical conclusions from a given code and discover its output (as demonstrated in Fig. 5.7).

<b>LEVEL:</b> UNDERSTAND	<b>SKILL:</b> CLASSIFYING
<b>ID.:</b> #UNDCLA01CS	<b>CONTENT:</b> CONDITIONAL STATEMENTS

Sort the following code snippet between conditional statements types.

```
if (average>=7):
    print("Approved")
else:
    print("Failed")
```

a) If statement;  
b) If-else statement;  
c) Nested statement;  
d) Case statement.

Figure 5.5: Example of the Item Classifying Skill.

<b>LEVEL:</b> UNDERSTAND	<b>SKILL:</b> SUMMARIZING
<b>ID.:</b> #UNDSUM01CS	<b>CONTENT:</b> CONDITIONAL STATEMENTS

Explains the subgoal that each program section accomplishes in the overall program.

```
1. num = int(input("Enter a number:"))
2. if (num>10) and (num<50):
3.   print("Valid")
4. else:
5.   print("Not valid")
```

I. On line 1, a number is requested and stored in the variable num;  
II. On line 2, the if statement checks whether the value stored in num is in the range of 10 and 50;  
III. On line 3, if the condition is true, it prints the message "Valid";  
IV. On line 4, if the sentence is false, the else command allows executing the following line;  
V. On line 5, the message "Not valid" will be displayed since the sentence is false.

a) T,F, F, T, F;    b) T, T, T, T, T;    c) F, T, F, T, T;    d) T, T, F, F, T.

Figure 5.6: Example of the Item Summarizing Skill.

<b>LEVEL: UNDERSTAND</b>	<b>SKILL: INFERRING</b>
<b>ID.: #UNDINF01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

Infer the code's output, getting the following 5,8,9 inputs.

```
num1 = int(input("Enter a number:"))
num2 = int(input("Enter a number:"))
num3 = int(input("Enter a number:"))
if(num1>num2 and num1>num3):
    print(num1)
elif(num2>num3):
    print(num2)
else:
    print(num3)
```

a) 8;  
b) 9;  
c) 5;  
d) 589.

Figure 5.7: Example of the Item Inferring Skill.

### *Comparing*

Figure 5.8 depicts an item example that assesses a student's ability to detect similarities and differences between two or more ideas. Furthermore, this item determines how well-known an event relates to a less familiar one.

### *Explaining*

Finally, we can assess the student's ability to explain, that is, to build and use a system's cause-and-effect model. An example of several items that can assess this skill in programming involves reasoning, problem-solving, redesign, and prediction. Figure 5.9 depicts an example of an item that works on the student's reasoning, which asks a student to provide a reason for a specific event.

<b>LEVEL: UNDERSTAND</b>	<b>SKILL: COMPARING</b>	
<b>ID.: #UNDCOM01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>	
The pseudocode below expresses an algorithm that aims to print a number' half if it is even.		
<pre> <b>Algorithm</b> even <b>Var</b>   n: <b>integer</b> <b>Begin</b>   <b>Write</b>("Enter a number:")   <b>Read</b> (n)   <b>If</b> (n mod 2 = 0) <b>then</b>     <b>Write</b>("Half the number:", n/2)   <b>End_if</b> <b>End.</b> </pre>	<pre> a) num = int(input("Enter a number:")) if(num/2==0):     print("Half the number: ", num//2)  b) num = int(input("Enter a number:")) if(num%2==0):     num = num/2 print("Half the number: ", num) </pre>	<pre> c) num = int(input("Enter a number: ")) if(num%2==0):     print("Half the number: ", num/2)  d) num = int(input("Enter a number: ")) if(num/2==0):     num = num%2 print("Half the number: ", num) </pre>

Figure 5.8: Example of the Item Comparing Skill.

<b>LEVEL: UNDERSTAND</b>	<b>SKILL: EXPLAINING</b>	
<b>ID.: #UNDEXP01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>	
Complete the explanation below why the following program, given 11 as input, will output the message "number is odd"?		
<pre> num = int(input("Enter a number:")) if(num%2==0):     print("The number is even") else:     print("The number is odd") </pre>		
The number obtained in the input is 11. When performing the module of 11 by 2, it will result in ____; the expression test is _____, so the message "The number is odd" will be printed.		
Completing the explanation, which statement is correct:		
<ul style="list-style-type: none"> <li>a) 1; false;</li> <li>b) 0; true;</li> <li>c) 1; true;</li> <li>d) 0; false.</li> </ul>		

Figure 5.9: Example of the Item Explaining Skill.

## Apply

Once the student can understand the content, we must encourage him/her to apply the knowledge. This third cognitive level of Bloom's Taxonomy consists of two cognitive skills: Execution and Implementation. In addition, it involves using procedures to perform exercises or to solve problems.

### *Executing*

In Executing, the student solves a familiar task using a well-known procedure. For example, we can present a problem and the technique/model that solves this problem. We can ask students to provide the answer or, when appropriate, select the correct answer from a set of possible answers. Figure 5.10 depicts an example of an item that assesses a student's ability to select and use a procedure for a familiar problem.

<b>LEVEL: APPLY</b>		<b>SKILL: EXECUTING</b>	
<b>ID: #APPEXE01CS  </b>		<b>CONTENT: CONDITIONAL STATEMENTS</b>	
Given the variables x, y and z, select the code snippet that prints the largest value using maximum model (if/elif/else)			
a)	b)	c)	d)
if(x<y and x<z):	if(x>y and x>z):	if(x>y and x<z):	if(x<y and x>z):
print(x)	print(x)	print(x)	print(x)
elif(z<y):	elif(y>z):	elif(z>y):	elif(z>y):
print(y)	print(y)	print(y)	print(y)
else:	else:	else:	else:
print(z)	print(z)	print(z)	print(z)

Figure 5.10: Example of the Item Executing Skill.

### *Implementing*

In the implementation, the student analyzes an unfamiliar problem to find a solution. Therefore, most assessment formats start with problem specifications. We can ask students to determine the procedure needed to solve the problem, use the selected procedure, or both.



Figure 5.11 depicts an example of an item that assesses the student's ability to receive an unfamiliar problem to solve it.

<b>LEVEL: APPLY</b>		<b>SKILL: IMPLEMENTING</b>	
<b>ID.: #APPIMP01CS  </b>		<b>CONTENT: CONDITIONAL STATEMENTS</b>	
Given the variables x, y and z, select the code snippet that prints the intermediate value.			
a)	b)	c)	d)
if(x<y and x<z):	if(x>y and x>z):	if(x>y and x<z):	if(x>y and x<z):
print(x)	print(x)	print(x)	print(x)
elif(z<y):	elif(z>y):	elif(z>y):	elif(y<z):
print(y)	print(y)	print(y)	print(y)
else:	else:	else:	else:
print(z)	print(z)	print(z)	print(z)

Figure 5.11: Example of the Item Implementing Skill.

## Analyse

The fourth cognitive level of Bloom's Taxonomy refers to analyzing, that is, segmenting the material and its constituent parts and determining how the parts correlate with the general structure. Although learning to analyze is seen as an end, it is more educationally defensible to regard analysis as an extension of understanding or a prelude to evaluating or creating.

The process includes three cognitive skills at this level: differentiation, organization, and attribution. In programming, the analysis includes learning to determine the relevant or essential parts of a code (differentiation), how the parts of the code are organized (organization), and the code's purpose (attribution).

### *Differentiating*

Differentiation differs from the cognitive processes associated with Understanding because it involves structural organization and determining how the parts fit into the general or whole structure. Mainly, Differentiating differs from Comparing in the use of context.

Figure 5.12 depicts an item that assesses the student's ability to distinguish relevant parts from irrelevant ones of the oral algorithm of the presented code.

<b>LEVEL: ANALYSE</b>	<b>SKILL: DIFFERENTIATING</b>
<b>ID.: #ANADIF01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

See the code snippet below. The row-by-row analysis distinguishes relevant and irrelevant parts based on the suggested input. Strike out the lines that the code will not execute.

1. friend = "João"
2. time = 30
3. if (time <= 40 and friend== "Suse"):
4.    print("Bring 2 jacket")
5. elif(time<=35):
6.    print("Bring 1 jacket.")
7. else:
8.    print("You don't need a jacket!")

Based on the code shown:

- a) Lines 4, 7 and 8 will not be executed;
- b) Lines 4, 6, 7 and 8 will not be executed;
- c) Lines 6 and 8 will not be executed;
- d) Lines 5, 6, 7 and 8 will not be executed.

Figure 5.12: Example of the Item Differentiating Skill.

### *Organizing*

The student builds systematic and coherent connections between the information presented by Organizing. An organization usually relates to differentiation. The student first identifies the relevant or essential elements and then determines the general structure in which the elements fit. Figure 5.13 depicts an example of an item that assesses the student's ability to order the parts to achieve the program's objective.

### *Attributing*

Attributing involves a deconstruction process in which a student determines the intentions of the presented code. In contrast to interpretation, in which the student seeks to understand the meaning of the code, attribution involves extending beyond basic understanding to infer the presented code's underlying intent or point of view. Figure 5.14 depicts an example of an item that assesses the student's ability to determine the point of view of a given code.

<b>LEVEL: ANALYSE</b>	<b>SKILL: ORGANIZING</b>
<b>ID: #ANAORG01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

Note the following snippets; order them to print the largest value among three variables.

1. print(z)
2. elif(y>z):
3. print(x)
4. else:
5. if(x>y and x>z):
6. print(y)

- a) The correct sequence is 5, 6, 2, 1, 4 and 3;
- b) The correct sequence is 5, 3, 2, 1, 4 and 6;
- c) The correct sequence is 5, 3, 2, 6, 4 and 1;
- d) The correct sequence is 5, 3, 4, 6, 2 and 1.

Figure 5.13: Example of the Item Organizing Skill.

<b>LEVEL: ANALYSE</b>	<b>SKILL: ATTRIBUTING</b>
<b>ID: #ANAATT01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

What is the code purpose below?

```
a = int(input("Enter a number:"))
b = int(input("Enter a number:"))
c = int(input("Enter a number:"))
if(a<b e a<c):
    print(a)
elif(b<c):
    print(b)
else:
    print(c)
```

Mark the correct alternative:

- a) The code aims to find the smallest value among three variables;
- b) The code aims to find the largest value among three variables;
- c) The code aims to find the average value between three variables;
- d) The code will always print the first input value among three variables.

Figure 5.14: Example of the Item Attributing Skill.

## Evaluate

Most cognitive processes require some form of judgment. What most differentiates *Evaluate* as defined here from other judgments made by students is the use of performance standards with clearly defined criteria. The main objective of this cognitive level of Bloom's Taxonomy is to improve the student's critical sense, verifying and criticizing the presented code.

### *Checking*

Checking involves testing internal inconsistencies or fallacies in an operation or product. Checking occurs when a student tests whether or not a conclusion follows from its premises, whether the data supports or disproves a hypothesis, or whether the material presented contains parts that contradict each other. Figure 5.15 is an example of an item that assesses the student's ability to detect/correct flaws in an implemented program.

<b>LEVEL: EVALUATE</b>	<b>SKILL: CHECKING</b>
<b>ID: #EVACHE01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

From the code snippet below, determine if it prints the minimum value for variables a, b, and c, each storing a number, here 2, 3, and 1, respectively. Otherwise, identify the error.

```
a = int(input("Enter a number:"))
b = int(input("Enter a number:"))
c = int(input("Enter a number:"))
if(a<b and a<c):
    print(c)
elif(b<c):
    print(b)
else:
    print(a)
```

a) The code will not print the minimum value but the maximum value;  
b) The code has an error; the value being printed does not match the value being checked in the conditional;  
c) The code does not contain errors; the value to be printed will be 1;  
d) The code does not contain errors; the value to be printed will be 2.

Figure 5.15: Example of the Item Checking Skill.

### Critiquing

Critiquing involves judging a product or operation against externally imposed criteria and standards. When criticizing, the student looks at a product's positive and negative characteristics and makes a judgment based, at least partly, on those characteristics. Figure 5.16 depicts an example of an item that assesses the student's ability to criticize procedures for solving a problem.

<b>LEVEL:</b> EVALUATE	<b>SKILL:</b> CRITIQUING
<b>ID:</b> #EVACRI02CS	<b>CONTEÚDO:</b> COMANDOS DE SELEÇÃO

Consider the following Python code snippet. Check the alternative in which this command can be written more simply as:

```

if p==True:
    print(True)
elif q==True:
    print(True)
else:
    print(False)

```

a)	b)
if p==False and q==False: print(True)	if p==False and q==False: print(True)
else: print(False)	else: print(False)
c)	d)
if p==False or q==False: print(True)	if p==False or q==False: print(True)
else: print(False)	else: print(False)

Figure 5.16: Example of the Item Critiquing Skill.

### Create

Traditional teaching in programming defines selection commands, presents an example using the syntax, and then asks the student to create programs that solve a given problem. The cognitive load involved in these processes is high, making it necessary for the foster learning to program to occur more naturally.

This last level of Bloom's Taxonomy involves the student's ability to create a new product by mentally Reorganizing some elements or parts into a pattern or structure that was not present. The student's previous learning experiences coordinate the cognitive processes involved in this phase. Although this level requires creative thinking on the part of the

student, this is not a free creative expression, not constrained by the demands of the task or learning situation.

Thus, the creative process can start with a divergent phase through various possible solutions. The student tries to understand the task (generate). A convergent phase follows, in which the student conceives a solution method and transforms it into an action plan (planning). Finally, the student executes the plan to build a solution (producing).

### *Generating*

Generating involves representing the problem and developing alternatives or hypotheses that meet specific criteria. Figure 5.17 depicts an example of an item that assesses the student's ability to create alternative hypotheses based on criteria.

<b>LEVEL: CREATE</b>	<b>SKILL: GENERATING</b>
<b>ID.: #CREGEN01CS  </b>	<b>CONTENT: CONDITIONAL STATEMENTS</b>

Which model is best suited to solve the following problem:

"If a car exceeds the speed of 80 km/h, it will pay a fine of R\$ 5 per km above the allowed".

For this problem, we can use commands from:

- a) If statement;
- b) If-else statement;
- c) Nested statement;
- d) Case statement.

Figure 5.17: Example of the Item Generating Skill.

### *Planning*

Planning involves designing a solution method that meets a problem's criteria and developing a plan to solve the problem. Figure 5.18 depicts an example of an item that assesses the student's ability to develop a procedure to perform a task.

<b>LEVEL:</b> CREATE	<b>SKILL:</b> PLANNING
<b>ID.:</b> #CREPLA01CS	<b>CONTENT:</b> <b>CONDITIONAL STATEMENTS</b>

Write the step by step on how to solve the following problem:

"If a car exceeds the speed of 80 km/h, it will pay a fine of R\$ 5 per km above the allowed".

Figure 5.18: Example of the Item Planning Skill.

### *Producing*

Producing involves planning to solve a particular problem that meets certain specifications. Figure 5.19 depicts an example of an item that assesses this ability.

<b>LEVEL:</b> CREATE	<b>SKILL:</b> PRODUCING
<b>ID.:</b> #CREPRO01CS	<b>CONTENT:</b> <b>CONDITIONAL STATEMENTS</b>

Create a program that asks the user's car' speed. If it exceeds 80 km/h, display a message saying the user has been fined. In this case, display the fine amount, charging BRL 5 per km above 80 km/h.

Figure 5.19: Example of the Item Producing Skill.

## 5.3 Study Design

This section presents the study design to validate the item bank content and semantics that measures cognitive programming skills.

### 5.3.1 Participants

For the item bank evaluation process, we had a team of experts comprising three university educators from the local community who teach CS1 courses. In addition, we had 35 experienced programmers who participated in the evaluation process.

### 5.3.2 Techniques and Metrics

The survey covered both the item bank semantics and the content. We asked the following questions: i) **semantic analysis**: Indicate the degree of agreement in understanding the item. Is the item understandable? Easy to read? ii) **content analysis**: Indicate the degree of agreement of the item's suitability for the desired skill. Is the item suitable for measuring this skill?

In the semantic analysis, we present a Likert-type scale, with the options for the judge to carry out the evaluation, with the answer options for each question represented as follows: 1—Not understandable: Not at all understandable, not understood, not understood when reading. 2—Not very understandable: 25% understandable, understood, understand very little when reading. 3—Moderately understandable: 50% understandable, understood, understand moderately when reading. 4—Very understandable: 75% understandable, understood, understand intensely when reading. 5—Completely understandable: 100% understandable, understood, understood perfectly when reading.

In the content analysis, we present a Likert-type scale, with the options for the judge to carry out the evaluation, with the answer options for each question represented as follows: 1—Not appropriate: Not at all appropriate, not adapted, not corresponding at all to the objective of the skill proposed. 2—Not very appropriate: 25% appropriate, adapted, corresponding very little to the proposed skill objective. 3—Moderately appropriate: 50% appropriate, adapted, moderately corresponding to the objective of the skill proposed. 4—Very appropriate: 75% appropriate, adapted, and intensely corresponding to the proposed



skill objective. 5—Completely appropriate: 100% appropriate, adapted, corresponding perfectly to the proposed skill objective.

### 5.3.3 Data Analysis

A group of judges inspected the semantics and content present in the item bank. Then, we extracted the data contained in this study through an exploratory survey. We use the function *agree*, available in the IRR package in the R language, to evaluate the agreement's rate between the judges. We used the intraclass correlation coefficient to assess inter-judge reliability (Subsection 4.4.3).

### 5.3.4 Methodological Process

Figure 5.20 represents the methodological process of the item bank semantic and content evaluation stage.

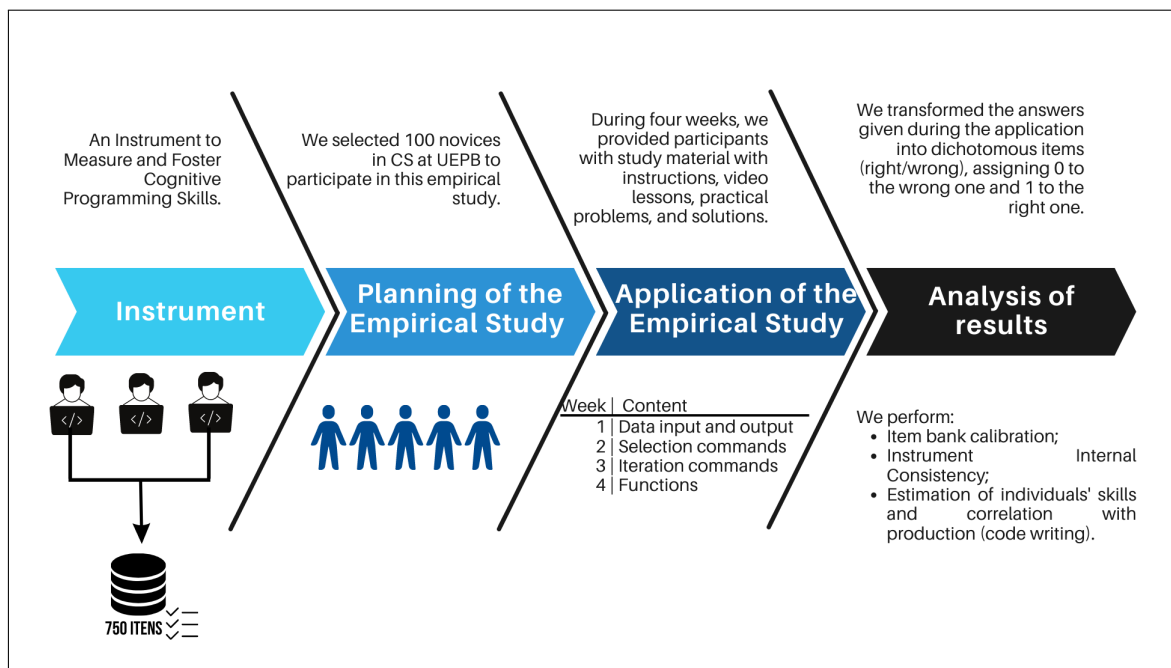


Figure 5.20: Methodological Process of the Item Bank Semantic and Content Evaluation Stage.

We have developed an item bank that includes cognitive programming skills. Each item has a description, alternatives (for multiple-choice items), and a correct answer. We designed

all items so the examinee could express his/her ability by responding to the item.

After preparing the items, we developed a survey (Appendix I) for the items' theoretical evaluation. We applied an online survey to expert judges in the researched area. The judges checked whether the items were well understood (semantic analysis) and adequate to measure the desired skill (content analysis).

### 5.3.5 Threats to Validity

We consider some factors that can generate threats and directly influence this study's conclusions; among them:

- It is possible that the application of this research was long, which may influence the specialists' answers. One way to mitigate this threat was to guarantee the judge's withdrawal at any time during the research. We applied the Ethics Committee in Research with Human Beings' guidelines of the UFCG and the UEPB, which approved this research (Protocols: 23933919.4.0000.5182 | 23933919.4.3001.5187). Only participants who signed the ICF (Appendix H) participated in this study;
- The instrument was automatically corrected to mitigate possible human errors;
- The number of subjects participating in the study does not allow the inference of the results. The criteria must be clarified, and the experts' answers must reflect our expected response for this study. Which may also disqualify the adaptation validation.

## 5.4 Results

In Tables 5.3 5.4, we present the results of the items' content and semantic analysis by skill, respectively. The degree of agreement among the judges for content analysis, between skills, ranged between 90%-97.5%; for semantic analysis, this level ranged between 72.5%-95%. The internal consistency in the responses provided by the judges ranged from 0.740 to 0.979 for content analysis. As for semantic analysis, the responses' internal consistency ranged from 0.716 to 0.897. In addition, the intraclass correlation coefficient indicated excellent reliability between the scores assigned by the judges in the content analysis ranging from

(0.743 – 0.979 [95% = 0.565 – 0.988];  $F_{(39,78)} = 3.848 – 60.888$ ,  $p - value < 0.001$ ).

For the semantic analysis, this index varied between (0.720 – 0.897 [95% = 0.525 – 0.942];

$F_{(39,78)} = 3.681 – 9.619$ ,  $p - value < 0.001$ ).

Skill	Agree.	$\alpha$	ICC	Lower	Upper	F-test	df1	df2	P-value
Recognizing	97.5%	0.970	0.970	0.950	0.983	33.615	39	78	<0.001
Recalling	97.5%	0.921	0.921	0.867	0.956	12.692	39	78	<0.001
Interpreting	97.5%	0.898	0.898	0.829	0.943	9.846	39	78	<0.001
Exemplifying	97.5%	0.952	0.952	0.919	0.973	20.923	39	78	<0.001
Classifying	95%	0.984	0.984	0.973	0.991	60.888	39	78	<0.001
Summarizing	95%	0.875	0.876	0.791	0.930	7.987	39	78	<0.001
Inferring	95%	0.897	0.894	0.822	0.940	9.684	39	78	<0.001
Comparing	95%	0.929	0.928	0.878	0.959	14.013	39	78	<0.001
Explaining	90%	0.966	0.967	0.944	0.981	29.377	39	78	<0.001
Executing	97.5%	0.969	0.969	0.947	0.982	32.000	39	78	<0.001
Implementing	97.5%	0.979	0.979	0.964	0.988	47.154	39	78	<0.001
Differentiating	90%	0.852	0.843	0.713	0.920	6.778	29	58	<0.001
Organizing	95%	0.740	0.743	0.565	0.855	3.848	39	78	<0.001
Attributing	95%	0.938	0.937	0.893	0.964	16.197	39	78	<0.001
Checking	97.5%	0.965	0.965	0.941	0.980	26.692	39	78	<0.001
Critiquing	97.5%	0.921	0.921	0.867	0.956	12.692	39	78	<0.001
Generating	97.5%	0.898	0.898	0.829	0.943	9.846	39	78	<0.001
Planning	97.5%	0.937	0.937	0.895	0.965	16.000	39	78	<0.001
Producing	97.5%	0.959	0.959	0.931	0.977	24.385	39	78	<0.001

Table 5.3: Reliability Analysis of Item Content Among Judge.

Skill	Agree.	$\alpha$	ICC	Lower	Upper	F-test	df1	df2	P-value
Recognizing	85%	0.837	0.832	0.717	0.905	6.124	39	78	<0.001
Recalling	95%	0.853	0.851	0.750	0.916	6.805	39	78	<0.001
Interpreting	92.5%	0.869	0.865	0.772	0.924	7.611	39	78	<0.001
Exemplifying	90%	0.826	0.823	0.703	0.900	5.745	39	78	<0.001
Classifying	80%	0.881	0.880	0.798	0.933	8.374	39	78	<0.001
Summarizing	87.5%	0.716	0.720	0.525	0.843	3.518	39	78	<0.001
Inferring	87.5%	0.766	0.758	0.594	0.864	4.273	39	78	<0.001
Comparing	90%	0.869	0.862	0.766	0.922	7.612	39	78	<0.001
Explaining	72.5%	0.859	0.862	0.766	0.922	7.075	39	78	<0.001
Executing	75%	0.879	0.880	0.798	0.933	8.242	39	78	<0.001
Implementing	87.5%	0.897	0.897	0.826	0.942	9.619	39	78	<0.001
Differentiating	86.7%	0.728	0.723	0.498	0.858	3.681	29	58	<0.001
Organizing	90%	0.729	0.729	0.544	0.847	3.692	39	78	<0.001
Attributing	85%	0.784	0.786	0.638	0.880	4.620	39	78	<0.001
Checking	87.5%	0.802	0.803	0.668	0.889	5.056	39	78	<0.001
Critiquing	95%	0.847	0.851	0.747	0.916	6.550	39	78	<0.001
Generating	95%	0.740	0.743	0.565	0.855	3.848	39	78	<0.001
Planning	95%	0.817	0.813	0.686	0.894	5.461	39	78	<0.001
Producing	92.5%	0.883	0.883	0.803	0.934	8.538	39	78	<0.001

Table 5.4: Reliability Analysis of Item Semantics Among Judges.

This way, we can analyze the judges' reliability and agreement to a high degree. After analyzing the answers among the judges, all items obtained agreement rates above 80%, and

we selected them for the instrument's construction for all skills.

## 5.5 Discussions

Content analysis is an essential step in the development of a new instrument. At this stage, we can associate abstract concepts with measurable indicators. Content analysis determines the degree to which each item of a measurement instrument is relevant and representative of a specific construct. Therefore, the procedures for verifying the evidence validity based on content can assess whether the items refer to the behavior characteristic in question.

The Semantic analysis assesses the understanding of the items by the target audience. In this analysis, judges can make comments to determine the degree of difficulty and specific terms and suggest possible changes in the language of the item.

Judges who are specialists in the subject are responsible for carrying out these steps [101]. In the present study, we used a group of expert judges to analyze evidence on the item's content and semantic validity. The analysis of the judges contributed substantially to the validation process of the items' content, indicating which items could confuse the target population and including behaviors aimed at the Brazilian reality. The semantic analysis identified difficulties in understanding the instructions, examples, and items.

These steps were essential for validating the instrument, as the suggestions given by the judges served to ease the item's understanding, avoiding possible mistakes by the participants. Thus, throughout this investigation, we verified the need for changes in the items to make them more objective. These changes are pertinent to avoid causing doubts to the participants and provide improvements in the instructions' descriptions. We accepted all the suggestions made by the judges and incorporated these adaptations into the items. The collaborations mentioned were helpful and relevant for improving the instrument.

In addition, presenting the instrument to a group of judges who were more experienced programmers made it possible to verify whether the participants could understand their items and instructions. It provided essential reflections on the presentation of the items, enabling their adequacy and instructions through suggestions from the population for which we have designed the instrument.

In short, the methodological procedures performed were relevant. In this context,

instrument validation methods can contribute to the design of future research on the construction of instruments to measure skills. From a practical viewpoint, the instrument can help in the initial exploration of the metacognitive skills of novice programmers. The instrument can provide indications that ease professionals' understanding of these skills and foster them. Therefore, it is necessary to accumulate other valid evidence to determine the quality of the instrument; we will address these studies in the following chapters.

## 5.6 Related Works

Significant research in the CS area did not perform content, and semantic analysis on its items [2, 9, 11, 16, 17, 19–21, 23, 28–30, 32, 33, 35–38, 40–42, 47, 49, 51–55, 57, 60–62, 69, 73, 77, 82–84, 86, 87, 89, 90, 92, 93, 95, 100, 102, 108, 116, 117, 119, 124, 130, 133, 135–140]. This step is essential since the items need to undergo a theoretical evaluation [101]. It is important to use quantitative analysis to give greater consistency, validity and reliability to the items bank [8].

The theoretical assessment of the items involves content analysis and semantic analysis. An experienced group of judges in the target area of the instrument can perform these analyses. The content analysis investigates whether the content is correct and appropriate to the proposed. The semantic analysis seeks to identify whether the target audience understands the items [101].

Recently, one study presented and discussed an instrument's construction and validation stages through psychometric and semantic analyses. The instrument assesses the Critical Thinking skill of high school students. The researchers validated the instrument in four domains: agreement percentage, content validity index, coherence and agreement between participants, and instrument reliability. The analysis investigated whether the 20 items seek to identify the capacities manifestation of Critical Thinking [134].

In this Chapter, we report the instrument's construction based on the adaptation of the cognitive domain of Bloom's Revised Taxonomy or to programming teaching. Then, we analyze the semantics and content of the items present in this instrument through a survey applied to a group of judges. To reduce subjectivity in this analysis, we verify the judges' degree of agreement, consistency, and reliability.

## 5.7 Final Considerations

Based on the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching, we developed an items bank. We performed a theoretical analysis of these items through a survey applied to a group of CS1 specialists. The results indicated that:

- **(RQ5:)** Items have an appropriate content analysis that includes indicators of cognitive programming skills;
- **(RQ6:)** The items have an appropriate semantic analysis; that is, they are understandable.

# Chapter 6

## Assessing the Instrument's Reliability

In this Chapter, we introduce the calibration of the item bank, instrument consistency to measure and foster cognitive programming skills, and these skills' correlation with code writing. We conducted an empirical study, applying the instrument to a group of novices in CS1 at UEPB to analyze whether the constructed scale is minimally adjusted to continue the study.

### 6.1 Initial Considerations

In previous Chapters, we presented the adaptation of the cognitive domain of Bloom's Revised Taxonomy to teaching programming (Chapter 4). This adaptation proposes the development of cognitive programming skills in a sequential order to avoid students' cognitive overload. In addition, we developed an instrument based on this adaptation to measure and encourage such skills. Based on empirical studies, the items have adequate content and semantic analysis (Chapter 5). We have not analyzed the item's psychometric properties and the response's internal consistency to determine the instrument's reliability. Thus, this Chapter has the following objectives:

- **SO6.** To calibrate the item bank in the information terms they provide regarding the specific psychological construct assessed;
- **SO7.** To evaluate the internal consistency of the item bank through measurement theories;

- **SO8.** To investigate which cognitive programming skills are relevant to the participant's ability to program;

We answer the following research questions:

- **RQ7.** Does the instrument have items with appropriate psychometric properties?
- **RQ8.** Does the instrument have appropriate internal consistency?
- **RQ9.** Which of the cognitive programming skills present in the instrument has a strong correlation with the participant's ability to program?

To answer our RQ7, we used the IRT to assess whether the items have appropriate psychometric characteristics [8]. We used the CTT to assess the instrument's internal consistency to answer our RQ8 [96]. Finally, to answer our RQ9, we combined the instrument's abilities to explain the ability to produce code.

The IRT considers an item with appropriate psychometric properties if this item has an index between 0 and 4 for the slope parameter, an index between -3 and 3 for the threshold parameter, and an index less than 40% for the asymptote parameter. The CTT considers an instrument appropriate when it has an index greater than 0.8 for Cronbach's alpha [8,14,101].

Our subjects consist of 100 students who participated in CS1 at UEPB. We developed and performed our study using the Python language in CS1. We provide participants with study material, instructions, educational videos, practice problems, and a database containing items to build and measure their skills.

We then assessed the participants to investigate how much they learned. We verified the relationship between cognitive programming skills (present in the instrument) and the participant's ability to produce code. This analysis is essential and can reduce the number of skills investigated, focusing only on those that have the most significant impact on writing code. To contribute to the state of art, we aim to advance this Chapter's discussions.

We organize the remainder of this Chapter as follows. In Section 6.2, we describe the study design. In Section 6.3, we present the results. After that, in Section 6.4, we discuss the results. In Section 6.5, we discuss the related works. Finally, in Section 6.6, we conclude the Chapter with final remarks.



## 6.2 Design

The study design aims to calibrate the item bank, verify the instrument's consistency, and identify the correlation between the investigated skills and producing (writing code).

### 6.2.1 Participants

We selected 100 novices in CS at UEPB to participate in this empirical study. We estimate the participant's skills during the academic semester. Participants met the following inclusion criteria: i) the participants signed the ICF (Appendix J), and we informed them, in a simplified way, about the procedures we apply; ii) the participant was enrolled in the first semester of a course in CS; and, iii) the participant could not have any sensory, cognitive, auditory, or visual problems so as not to compromise the results of the calibration of the items.

### 6.2.2 Data Analysis

We analyzed the IRT's data collected at this stage with the help of the Excel tool available at: <http://psychometricon.net/libirt/>. We use tools' resources to analyze and adjust the 2PL and 3PL logistic models by the maximum likelihood's marginal estimate.

We analyzed the instrument's reliability through the item's psychometric properties and the internal consistency (correlation between different items in the same Instrument). This procedure is essential in constructing any instrument as it checks if the constructed scale is minimally adequate to continue the study. Through this study, we verify whether the instrument adds sufficient reliability; if not, we should improve that specificity.

Finally, we estimate the individuals' skills and correlate them with code production. This analysis can reduce the number of skills investigated, focusing only on those that have the most significant impact on code production.

### 6.2.3 Methodological Process

Figure 6.1 represents the methodological process of the item bank calibration steps, the instrument's internal consistency, and the correlation of cognitive abilities with the

participants' programming ability.

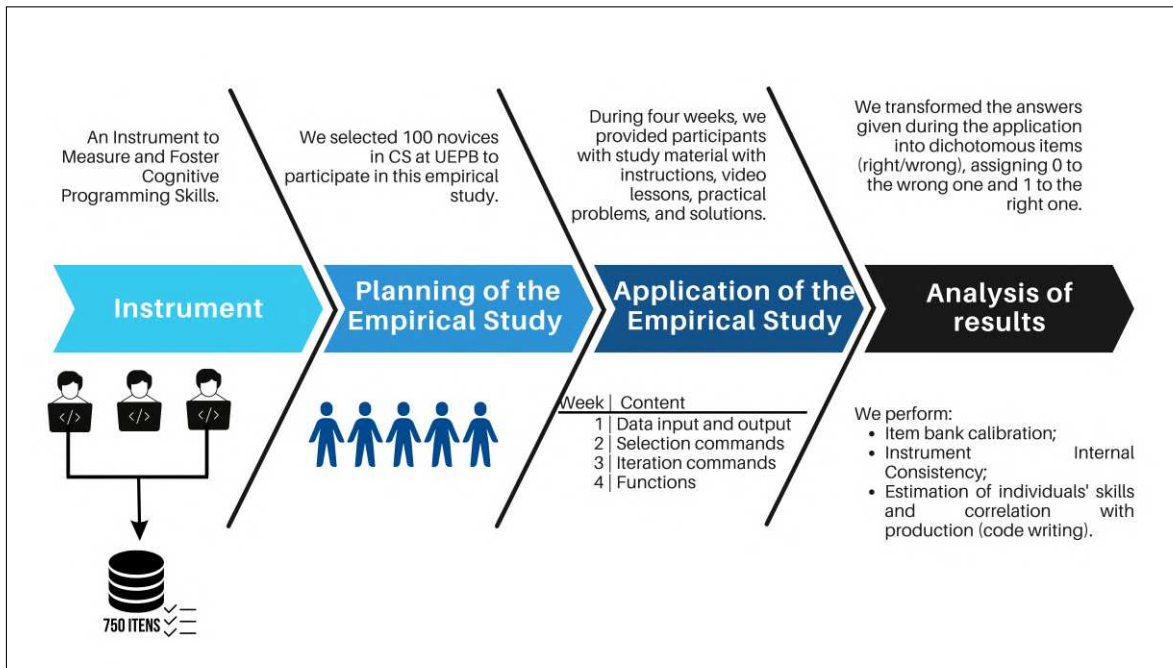


Figure 6.1: Methodological Process of Instrument Reliability Assessment.

We mentioned the purpose of the research to participants who signed the IFC to participate in the study. **We divided the tasks so that some were virtual and others face-to-face to ensure the accessibility of the study participants.** Individuals needed a cell phone or a computer with an internet connection to respond to the assessments in the virtual groups.

During four weeks, we provided participants with study material, instructions, video lessons, practical problems, and solutions. The participants worked on the content at their own pace. However, we suggested an agenda to simplify the study control, noting that the response should be made without online consultation or with a tutor of programming courses. Each participant answered all items present in the instrument.

We transformed the answers given during the application into dichotomous items (right/wrong), assigning 0 to the wrong one and 1 to the right one. We calibrated the item bank and analyzed the instrument consistency. Finally, we estimated the individuals' abilities and correlated them with producing (writing code).

### 6.2.4 Threats to Validity

We consider some factors that generated threats and directly influenced the conclusions of this phase. Among these threats, we have:

- Problems related to the incorrect interpretation of the questions;
- The survey participants may be intimidated or uncomfortable performing the tests. To mitigate these threats, we applied the Ethics Committee in Research with Human Beings' guidelines of the UFCG and the UEPB, which approved this research (Protocols: 23933919.4.0000.5182 | 23933919.4.3001.5187). Only participants who signed the free and ICF participated in this study;
- Part of the answers given to the instrument was manually corrected. Human errors can happen in corrections. However, part of the responses given by the participants involved computerized applications and corrections to reduce possible errors;
- Like all empirical research, this work has threats to validity. The subjects number participating in the study does not allow results' generalization;
- The sample must be considered since it will allow the creation of a database based on probability, statistics, measurement axioms, and considering the instrument's objective. This bank must have centralized application control.

## 6.3 Results

This section presents the instrument's reliability and the relationship between cognitive skills and Producing skills (writing code). We analyzed the instrument's reliability through the items' psychometric properties and internal consistency.

### 6.3.1 Items' Psychometric Properties

We interpreted the distribution of the participant responses on each task item using 2PL/3PL. Planning and Producing skills are open items, so their model is 2PL; we consider only slope and threshold parameters. The items for the other skills are multiple-choice. We use the 3PL,

considering the slope, threshold, and asymptote. In addition, we considered the hit ratio and the point-biserial correlation between the correct answer on the item and the total score on the task. We present the output part in Table 6.1, in which we present the first ten items of the 40 that compose the recognizing skill with the respective parameters. The complete item table and all skills are available in Appendix K.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
REMREC01ES	0.912	0.619	0.200	0.470	0.234
REMREC02ES	1.702	0.000	0.200	0.380	0.044
REMREC03ES	0.847	-0.189	0.152	0.560	0.296
REMREC04ES	0.953	0.350	0.167	0.480	0.289
REMREC05ES	1.346	1.139	0.187	0.340	0.225
REMREC06ES	1.193	-2.311	0.161	0.900	0.280
REMREC07ES	1.331	-0.466	0.160	0.620	0.336
REMREC08ES	1.007	0.218	0.157	0.490	0.318
REMREC09ES	1.181	-1.083	0.192	0.750	0.315
REMREC10ES	1.016	-1.587	0.149	0.790	0.272
<b>Mean</b>	1.386	-0.179	0.171	0.561	0.303
<b>Standard deviation</b>	0.445	0.997	0.034	0.179	0.088

Table 6.1: Recognizing Skill—Ten First Items Calibrated.

In Figure 6.2, we present the ICC's graphic representation for the first ten items of the Recognizing skill, highlighting the extreme values of the slope, threshold, and asymptote parameters. Moreover, in Figure 6.3, we present the IIF's graphic representation for the first ten items of the Recognizing skill. Each item provides information in a specific latent trait region. All item figures and all skills are available in Appendix L and Appendix M, respectively.

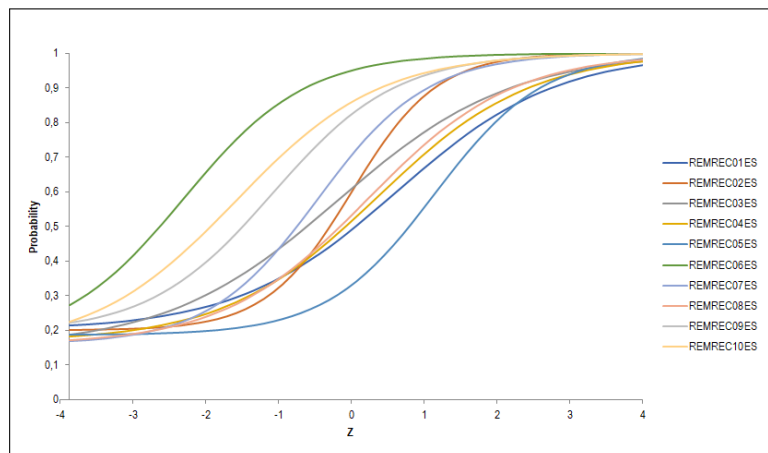


Figure 6.2: Recognizing Skill ICC's With Ten First Items Calibrated.

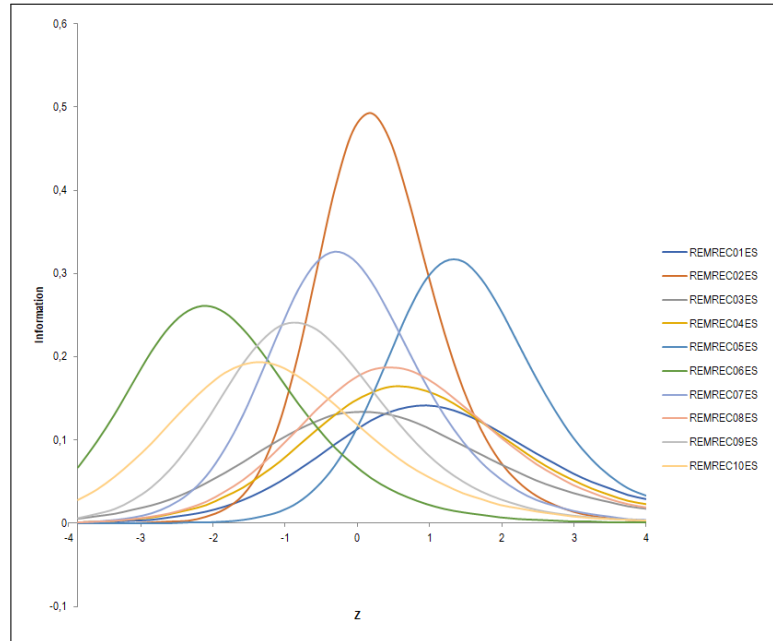


Figure 6.3: Recognizing Skill IIF's With Ten First Items Calibrated.

### 6.3.2 Internal Consistency

In Table 6.2, we present the instrument's internal consistency through the values obtained by Cronbach's alpha.

Skill	Items	Score's mean	Standard deviation	Cronbach's alpha
Recognizing	40	22.450	6.709	0.830
Recalling	40	22.600	6.902	0.840
Interpreting	40	22.630	6.985	0.844
Exemplifying	40	22.650	7.178	0.854
Classifying	40	22.710	7.252	0.858
Summarizing	40	22.680	7.193	0.855
Inferring	40	22.670	7.092	0.850
Comparing	40	22.770	7.341	0.862
Explaining	40	22.610	7.120	0.851
Executing	40	22.710	7.338	0.861
Implementing	40	22.650	7.245	0.857
Differentiating	30	18.910	5.757	0.837
Organizing	40	22.540	7.274	0.859
Attributing	40	22.500	7.235	0.857
Checking	40	22.500	7.282	0.859
Critiquing	40	22.500	7.311	0.860
Generating	40	22.520	7.425	0.866
Planning	40	22.520	7.994	0.887
Producing	40	22.530	8.202	0.894

Table 6.2: Internal Consistency of the Instrument.

### 6.3.3 Factor Correlation Analysis

We investigated the relationship between cognitive programming skills (present in the instrument) and the participant's ability to produce code. We aim to investigate which of these skills strongly correlates with the Producing skill (code writing). We estimated the participants' abilities who answered the instrument and used the EAP method. Figure 6.4 provides the data summary.

Table 6.3 shows the measures of the central tendency, where the median values are similar, indicating a symmetrical distribution. The table also presents the standard deviation dispersion estimates, where it is possible to observe different values and outliers. At the end of the table, we present the Shapiro test for data normality.

Thus, Comparing, Organizing, Planning, and Producing skills estimates do not follow a normal distribution. The others follow a distribution with a confidence level of 95%. Some skill estimates did not follow a normal distribution. Therefore, we correlated skills using the "spearman" method. We illustrate the correlation between skills using a correlogram, as shown in Figure 6.5.

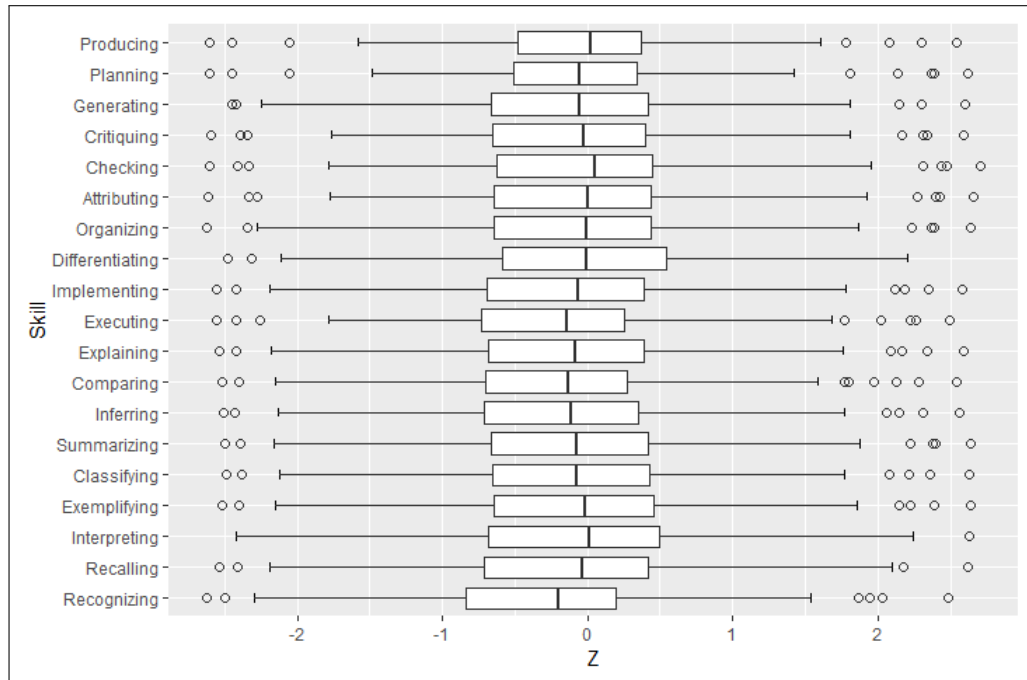


Figure 6.4: Boxplot in the Pre-test With the Estimate of the Subjects' Skills.

Skill	Median	Standard deviation	Shapiro	P-value
Recognizing	-0.204	0.901	0.974	0.053
Recalling	-0.034	0.943	0.983	0.234
Interpreting	0.015	0.929	0.981	0.168
Exemplifying	-0.020	0.968	0.981	0.164
Classifying	-0.073	0.982	0.971	0.030
Summarizing	-0.074	0.977	0.976	0.070
Inferring	-0.108	0.947	0.978	0.104
Comparing	-0.133	0.979	0.967	0.014
Explaining	-0.086	0.955	0.980	0.142
Executing	-0.141	0.965	0.976	0.067
Implementing	-0.061	0.972	0.980	0.146
Differentiating	-0.004	0.954	0.987	0.455
Organizing	-0.006	0.978	0.974	0.050
Attributing	0.005	0.979	0.975	0.055
Checking	0.048	0.996	0.997	0.099
Critiquing	-0.024	0.965	0.975	0.057
Generating	-0.051	0.975	0.976	0.071
Planning	-0.053	0.994	0.937	<0.001
Producing	0.021	1.001	0.934	<0.001

Table 6.3: Exploratory Data Analysis of the Correlation Between the Skills.

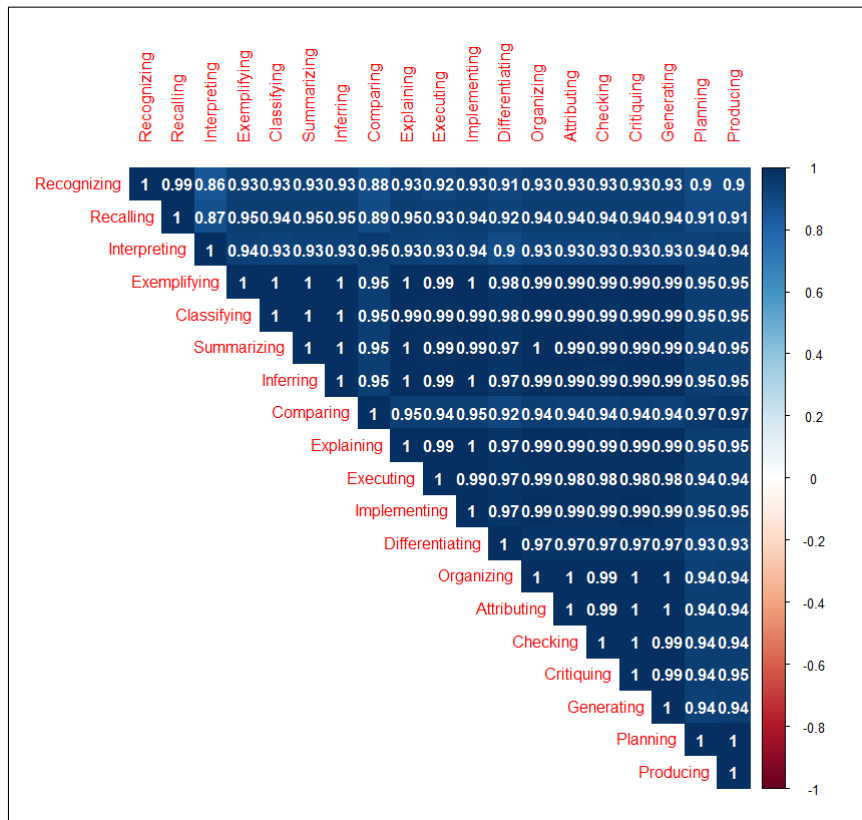


Figure 6.5: Correlation Heat Map Between the Skills.

## 6.4 Discussions

In this section, we discuss the previous results. Through CTT and IRT, we determined the instrument's construct validity indications.

### 6.4.1 Does the Instrument Have Items With Appropriate Psychometric Properties?

Using the IRT, we estimated the item's psychometric properties. We evaluated the conditions of the required item parameters for the 2PL/3PL model to avoid compromising the domain. After verifying the 3PL logistic model, we did not find critical values for the estimated parameters. All items for the skills of Recognizing, Recalling, Interpreting, Exemplifying, Classifying, Summarizing, Inferring, Comparing, Explaining, Executing, Implementing, Differentiating, Organizing, Attributing, Checking, Critiquing, and Generating have values greater than 0.30 for the slope; for the threshold values between 3.95 and -3.95; and, the asymptote below 0.40. Regarding the 2PL logistic model, we did not find critical values for the estimated parameters; all items for the Planning and Producing skills have values within the expected outcomes.

The threshold parameter presents values represented within the entire latent trait, representing subjects with varying skill levels. In short, this result is excellent given the complexity involved in estimating the difficulty of an item based only on the designer's tacit knowledge, without considering the empirical results. Furthermore, the results on the correct answer rate revealed that all skills have easy items (with scores above 75%), moderate items (with scores between 50 and 75%), and difficult items (with scores below 50%).

The point-biserial correlations revealed that the participants with higher scores for all skills tended to choose the wrong option. Despite this, all items fit correctly to 2PL/3PL, so they have appropriate reliability and an appropriate skill separation index.

Among the items presented in Table 6.2 and Figure 6.2, the easiest item is "REMREC06ES," and the most difficult is "REMREC05ES." The item with the most guesses is "REMREC02ES," which is also the item with the highest discrimination. The item "REMREC03ES" is the least discriminating. Whereas the item with the lowest guesses is "REMREC10ES." The skills' asymptote with multiple choice items presents average



rates below 17.1%, which is below the expected since the items of these skills have four alternatives each. It can be mathematically affirmed that the probability of a low-skill student hitting the item is approximately equal to 25%.

Regarding Figure 6.3, the item “REMREC02ES” offers more information to assess subjects of average ability, that is, and a computerized instrument. This item will be the first chosen if the instrument places the  $\theta$  skill at 0 as the average skill.

### **6.4.2 Does the Instrument Have Appropriate Internal Consistency?**

Using the CTT, we calculated the biserial point correlation and Cronbach’s alpha coefficient to estimate the instrument’s internal consistency. High biserial point correlation values contributed to the instrument’s internal consistency appropriateness. High item-total correlations indicate that items are closely associated with each other. From the values obtained for Cronbach’s alpha in Table 6.1, we conclude that the results obtained in evaluating the instrument are reliable for all skills.

Appropriate internal consistency indicates that the instrument has items that measure a single latent trait. We can evaluate each skill purely through this instrument. Although our instrument has 19 skills, we can measure and foster them individually.

Allied with the results of the previous section, for IRT, these findings guarantee the construction of a one-dimensional model with local independence. By one-dimensionality, we understand that the instrument measures a single latent trait at a time, which may represent a proficiency or even a composition of abilities and proficiencies of the assessed subjects (in our case). By local independence, we understand that the latent trait of the evaluated ones perfectly explains the dependency between the items. In such a way, it is possible to position items (based on the threshold parameter) and subjects in the same scale (represented in the plane by their estimated ability).

### **6.4.3 Which of the Cognitive Programming Skills Present in the Instrument Has a Strong Correlation With the Producing Skill?**

All the cognitive skills investigated in the instrument strongly correlate with the ability to produce code. The data suggest that this correlation increases when the student advances

in the cognitive levels in adapting Bloom's Revised Taxonomy to programming teaching, which indicates that learning occurs linearly.

## 6.5 Related Works

Various approaches and instruments support educators and students in improving the evaluation process in programming teaching. A study reports a project that explored students' skills in basic syntax, tracking code, comprehension code, and writing code, aiming to establish the relationships between these skills [75].

However, these works have no constructive validity in their instrument. For CCT and IRT, the trust and validation of an instrument are critical steps since they will be the ones that will determine the level of scientific evidence that the Instrument has on a given construct [8,96].

In CS [7,16,132], the practice of these theories cited has become common in the scientific community. Studies have proven the instrument's validity that measures individual skills. Bergersen et al. [16] have developed/validated an instrument that measures programming skills. The Instrument had satisfactory (internal) psychometric properties and correlated with external variables according to theoretical expectations. Araújo et al. [7] explored the computational thinking evaluation in CS1 using the Bebras test. They conducted a preliminary study using IRT to improve the item selection and the instrument design. Wang et al. [132] used IRT to model students' learning results in a programming language course.

One study assessed whether there is still a difference between tracing and writing code. This study found that not only does the gap appear to be missed by CS2, but students are just as likely to show a reverse gap in the writing-tracing direction. However, a more accurate analysis reveals a strong correlation between students who remain with a gap (in either direction) and poor overall achievement in the course [52]

This study used CTT and IRT to measure cognitive skills according to Bloom's Revised Taxonomy for programming teaching (Chapter 4). We evaluated the instrument's reliability, assessing its internal consistency and whether the items had appropriate psychometric characteristics. Finally, we combined the instrument's skills to explain the skill of producing code.

## 6.6 Final Considerations

In this study, we explored the IRT/CTT to analyze the instrument's internal consistency and items' psychometric properties by measuring the programming skills. We developed an instrument based on an adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching. As a result, we conclude that:

- **(RQ7:)** the 750 items present in the instrument are well-calibrated according to the slope, threshold, and asymptote parameters;
- **(RQ8:)** the instrument for each skill has a reliable internal consistency, with Cronbach's alpha ranging between 0.8 and 1;
- **(RQ9:)** all the cognitive skills investigated in the instrument present strong correlations with the ability to produce code. The data suggested that this correlation increases when the student advances in the cognitive levels within the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching, which indicates that learning occurs linearly.

# Chapter 7

## Integrating Adaptive Selection to the Instrument

In this Chapter, we aim to integrate adaptive selection into the instrument to improve the measurement of cognitive programming skills. We developed algorithms based on the IRT, defining an initial criterion for skill, item selection procedure, skill estimation method, and different stopping criteria. To determine the best algorithm for the items' adaptive selection, we used the item bank and feedback from study participants in Chapter 6. We performed simulations of how the instrument would behave through these inputs.

### 7.1 Initial Considerations

We developed an instrument with understandable items and indicators of cognitive programming skills (Chapter 5). The instrument has adequate internal consistency and items that have adequate psychometric properties. In addition, all cognitive skills investigated in the instrument showed strong correlations with the ability to produce code (Chapter 6). Thus, the instrument measures cognitive programming skills through items presented in sequential order.

However, the sequence/number of items must be appropriate for the student's skill level. An item with a low level of difficulty for high-ability subjects becomes exhausting, as the correct answers add little information to the proficiency assessment of these individuals [103]. In addition, items that do not offer any challenge can make the individual

bored, responding without further care. Likewise, items with high degrees of difficulty for subjects with low proficiency can become exhausting. Such items allow individuals to guess and incorporate incorrect responses into items. This fact may influence the student's withdrawal from the practice of the exercise [122]. Therefore, items presented sequentially may not discriminate well between low/high-ability students. A poorly chosen item can overestimate or underestimate the individual's ability, directly influencing the estimate of the student's ability on the instrument.

We can improve this reality if the algorithm chooses the item based on the student's skill level. These changes may reduce the number of items to be administered in the assessment and will estimate skill more accurately. Thus, it is possible to use the IRT and manage the Instrument items through an algorithm that adaptively selects them according to the subject's skill level.

The IRT considers the item as the basic unit of analysis and aims to represent the individual probability of answering the item according to its parameters and latent characteristics. This way, the adaptive selection of items can help professionals solve problems requiring quality tools and specialized knowledge. The same set of items is applied to all individuals, allowing greater accuracy, speed, and ease of updating. In addition, this instrument is less error-prone in disseminating its results [111]. Therefore, this Chapter aims:

- **SO9.** To integrate adaptive selection through algorithms that improve performance in estimating students' abilities;

Integrating adaptive selection into an instrument aims to improve the measurement of cognitive programming skills. In this way, we developed algorithms for the adaptive selection of items (Section 7.3). We analyzed them to determine the best option for selecting items and estimating ability. Thus, we answer the following question:

- **RQ10.** Which adaptive algorithm has better accuracy in estimating cognitive programming skills?

We organize the remainder of this Chapter as follows. In Section 7.2, we describe the study design. In Section 7.3, we present the adaptive algorithms' construction. In Section 7.4, we present the results. In Section 7.5, we discuss the results. In Section 7.6,

we discuss the related works. Finally, in Section 7.7, we conclude the Chapter with final remarks.

## 7.2 Adaptive Algorithms

In Figure 7.1, we present the criteria adopted for developing an adaptive algorithm targeting item selection.

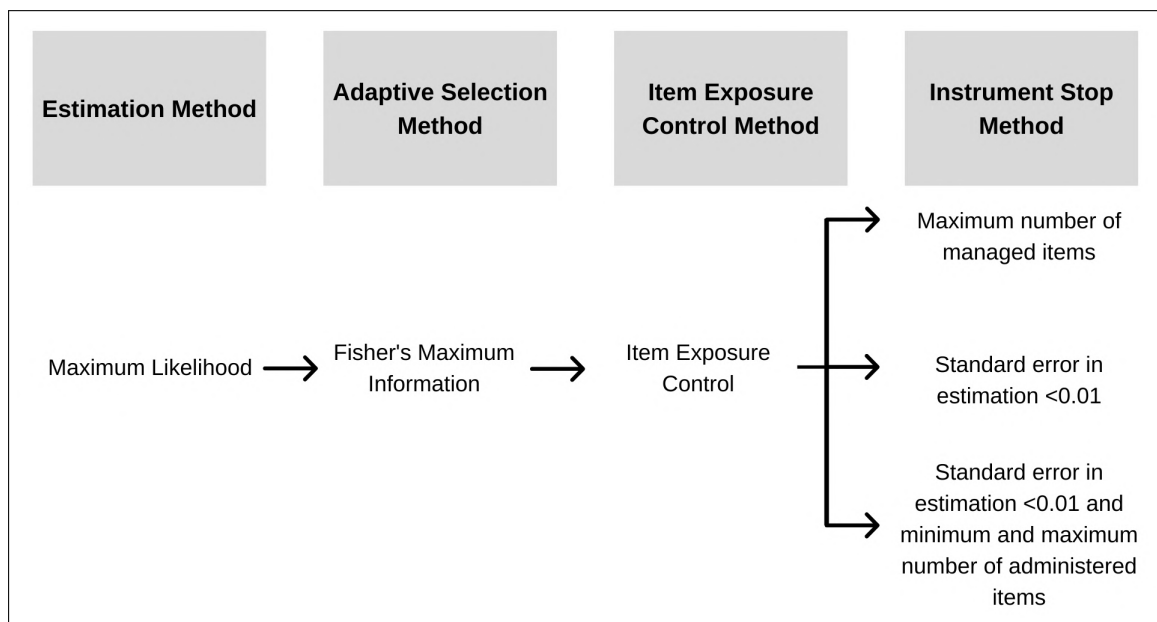


Figure 7.1: Criteria for Elaborating Algorithms Targeting the Selection of Adaptive Items.

We adopted the maximum likelihood method [13] as a procedure for estimating the individuals' abilities (Algorithm 1). For the item selection, we chose Fisher's Maximum Information method [101] (Algorithm 2). This method depends on the probability of an individual getting an item right (Algorithm 3). We control item exposure through auto-selection. Once the algorithm administers the item to the subject, the algorithm will not select this item in subsequent estimations.

**Algorithm 1** Maximum Likelihood Method.

---

```

denTheta ← 0.0f
numTheta ← 0.0f
numTheta ← numTheta + (a[id] * (mirror[iS][idmirror] - p[id]))
denTheta ← denTheta + ((a[id] * a[id]) * p[id] * (1 - p[id]))
theta ← theta + (numTheta/denTheta)
if theta < -3.0f then
    theta ← -3.0f
else if theta > 3.0f then
    theta ← 3.0f
end if
erro ← (1/Math.sqrt(denTheta))
if admin = 0 then
    difErro ← erro - 0
else if admint > 0 then
    difErro ← erroAux - erro
end if
erroAux ← erro

```

---

We summarize the procedure for calculating Maximum likelihood as follows:

1. *theta* starts with a value of zero, as the algorithm does not have prior information about the examinee;
2. The algorithm administers an item to the examinee. Based on the answer provided by the user (1—right or 0—wrong), the *theta* will be updated based on the following formula:  $theta = theta + (numTheta + (a[id] * (mirror[iS][idmirror] - p[id])))/denTheta + ((a[id] * a[id]) * p[id] * (1 - p[id]))$ . The *theta* depends on the slope parameter, the answer provided, and the individual's probability of getting the item right. If *theta* is less than -3, the algorithm updates *theta* to -3. However, if *theta* is greater than 3, the algorithm updates *theta* to 3;
3. Next, the algorithm calculates the error in estimating *theta* based on the following formula  $error = (1/Math.sqrt(denTheta + ((a[id] * a[id]) * p[id] * (1 - p[id]))))$ ;

4. At each estimate of  $\theta$ , the algorithm calculates the  $difError$ . If the number of items managed by the algorithm equals zero, the algorithm calculates this difference based on  $error - 0$ . Otherwise, the algorithm calculates this difference based on the last two errors in the skill estimates;
5. This procedure returns the  $\theta$  and the  $difError$ .

---

**Algorithm 2** Fisher's Maximum Method.
 

---

```

max ← 0.0f
index ← 0
while index < i.length do
  if model is 3-parameter then
    i[index] ← (a[index] * a[index]) * ((1.0f - p[index])/p[index]) * ((p[index] -
c[index]) * (p[index] - c[index]))/((1.0f - c[index]) * (1.0f - c[index]))
  else if model is 2-parameter then
    i[index] ← (a[index] * a[index]) * p[index] * (1.0f - p[index])
  end if
  index ← index + 1
end while
index ← 0
while index < i.length do
  if i[index] > max AND i[index] = 0 then
    max ← i[index]
    id ← index
    idmirror ← id + 1
  end if
  index ← index + 1
end while

```

---

Fisher's maximum information procedure determines the item's information in each latent trait. The algorithm will select the item that obtains the most information. We summarize this procedure as follows:



1. As we do not know which item (*index*) the algorithm will select, as well as the maximum information (*max*). The algorithm starts *index* and *max* with zero;
2. While the *index* is smaller than the size of the information vector  $i[index]$ , each index of the vector  $i[index]$  will be updated according to the logistic model adopted. If the logistic model is 3PL,  $i[index]$  will be updated based on the following formula:  

$$i[index] = (a[index] * a[index]) * ((1.0f - p[index])/p[index]) * ((p[index] - c[index]) * (p[index] - c[index])) / ((1.0f - c[index]) * (1.0f - c[index]))$$
 If the logistic model is 2PL,  $i[index]$  will be updated based on the following formula:  

$$i[index] = (a[index] * a[index]) * p[index] * (1.0f - p[index])$$
 For 2PL, item information depends on the slope parameter and the probability of hitting the item (Algorithm 3). For 3PL, in addition to the above information, the asymptote parameter is also required to calculate item information;
3. After filling the  $i[index]$  array, the algorithm will select the highest value using a sequential search;
4. This algorithm returns the item (*index*) that has more information (*max*) in the latent trait.

---

**Algorithm 3** Probability of Hitting the Item.
 

---

*index*  $\leftarrow$  0

**while** *index* < *p.length* **do**

**if** model is 3-parameter **then**

$den \leftarrow (1.0f + \text{Math.exp}(-a[index] * (theta - b[index])))$

$p[index] \leftarrow c[index] + ((1.0f - c[index])/den)$

**else if** model is 2-parameter **then**

$den \leftarrow (1.0f + \text{Math.exp}(-a[index] * (theta - b[index])))$

$p[index] \leftarrow (1.0f/den)$

**end if**

$index \leftarrow index + 1$

**end while**

---

We summarize the procedure for calculating the probability of getting the item right as follows:

1. The *index* starts at zero and controls the position of the item within the probability vector  $p[index]$ ;
2. While the *index* is smaller than the size of the probability vector  $p[index]$ , each index of the vector  $p[index]$  will be updated according to the adopted logistic model. If the logistic model is 3PL,  $p[index]$  will be updated based on the following formula:  $p[index] = c[index] + ((1.0f - c[index]) / (1.0f + \text{Math.exp}(-a[index] * (\text{theta} - b[index]))))$ . If the logistic model is 2PL,  $p[index]$  will be updated based on the following formula:  $p[index] = (1.0f / (1.0f + \text{Math.exp}(-a[index] * (\text{theta} - b[index]))))$ . For 2PL, the item probability depends on the slope and threshold parameters. For 3PL, in addition to these parameters, the asymptote parameter is also needed to calculate the probability of getting the item right;
3. This algorithm returns the probability vector that the individual has in hitting each of the items ( $p[index]$ ) according to his latent trait.

It is essential to determine when the individual will no longer answer to items (stopping criterion). We vary the algorithms on the instrument's stopping criterion in three ways:

- **Z1.** maximum number of managed items;
- **Z2.** estimation standard error <0.01;
- **Z3.** previous criteria combination and the minimum and maximum number of managed items.

We developed a script to find the estimate of each skill, the number of items administered, and if the subject responded to the adaptive instrument for each stopping criterion—Z1, Z2, and Z3, respectively. We found these estimate values by simulating the instrument application based on the answers given by the participants in the study of chapter 5. Next, we detail these algorithms.

### 7.2.1 Adaptive Selection Algorithm Z1

In the Algorithm 4, we generated the estimate of the subjects' skills adaptively. The algorithm manages all items according to the subject's skill level.

The algorithm 4 works as follows:

1. For this algorithm to work, it is necessary: the number of items for the skill to be estimated (*item*); vectors containing slope ( $a[index]$ ), threshold ( $b[index]$ ) and asymptote ( $c[index]$ ) indices of all items for skill a to be cherished; vectors to store, at each iteration, the item's information ( $i[index]$ ) and the item's probability ( $p[index]$ ); a vector that controls the exposure of items ( $ic[index]$ ), and the responses given by users to each administered item ( $mirror[iS][index]$ );
2. For each user managed by the algorithm, *theta*, *error*, the vector ( $ic[index]$ ) and the number of managed items (*admint*) are reset;
3. As long as there are users to be managed, the algorithm will manage the items and estimate the users' skills;
4. *theta* starts with a value of zero, as the algorithm does not have prior information about the examinee;
5. The algorithm calculates the probability of hitting the item and the item information based on the current *theta*;
6. Based on the item information, the algorithm will select the item that contains the maximum fisher information;
7. The algorithm manages this item and adds it to the array of items we manage, thus controlling its exposure;
8. Based on the answer provided by the examinee, the algorithm estimates their ability;
9. The algorithm repeats steps 5, 6, 7, and 8 until the number of administered items (*admint*) is less than the number of skill items (*item*);
10. The algorithm returns the administered items and the individual's estimated skill at each iteration.

**Algorithm 4** Adaptive Selection Algorithm Z1.**Require:**

$item \leftarrow 40$  ▷ Insert the quantity of items for the skill  
 $a \leftarrow []\{\}$  ▷ Insert the calibrated items for the slope parameter  
 $b \leftarrow []\{\}$  ▷ Insert the calibrated items for the threshold parameter  
 $c \leftarrow []\{\}$  ▷ Insert the calibrated items for the asymptote parameter  
 $p \leftarrow [item]\{\}$  ▷ Probability to hit the item  
 $i \leftarrow [item]\{\}$  ▷ Item Information Function  
 $ic \leftarrow [item]\{\}$  ▷ [Item Expose Control Method](#)  
 $mirror \leftarrow [][]\{\}$  ▷ Insert the participant responses

**Ensure:**

$iS \leftarrow 0$   
 $idStudent \leftarrow mirror[iS][0]$   
**do**  
 $admint \leftarrow 0$   
 $theta \leftarrow 0.0f$   
**do**  
 $probI()$  ▷ Calculates the subject's probability of answering the item  
 $infoI()$  ▷ [Fisher's Maximum Information Method](#)  
 $newTheta()$  ▷ [Maximum Likelihood Method](#)  
 $ic[id] \leftarrow 1$   
 $admint \leftarrow admint + 1$   
**while**  $admint < item$  ▷ [Instrument Stop Method](#)  
print theta  
print admint  
 $ic \leftarrow newint[]\{0, \dots, 0\}$   
 $iS \leftarrow iS + 1$   
**if**  $iS < mirror.length$  **then**  
 $idStudent \leftarrow mirror[iS][0]$   
**end if**  
 $denTheta \leftarrow 0.0f$   
 $numTheta \leftarrow 0.0f$   
 $erro \leftarrow 0.0f$   
**while**  $iS < mirror.length$

## 7.2.2 Adaptive Selection Algorithm Z2

In the Algorithm 5, we add the standard error  $<0.01$  as a stopping criterion. Thus, based on the data obtained, we analyzed that some subjects would respond quickly without needing to manage all the items in the bank. Then, we obtained the frequency of items applied, as shown in Figure 7.2 and Table 7.1.

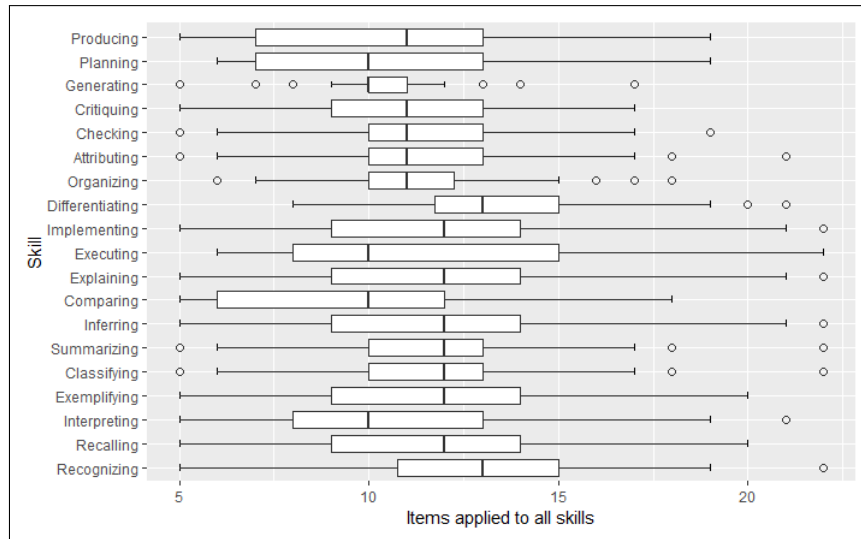


Figure 7.2: Item Frequency Boxplot Applied Across All Skills for the Z3.

	Min.	1 quartile	Average	S. d.	3 quartile	Max.
Recognizing	5	10.75	12.610	3.673	15	22
Recalling	5	9	11.460	3.751	14	20
Interpreting	5	8	10.870	4.012	13	21
Exemplifying	5	9	11.600	3.774	14	20
Classifying	5	10	11.780	3.070	13	22
Summarizing	5	11	11.670	3.012	13	23
Inferring	5	11	11.780	3.886	13	22
Comparing	5	6	9.970	3.471	12	18
Explaining	5	9	11.650	4.215	12	22
Executing	6	8	11.110	4.362	15	22
Implementing	5	9	11.960	4.144	14	22
Differentiating	8	11.75	13.240	2.931	15	21
Organizing	6	10	10.980	2.542	12.25	18
Attributing	5	10	11.190	2.856	13	21
Checking	5	10	11.410	3.072	13	19
Critiquing	5	9	10.950	2.826	13	17
Generating	5	10	10.100	2.254	11	17
Planning	6	7	10.370	3.311	13	19
Producing	5	7	10.590	3.450	13	19

Table 7.1: Minimum and the Maximum Number of Items for All Skills.

**Algorithm 5** Adaptive Selection Algorithm Z2.**Require:**

$item \leftarrow 40$  ▷ Insert the quantity of items for the skill  
 $a \leftarrow []\{\}$  ▷ Insert the calibrated items for the slope parameter  
 $b \leftarrow []\{\}$  ▷ Insert the calibrated items for the threshold parameter  
 $c \leftarrow []\{\}$  ▷ Insert the calibrated items for the asymptote parameter  
 $p \leftarrow [item]\{\}$  ▷ Probability to hit the item  
 $i \leftarrow [item]\{\}$  ▷ Item Information Function  
 $ic \leftarrow [item]\{\}$  ▷ [Item Expose Control Method](#)  
 $mirror \leftarrow [][]\{\}$  ▷ Insert the participant responses

**Ensure:**

$iS \leftarrow 0$   
 $idStudent \leftarrow mirror[iS][0]$   
**do**  
 $admint \leftarrow 0$   
 $theta \leftarrow 0.0f$   
**do**  
 $probI()$  ▷ Calculates the subject's probability of answering the item  
 $infoI()$  ▷ [Fisher's Maximum Information Method](#)  
 $newTheta()$  ▷ [Maximum Likelihood Method](#)  
 $ic[id] \leftarrow 1$   
 $admint \leftarrow admint + 1$   
**while**  $admint < item$  AND  $difErro > 0.01$  ▷ [Instrument Stop Method](#)  
print theta  
print admint  
 $ic \leftarrow newint[]\{0, \dots, 0\}$   
 $iS \leftarrow iS + 1$   
**if**  $iS < mirror.length$  **then**  
 $idStudent \leftarrow mirror[iS][0]$   
**end if**  
 $denTheta \leftarrow 0.0f$   
 $numTheta \leftarrow 0.0f$   
 $erro \leftarrow 0.0f$   
**while**  $iS < mirror.length$

The only difference between the algorithm 4 and the algorithm 5 is in step 9, where the stopping criterion is a combination of two factors: the number of items managed (*admint*) is less than the number of skill items (*item*), and *difErro* is less than 0.01.

### 7.2.3 Adaptive Selection Algorithm Z3

We use the previous results to generate the algorithm with stopping criterion Z3 (Algorithm 6). It consists of the estimated standard error when less than 0.01 and the minimum and maximum number of administered items. We found the minimum and maximum values based on the first and third quartiles of the data set represented by the boxplot (Fig. 7.2)

According to Table 7.1 and Figure 7.2, the selection algorithm will not select more items when the standard error of the skill calculation is less than 0.01 and the number of applied items is at least nine and a maximum of 14, whichever comes first, for the Recalling, Implementing and Exemplifying skill. Classifying, Attributing, and Checking skills are between 10 and 13 items. For Planning and Producing skills between 7 and 13. For Summarizing and inferring skills, between 11 and 13 items, recognizing between 11 and 15, interpreting between 8 and 13, comparing between 6 and 12, explaining between 9 and 12, executing between 8 and 15, differentiating between 12 and 15, organizing between 10 and 12, and critiquing between 9 and 13, finally, for the Generating skill between 10 and 11.

The only difference between the algorithm 5 and the algorithm 6 is in step 9, where the stopping criterion is a combination of two factors: the number of managed items (*admint*) is in the skill's minimum, and maximum item amount range and *difError* is less than 0.01, whichever comes first.

**Algorithm 6** Adaptive Selection Algorithm Z3.**Require:**

$item \leftarrow 40$  ▷ Insert the quantity of items for the skill  
 $a \leftarrow []\{\}$  ▷ Insert the calibrated items for the slope parameter  
 $b \leftarrow []\{\}$  ▷ Insert the calibrated items for the threshold parameter  
 $c \leftarrow []\{\}$  ▷ Insert the calibrated items for the asymptote parameter  
 $p \leftarrow [item]\{\}$  ▷ Probability to hit the item  
 $i \leftarrow [item]\{\}$  ▷ Item Information Function  
 $ic \leftarrow [item]\{\}$  ▷ [Item Expose Control Method](#)  
 $mirror \leftarrow [][]\{\}$  ▷ Insert the participant responses

**Ensure:**

$iS \leftarrow 0$   
 $idStudent \leftarrow mirror[iS][0]$   
**do**  
 $admint \leftarrow 0$   
 $theta \leftarrow 0.0f$   
**do**  
 $probI()$  ▷ Calculates the subject's probability of answering the item  
 $infoI()$  ▷ [Fisher's Maximum Information Method](#)  
 $newTheta()$  ▷ [Maximum Likelihood Method](#)  
 $ic[id] \leftarrow 1$   
 $admint \leftarrow admint + 1$   
**while** ( $admint < minItem$  OR ( $admint < maxItem$  AND  $difErro > 0.01$ )) ▷

**Instrument Stop Method**

print theta  
print admint  
 $ic \leftarrow newint[]\{0, \dots, 0\}$   
 $iS \leftarrow iS + 1$   
**if**  $iS < mirror.length$  **then**  
 $idStudent \leftarrow mirror[iS][0]$   
**end if**  
 $denTheta \leftarrow 0.0f$   
 $numTheta \leftarrow 0.0f$   
 $erro \leftarrow 0.0f$   
**while**  $iS < mirror.length$



## 7.3 Design

In this section, we present the study design to integrate adaptive selection into the instrument to improve the measurement of cognitive programming skills.

### 7.3.1 Dependent and Independent Variables

One of the requirements for having an instrument managed by adaptive algorithms is the existence of a calibrated items bank. These algorithms have:

- *Estimation Method;*
- *Adaptive Selection Method;*
- *Item Exposure Control Method;*
- *Instrument Stop Method.*

We use these measures as **Independent Variables**. In such a way, these measures allowed the elaboration of different algorithms. Then, we conducted simulations with the calibrated items bank and the answers provided by the subjects in the pre-test and performed the *subjects' abilities estimation* (**Dependent Variable**) based on these algorithms.

### 7.3.2 Data Analysis

We simulated the data of this phase through the answers provided by the subjects in the pre-test. Based on these answers and the item bank calibration, we adapted the participants' answers through a script in Java language (Appendix N). We analyzed these data based on descriptive and inferential statistics, aiming to meet the proposed objectives of this study regarding the integration of adaptive selection through algorithms that improve performance in estimating students' abilities.

### 7.3.3 Methodological Process

Figure 7.3 represents the methodological process of analyzing the integration of adaptive selection to the instrument.

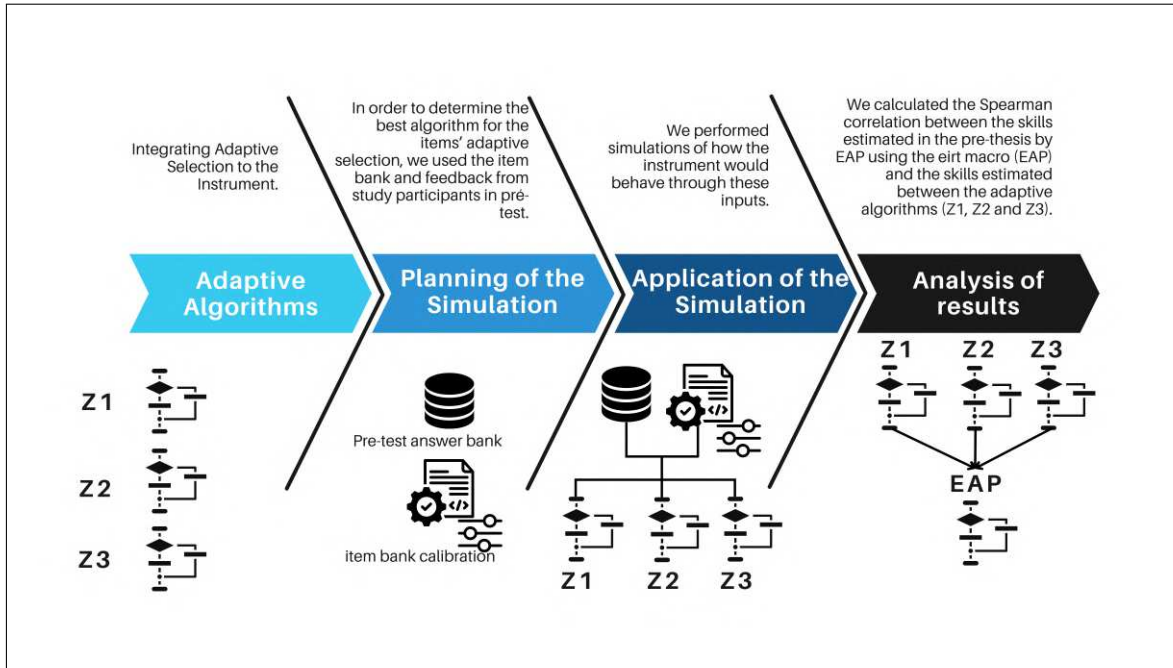


Figure 7.3: Methodological Process of Analyzing the Integration of Adaptive Selection to the Instrument.

In order to determine the best algorithm for the items' adaptive selection, we used the item bank and feedback from study participants in the pré-test. We performed simulations of how the instrument would behave through these inputs. We calculated the Spearman correlation between the skills estimated in the pre-thesis by EAP using the eirt macro (EAP) and the skills estimated between the adaptive algorithms (Z1, Z2, and Z3).

### 7.3.4 Threats to Validity

The main idea in implementing the algorithms is to perform a computerized adaptive test with the exact specifications (and the same validity) as a standard pen-paper test and still provide a smaller number of items. In the simulations, we used the responses provided in the pre-test stage (Chapter 6). In this way, we must consider the same threats to the validity of the previous step. We use the calibrated item bank and response database to estimate the algorithms' abilities, and the ability estimates may not exactly match reality.

## 7.4 Results

We estimated the students' ability in the EAP pre-test using the *eirt macro* (**EAP**), as well as the estimate generated by the three algorithms: **Z1**, **Z2** and **Z3**. The skill estimates generated by the algorithms are quantitative variables without causality; one variable does not influence the other. In Table 7.2, we present the descriptive statistics (mean and standard deviation (*Sd*)) of the ability estimation between the algorithms. As we can see from the data, the algorithms generated different estimates for each observed scenario.

Skill	EAP		Z1		Z2		Z3	
	Mean	<i>Sd</i>	Mean	<i>Sd</i>	Mean	<i>Sd</i>	Mean	<i>Sd</i>
Recognizing	-0.267	0.901	-0.248	2.079	-0.304	1.878	-0.499	1.793
Recalling	-0.090	0.943	-0.021	1.850	-0.094	1.890	-0.217	1.808
Interpreting	-0.032	0.930	-0.190	1.779	-0.041	1.805	-0.100	1.772
Exemplifying	-0.043	0.969	-0.074	1.747	0.053	1.886	-0.050	1.810
Classifying	-0.063	0.983	-0.176	1.874	-0.095	1.845	-0.233	1.751
Summarizing	-0.066	0.978	-0.050	1.869	-0.126	1.793	-0.275	1.765
Inferring	-0.113	0.947	-0.104	1.873	-0.090	1.967	-0.333	1.871
Comparing	-0.119	0.980	-0.188	1.883	-0.034	1.927	-0.118	1.882
Explaining	-0.092	0.955	-0.137	1.727	0.031	1.858	-0.204	1.804
Executing	-0.145	0.966	-0.186	1.770	-0.130	1.651	-0.201	1.663
Implementing	-0.090	0.972	-0.101	1.777	-0.179	1.816	-0.360	1.743
Differentiating	0.003	0.954	0.024	1.907	0.049	1.918	0.024	1.923
Organizing	-0.060	0.978	-0.018	1.863	-0.080	1.857	-0.237	1.734
Attributing	-0.056	0.980	-0.019	1.883	-0.056	1.796	-0.193	1.705
Checking	-0.023	0.997	0.067	1.962	-0.036	1.805	-0.181	1.728
Critiquing	-0.076	0.965	0.042	1.939	-0.087	1.854	-0.178	1.740
Generating	-0.067	0.975	0.029	2.027	-0.068	1.731	-0.144	1.644
Planning	0.026	0.994	0.079	2.009	0.218	1.916	0.235	1.896
Producing	0.027	1.002	0.073	1.978	0.351	1.819	0.365	1.776

Table 7.2: Descriptive Statistics of Abilities' Estimation Among Algorithms.

To determine which of the algorithms (**Z1**, **Z2** and **Z3**) has the best estimate when compared to **EAP**, we correlate the variables. For this, first, we calculate the normal distribution of the data through the *shapiro.test()* function present in the *R* language. In Table 7.3, we present the results of the normality test of the ability estimates between the algorithms.

<b>Skill</b>	<b>EAP</b>	<b>Z1</b>	<b>Z2</b>	<b>Z3</b>
Recognizing	0.052	<0.001	<0.001	<0.001
Recalling	0.233	<0.001	<0.001	<0.001
Interpreting	0.167	0.002	0.002	0.004
Exemplifying	0.163	0.002	<0.001	0.001
Classifying	0.030	<0.001	<0.001	0.002
Summarizing	0.070	<0.001	0.002	0.002
Inferring	0.104	<0.001	<0.001	<0.001
Comparing	0.014	<0.001	<0.001	<0.001
Explaining	0.141	0.002	<0.001	0.001
Executing	0.067	0.003	0.023	0.001
Implementing	0.146	0.002	0.001	0.002
Differentiating	0.454	<0.001	<0.001	<0.001
Organizing	0.050	<0.001	<0.001	0.002
Attributing	0.054	<0.001	<0.001	0.003
Checking	0.098	<0.001	<0.001	0.001
Critiquing	0.056	<0.001	<0.001	0.002
Generating	0.070	<0.001	0.001	0.015
Planning	<0.001	<0.001	<0.001	<0.001
Producing	<0.001	<0.001	<0.001	<0.001

Table 7.3: Normality Test of Skills Estimates Between Algorithms.

According to the results of Table 7.3, all skill estimates by the algorithms do not follow a normal distribution, as the  $p$  – values were less than 0.05. In this way, we calculated the Spearman correlation between the skills estimated in the pre-thesis by EAP using the eirt macro (**EAP**) and the skills estimated between the adaptive algorithms (**Z1**, **Z2** and **Z3**). To calculate the Spearman correlation, we use the `cor.test()` function present in the *R* language, passing the estimates to the parameters  $x$  and  $y$ , and `spearman` for the `method` parameter. In Table 7.4, we present these results.

	<b>Skill</b>	<b>Z1</b>	<b>Z2</b>	<b>Z3</b>
<b>EAP</b>	Recognizing	0.605	0.634	0.649
	Recalling	0.634	0.752	0.781
	Interpreting	0.622	0.725	0.738
	Exemplifying	0.663	0.804	0.824
	Classifying	0.710	0.693	0.748
	Summarizing	0.665	0.713	0.759
	Inferring	0.651	0.763	0.783
	Comparing	0.671	0.792	0.800
	Explaining	0.674	0.778	0.827
	Executing	0.670	0.784	0.826
	Implementing	0.685	0.737	0.790
	Differentiating	0.701	0.753	0.744
	Organizing	0.650	0.705	0.731
	Attributing	0.647	0.716	0.724
	Checking	0.673	0.698	0.724
	Critiquing	0.655	0.710	0.723
Generating	0.662	0.770	0.797	
Planning	0.622	0.726	0.735	

Table 7.4: Correlation of Skills Estimation Between Eap and Adaptive Algorithms.

## 7.5 Discussions

As we can observe in the results of Table 7.4, all algorithms present moderate or strong correlations to estimate the individuals' abilities. However, among the correlations, the **Z3** algorithm was the one that obtained the best estimate for all skills. Based on our results, the algorithm that best selects items and estimates abilities more accurately to the full test has the following characteristics:

- **Estimation method:** maximum likelihood;
- **Adaptive Selection Method:** maximum information from Fisher;
- **Item Exposure Control Method:** item exposure control;
- **Instrument Stop Method:** standard error in estimation  $<0.01$  and the minimum and maximum quantity of administered items.

## 7.6 Related Works

Assessment is one of the most critical issues in the learning process. In many cases, it defines the instruction sequence because it measures the student's performance in the educational process. In recent decades, information technology inclusion in the teaching-learning process has caused meeting the characteristics of the diversity of students and teachers. Learning technologies make it possible to adapt different learning and teaching methods in the educational context through user adaptation and modeling [128].

Because testing consumes time that might otherwise be devoted to instruction, it is important to design efficient tests. Doing so requires a careful balance of contributions from technology, psychometrics, test design, and learning sciences. Educational research and student testing use computer-adaptive testing [106].

One study offered suggestions for implementing a CAT in practice, providing an overview of how it works. The study aimed to motivate researchers and professionals to conduct tests and evaluations in this little-practiced area. The works above help them understand some relevant technical concepts [25, 106].

A study described the IRT and how it can be applied in an online course test scenario to generate adaptive assessments. This application took place within an Introduction to Object-Oriented Programming course based on items (evaluation questions). The study outlines the variables to consider and how to calculate the probability of a student correctly answering a specific item based on an ability test and their past responses, duly normalized in the adaptation process. The main contributions of this study are the probabilistic theory's implementation in the generation of adaptive evaluation. Whereas the used distributed repositories allow properly parameterized items' reuse [128].

In this chapter, we integrate adaptive selection into the instrument to improve the measurement of cognitive programming skills. We follow the IRT and CTT guidelines for the development of algorithms [101]. Thus, to understand some specific concepts, we follow the suggestions given by the works of Costa [25] and Sadeghi [106]. Finally, we determined the best algorithm for the adaptive selection of items through simulations with the database from the study of Chapter 6.

## 7.7 Final Considerations

This thesis started with the need to integrate adaptive selection into an instrument to measure cognitive programming skills to improve the system's efficiency in selecting items and estimating students' skills. To better understand whether it is possible to improve the application of an instrument adaptively, we selected items according to the subject's skill level and administered fewer items than conventional. As a result, we conclude that:

- **(RQ10:)** among the algorithms developed to select items adaptively, the algorithm that obtained the best accuracy in the skill estimation process has the following characteristics: i) maximum likelihood as a skill estimation method; ii) maximum fisher information as an adaptive selection method; iii) obtain an item exposure control; iv) and for the method of stopping the instrument, a combination of factors such as the standard error in the estimation  $<0.01$  and the minimum and a maximum number of administered items.

# Chapter 8

## Assessing the Instrument Validity

In this Chapter, we present an exploratory investigation into the instrument's validity for measuring introductory programming skills.

### 8.1 Initial Considerations

As reported in previous chapters, one way to gradually reduce the cognitive load of novices to programming is to build cognitive skills in CS1. Skills can be fostered by following the six cognitive domains of Bloom's Revised Taxonomy.

Therefore, (i) we adapted Bloom's Revised Taxonomy to programming teaching and empirically evaluated this adaptation (Chapter 4); (ii) according to this adaptation, we developed an instrument to measure and foster cognitive programming skills and analyzed the items' semantics and content (Chapter 5); (iii) we evaluated the items' psychometric properties and the instrument's internal consistency; we evaluated the correlation of cognitive programming skills with code writing (producing) (Chapter 6); finally, we integrate adaptive selection into the instrument through an algorithm that improves item selection and the estimation of students' ability (Chapter 7). In this Chapter, we aim to investigate whether fostering these skills improves the novices' performance in writing code. Therefore, this Chapter aims to:

- **SO10.** To investigate whether fostering cognitive programming skills improves novice code-writing performance.

We will answer the following research question:

- **RQ11.** Does the instrument that fosters cognitive programming skills improve the code-writing performance of novices?

Thus, we have the following hypotheses:

- **H1-0:** Participants who had their skills fostered by the instrument had **lower** estimates of writing code when compared to participants using the traditional method.
- **H1-1:** Participants who had their skills fostered by the instrument had **higher** estimates of writing code when compared to participants using the traditional method.

To investigate whether practicing a set of cognitive skills improves students' performance in writing code, we applied an experiment to a group of 50 individuals who participated in a CS1 course at UEPB. We provide the participants with study material, instructions, practical problems, and solutions based on different experimental conditions. We then assessed the participants to investigate their learning performance. We estimate the individuals' abilities to verify whether the instrument can distinguish the groups.

We organize the remainder of this Chapter as follows. In Section 8.2, we describe the study design. In Section 8.3, we present the results. In Section 8.4, we discuss the results. In Section 8.5, we discuss the related works. Finally, In Section 8.6, we conclude the Chapter with final remarks.

## 8.2 Design

This section presents the experiment's design to verify whether the support provided by the instrument can positively impact the student's performance in programming.

### 8.2.1 Participants

We selected 50 novices in CS from UEPB to participate in this study. We also allocated the participants into two study groups called the experimental group and the control group. In this step, we consider the same inclusion criteria described in subsection 6.3.1.



We divided the experiment participants according to a subjective analysis of trust in Python (Appendix O). To avoid errors in this division—a group that receives participants with excellent performance and others that do not—we applied a survey before the study. We adopted a seven-point Likert scale assessment model in the survey.

We represented the answer to each question as follows: 1 (not at all confident) to 7 (completely confident). We calculated the response average for each subject based on the scores assigned. We order and separate the groups as we present the distribution of the measures of the groups as depicted in Figure 8.1.

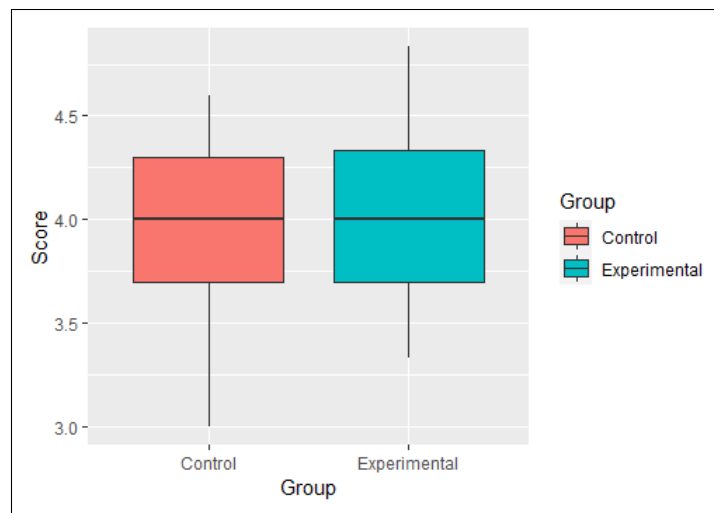


Figure 8.1: Boxplot Between Groups of How Many Users Master the Python Language.

Then, we test the data normality and obtain the p-value = 0.7144 for the control group and p-value = 0.8455 for the experimental group. Thus, we can conclude that the samples follow a normal distribution with a confidence level of 95%.

As the data follow a normal distribution and the samples are independent, we apply a parametric test called *t-test*. This test compares two means and shows whether their differences are significant. This test allows us to assess whether these differences occurred by chance. We present the output of this test in Table 8.1.

Null hypothesis	Control Group		Experimental Group		P-value
	Mean	Sd	Mean	Sd	
There is significance in the difference between groups	3.968	0.413	4.020	0.399	0.6531

Table 8.1: Cross-Group Analysis of How Many Users Master the Python Language.

The test shows an insignificant difference between the treatment and control groups. As a null hypothesis, the confidence level in Python in the experimental group is the same compared to the control group. In the pre-research, the *p-value* of 0.6531 shows that we distributed the individuals evenly among the groups, that is, without having more qualified individuals in one group compared to the other.

### 8.2.2 Dependent and Independent Variables

In research, variables are characteristics that can take on different values, such as height, age, species, or exam grade. In scientific research, we often aim to study the effect of one variable on another. For example, a study investigates whether students who spend more time studying can score better on exams.

The variables in a cause-and-effect relationship study are called independent and dependent variables. The **Independent Variable** is the *cause*, and its value does not depend on other variables in the study. The **Dependent Variable** is the *effect*, and its value depends on changes in the independent variable. Next, we will detail these variables in our experiment.

The Independent Variables used in our experiment are the abilities: **recognizing, recalling, interpreting, exemplifying, classifying, summarizing, inferring, comparing, explaining, executing, implementing, differentiating, organizing, attributing, checking, critiquing, generating** and **planning**. The Dependent Variable is the estimate of the **producing** skill.

### 8.2.3 Data Analysis

Data collected through psychometric tests were initially analyzed, as indicated in the instrument application manuals. Then, all collected data were analyzed according to inferential descriptive statistics, aiming to meet the proposed objectives of the experiment.

### 8.2.4 Methodological Process

Figure 8.2 represents the methodological process of assessing the Instrument Validity.

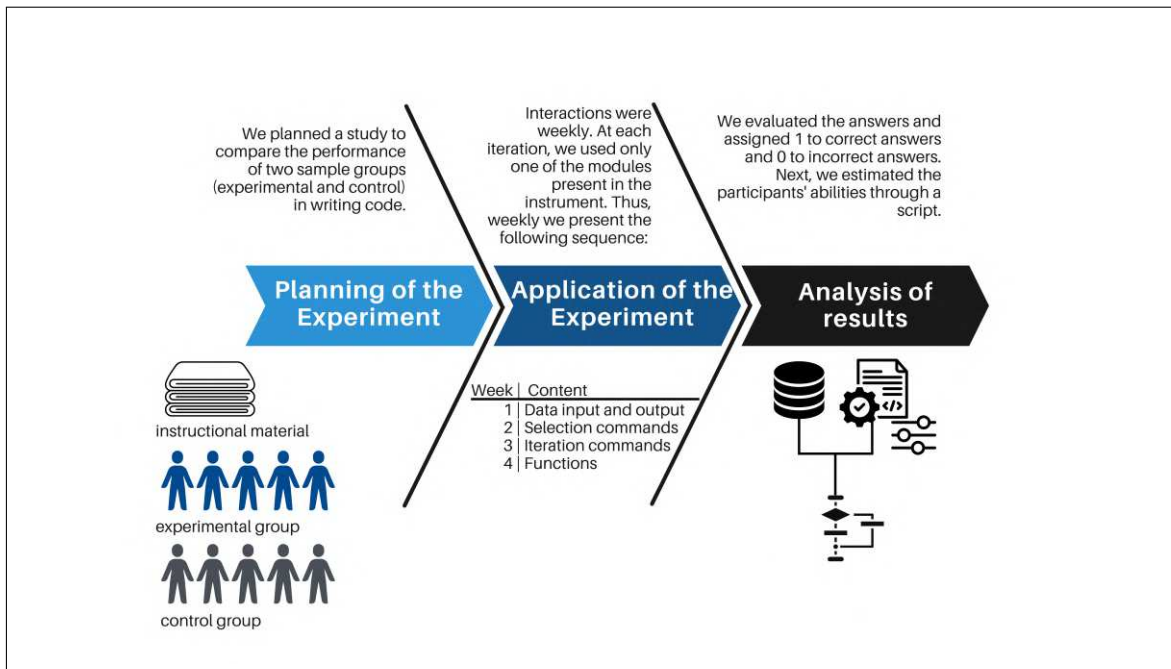


Figure 8.2: Methodological Process of Assessing the Instrument Validity.

We conducted the study simultaneously in two classrooms, with participants separated by condition (control and experimental). We explained the purpose of the research to the participants who signed the consent form to participate in the study.

We conducted a pre-research with questions about how much they mastered the Python language. We separated the participants as described in the subsection 8.2.1. It should be noted that we did not exclude students who already knew how to program or had previous experience. However, most students had their first contact with programming through this study.

During the 4-hour instructional time, we provided both groups with study material, instructions, practical problems, and solutions. In addition, unlike the control group, we fostered, through the instrument, the cognitive abilities of the participants in the experimental group (*independent variables*), except for the producing ability. Participants could pause and ask questions only related to the material's content. We answered practice questions after we completed the study.

Interactions were weekly. At each iteration, we used only one of the modules present in the instrument. Thus, weekly we present the following sequence: input and output data, selection commands, iteration commands, and functions. The participants worked through

the content at their own pace. We encouraged them to work sequentially through the material and try to solve problems before looking for solutions. Participants could pause and ask questions. We answered all questions related to the content of the material. Finally, after the study's conclusion, we answered all doubts related to the questions of the evaluated instrument.

In the end, we verified if the actions contained in the instrument positively impacted the final performance of the experimental group. Both groups had to produce (code-writing) skills estimated by a 10-item assessment test<sup>1</sup>. Participants took up to two hours to complete it to measure how much they learned.

In total, participants answered 40 items. We evaluated the answers and assigned 1 to correct answers and 0 to incorrect answers. Next, we estimated the participants' abilities through a script (Chapter 7). This script simulated the participants' abilities if they had responded to an online tool. That is, selecting items according to the student's skill level.

### 8.2.5 Validity Threats

We consider some factors that generated threats and directly influenced the conclusions of this last phase. Among these threats, we have:

- Problems related to incorrect interpretation of questions;
- Survey participants may feel intimidated or uncomfortable when taking the tests. To mitigate these threats, we applied the guidelines of the Ethics Committee for Research with Human Beings of UFCG and UEPB, which approved this research (Protocols: 23933919.4.0000.5182 | 23933919.4.3001.5187). Only participants who signed the free and ICF participated in this study;
- Part of the answers given to the instrument was manually corrected. Human errors can happen in corrections. However, part of the answers given by the participants involved computerized applications and corrections to reduce possible errors;

---

<sup>1</sup>These are the ten items developed in the previous phase for each module 6. We did not present these items in the intervention, only in applying the final performance test.

- Like all empirical research, this work has threats to validity. The number of subjects participating in the study does not allow the generalization of results.

## 8.3 Results

We analyzed the performance of the groups in the evaluation test. We aimed to investigate whether promoting the skills of recognizing, recalling, interpreting, exemplifying, classifying, summarizing, inferring, comparing, explaining, executing, implementing, differentiating, organizing, attributing, checking, critiquing, generating, and planning improves the performance of novices when writing code.

To answer this question, we estimated the participants' abilities in both groups who answered the producing item bank. Figure 8.3 provides a data summary. Despite some discrepant values, the experimental group showed less variation when compared to the control group. In general, many individuals in the experimental group had significantly higher ability estimates when compared to the control group.

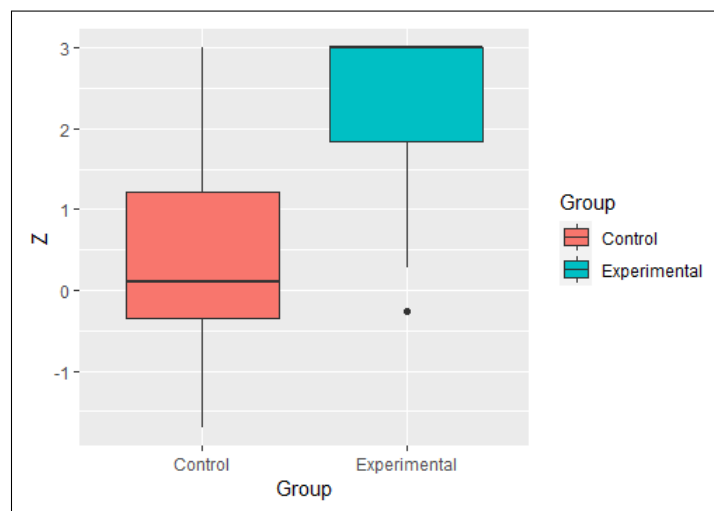


Figure 8.3: Boxplot of Groups' Performance in the Ability to Produce Codes.

Table 8.2 shows the mean and standard deviation of the performance of both groups in the assessment, respectively. The mean estimate of the participant's skills in the assessment is higher for the experimental group. We checked the distribution of our data set. We applied the Shapiro-Wilk test, with a significance level of 95%, to observe whether the data set follows a normal distribution. The control group had a p-value of 0.680. The experimental

group p-value <0.001. Since not all variables follow a normal data distribution, we used the Mann-Whitney test to compare the estimated ability of the participants in both groups who responded to the assessment, as shown in Table 8.2.

Null hypothesis	Control Group		Experimental Group		P-value
	Median	<i>Sd</i>	Median	<i>Sd</i>	
Participants who had their skills fostered by the instrument had lower estimates of writing code when compared to participants using the traditional method.	0.108	1.228	3	1.105	<0.001

Table 8.2: Analysis of the Group's Performance in the Ability to Produce Codes.

We consider as a null hypothesis that the estimate for each skill of the experimental group is lower when compared to the control group for the performance in the assessment. Based on the p-value of Table 8.2, which was <0.001, we can conclude that the performance of participants who practiced the skills of recognizing, recalling, interpreting, exemplifying, classifying, summarizing, inferring, comparing, explaining, executing, implementing, differentiating, organizing, attributing, checking, critiquing, generating, and planning performed better in producing skill when compared to participants in the control group.

## 8.4 Discussions

With our proposal to adapt the cognitive domain of Bloom's Revised Taxonomy for programming teaching, we developed a valid and adaptable instrument to measure and foster cognitive skills in CS1. We evaluated the potential learning outcomes of this instrument based on this experiment in such a way that the contributions of this research improve the instructional design in CS1.

Results suggest that sequential practice of each of the cognitive programming skills fostered by the instrument can improve code writing and instruction in CS1. We found that experimental group participants who received practice in all skills had a better estimate for producing code, providing better solutions to problems. These results support our proposed adaptation of the cognitive domain of Bloom's Revised Taxonomy. Furthermore, these

results show that sequential instructions in CS1 help novices learn to program compared to traditional teaching instructions.

The imbalance of the cognitive stimuli in the experimental conditions caused the difference in the estimates of skills by the groups. While the control group only reached stimuli from the traditional learning material, the experimental group reached stimuli through the instrument that fostered cognitive skills in sequential programming.

## 8.5 Related Works

A preliminary study investigated the validity of instruments to measure cognitive programming skills [110]. This study provided participants with instructional material, practical problems, and solutions based on different experimental conditions. The study used CTT and IRT to estimate individuals' abilities to verify whether the instruments can distinguish groups. The authors investigated whether explicit enunciation and the applicable provision of the ability to read semantics and reading models improve students' performance in semantics and writing models. The results suggest that participants who practiced code reading outperformed in problem-solving compared to participants in the control group.

Another study presented results of an intervention strategy aimed at improving students' program understanding skills [86]. The authors used the automatic generation of items to generate a set of individualized practical exercises for the needs of each student. Results from pre- and post-assessments for the intervention group and a control group show the benefits of taking time to understand the program.

A moderating effect on programming ability between the level of familiarity with programming and abstract thinking has also been reported in recent studies [97] [98]. It is noticed that when students have a high level of abstract thinking skills, their programming skills improve much more than those with a low level of abstract thinking skills. This pattern of behavior is observable with the application of Screening Programming and the Programming Aptitude Test. In programming questions involving a high level of students' abstract thinking, students who performed poorly in abstract thinking also failed these items [50].

In this chapter, we present the results of the validity of an instrument to measure

cognitive programming skills. We provide participants with study material, instructions, practice problems, and solutions based on different experimental conditions. We then assessed participants to investigate their learning performance. We estimated the abilities of individuals to verify whether the instrument could distinguish between groups. We conclude that participants who practiced programming using the instrument had better performance in problem-solving when compared to participants in the control group.

## 8.6 Final Considerations

In this Chapter, we present a study to compare the performance of two sample groups (experimental and control) in writing code. We estimated the participants' ability to write codes through the experimental conditions in both groups. As a result, we conclude that:

- **(RQ11:)** Participants who practiced programming using the instrument had better performance in code production when compared to participants in the control group.

This result reinforces the claim that students should learn other cognitive skills before learning to write code. Through this study, we present ways to foster or measure preliminary skills in a CS1. Finally, this result supports our theoretical assumptions that the cognitive levels of Bloom's Revised Taxonomy, through its measurement instrument, can sequence the learning process in CS1.



# Chapter 9

## Conclusions and Future Work

In this Chapter, we present this thesis' final considerations, limitations of this research, and future work.

### 9.1 Conclusions

Teaching CS1 is a complex task, and several researchers point out problems related to students' difficulties in understanding programming concepts and the motivation to perform the problem-solving activity correctly. Despite the advances made by researchers in teaching and learning CS1 courses, dropout and failure rates are still high.

At the same time, recent studies indicate that it should foster cognitive programming skills during CS1. Some studies have found that commenting, debugging, and understanding code have significantly improved students' ability to write code and solve problems, decreasing dropout and failure rates. However, current introductory instructions need to identify, structure, and sequence the cognitive skills involved in programming.

Despite advances in the literature, we need studies that guide the instruction of educators to work with these skills and at what level they should be encouraged. Therefore, our main objective in this doctoral research was to investigate the cognitive performance of novice programmers in code writing tasks. For that, we did the following:

- We performed a systematic literature review to describe cognitive programming skills and the approaches/instruments that measure/foster such skills. We concluded that **(RQ1.)** cognitive programming skills are tracing, explaining, comprehension, reading,

- debugging, modifying, and writing. **(RQ2.)** Researchers use CTT to measure cognitive programming skills. However, some universities have adopted other theories and taxonomies for this practice, such as IRT, SOLO Taxonomy, and Bloom's Revised Taxonomy. **(RQ3.)** Researchers use different approaches to programming teaching based on different educational theories, teaching structures, or educational approaches.
- We presented **(RQ4.)** a proposal for adapting the cognitive domain of Bloom's Revised Taxonomy to programming teaching and the cognitive skills' association raised in the literature within the adapted taxonomy. We empirically analyze this proposal through a survey applied to a group of specialists in CS1. We concluded that the judges' responses achieved high agreement regarding the adaption/association of the cognitive level skill definition of Bloom's Revised Taxonomy for programming teaching. High consistency and ICC test also indicated excellent reliability between the scores assigned by the judges.
  - We reported the instrument's development based on the adaptation of the cognitive domain of Bloom's Revised Taxonomy to programming teaching. Then, we analyzed the items' semantics and content present in this instrument. We concluded that **(RQ5)** items have appropriate content analysis and **(RQ6)** items have appropriate semantics;
  - We analyzed the item's psychometric properties present in the instrument and their internal consistency through Measurement Theories. We correlated cognitive programming skills (present in the instrument) and the participant's ability to produce code. We concluded **(RQ7)** that the 750 items present in the instrument are well-calibrated according to the slope, threshold, and asymptote parameters. **(RQ8)** The instrument for each skill had a reliable internal consistency, with Cronbach's alpha ranging between 0.8 and 1. **(RQ9)** All the cognitive skills investigated in the instrument presented strong correlations with the ability to produce code. The data suggested that this correlation increases when the student advances in the cognitive levels within Bloom's Revised Taxonomy adaptation to programming teaching, which indicates that learning occurs linearly.
  - We presented the adaptive selection integration into assessment instruments to improve

the measurement of cognitive programming skills. We concluded (**RQ10.**) among the algorithms developed to select items adaptively, the algorithm that obtained the best accuracy in the skill estimation process has the following characteristics: i) maximum likelihood as a skill estimation method; ii) maximum fisher information as an adaptive selection method; iii) obtain an item exposure control; iv) and for the method of stopping the instrument, a combination of factors such as the standard error in the estimation  $<0.01$  and the minimum and a maximum number of administered items.

- Finally, we presented an exploratory investigation into the instrument's validity to measure introductory programming skills. We concluded that (**RQ11.**) the participants who practiced programming using the instrument performed better in code production when compared to participants in the control group.

## 9.2 Future Work

Among the possibilities for future work, the following stand out:

- To obtain other algorithm variations with item selection and skill estimation criteria. Algorithms may have different estimation methods, item selection, exposure control, and stopping criteria. Measurement Theories address these methods, their equation models, and how they work during item selection and skill estimation [14, 25]. Next, to check which of these algorithms correlates best with the skill estimates in the pretest. The results of this research are satisfactory, but there are areas for improvement so that other researchers can replicate the design of Chapter 7.
- To conduct experiments to investigate whether skills cannot be sequenced. Alternatively, even which one would have more impact on the student's ability to program being fostered separately. To achieve this goal, researchers can adapt the design of the study we describe in Chapter 8;
- To develop the instrument in its online version for wide application. Also, develop more items to expand the item bank and cover more latent trait regions. We describe these sub-tasks in the subsections 2.4.6 and 2.4.8, respectively;

- 
- To adapt the affective and psychomotor domain of Bloom's Revised Taxonomy for teaching programming. Moreover, produce instruments that foster and measure these skills. To achieve this goal, researchers can adapt the design of the studies we describe in Chapters 4 5 and 6.
  - Standardize the instruments to normalize the procedures for their use based on the precautions taken in their application. Evaluate uniformity of testing conditions, group control, and standardized instructions, and motivate examinees by reducing anxiety. Develop parameters or criteria to interpret the results obtained. This phase includes mathematical models to obtain norms for interpreting the scores of an instrument [14, 101].

# Bibliography

- [1] ACM. Acm digital library. <https://dl.acm.org/>. Accessed: 2019-08-16.
- [2] B. S. Alqadi and J. I. Maletic. An empirical study of debugging patterns among novices programmers. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Seattle, WA, United States, 2017.
- [3] I. Alshaye, Z. Tasir, and N. F. Jumaat. The conceptual framework of online problem-based learning towards problem-solving ability and programming skills. In *Proceedings of the Conference on e-Learning, e-Management and e-Services (IC3e)*. IEEE, Pulau Pinang, Malaysia, 2019.
- [4] A. P. Ambrósio, F. M. Costa, L. Almeida, A. Franco, and J. Macedo. Identifying cognitive abilities to improve cs1 outcome. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, Rapid City, SD, United States, 2011.
- [5] L. W Anderson and D. R. Krathwohl. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of educational objectives*. Longman, 2001.
- [6] J. M. Andrade, J. A. Laros, and V. V. Gouveia. O uso da teoria de resposta ao item em avaliações educacionais: Diretrizes para pesquisadores. *Avaliação Psicológica*, 9(3), 2010.
- [7] A. L. S. O. Araújo, J. S. Santos, W. L. Andrade, D. D. S. Guerrero, and V. Dagienè. Exploring computational thinking assessment in introductory programming courses. In *Proceedings of the IEEE Frontiers in Education Conference (FIE)*. IEEE, 2017.

- 
- [8] A. L. S. O. Araújo, J. S. Santos, M. R. A. Melo, W. L. Andrade, D. D. S. Guerreiro, and J. C. A. Figueiredo. *Metodologia de Pesquisa em Informática na Educação: Abordagem Quantitativa de Pesquisa*, chapter Teoria de Resposta ao Item. SBC, Porto Alegre, 2019.
- [9] P. Ardimento, M. L. Bernardi, M. Cimitile, and G. Ruvo. Reusing bugged source code to support novice programmers in debugging tasks. *ACM Transactions on Computing Education (TOCE)*, 20(1), 2019.
- [10] V. C. O. Aureliano, P. C. de A. R. Tedesco, and L. M. M. Giraffa. Desafios e oportunidades aos processos de ensino e de aprendizagem de programação para iniciantes. In *Congresso da Sociedade Brasileira de Computação*, volume 24, pages 2066–2075, 2016.
- [11] M. Azadmanesh and M. Hauswirth. Concept-driven generation of intuitive explanations of program execution for a visual tutor. In *Proceedings of the Working Conference on Software Visualization (VISSOFT)*. IEEE, Shanghai, China, 2017.
- [12] F. B. Baker. *The Basics of Item Response Theory*. ERIC, 2001.
- [13] F. B. Baker and S. Kim. *Item Response Theory: Parameter Estimation Techniques*. CRC Press, 2004.
- [14] F. B. Baker and S. Kim. *The Basics of Item Response Theory using R*. Springer, 2017.
- [15] R. M. Bekman. Aplicação dos blocos incompletos balanceados na teoria de resposta ao item. *Estudos em Avaliação Educacional*, 24:119–136, 2001.
- [16] G. R. Bergersen, D. I. Sjøberg, and T. Dybå. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering*, 40(12), 2014.
- [17] M. Berges and P. Hubwieser. Evaluation of source code with item response theory. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2015.

- [18] J. B. Biggs and K. F. Collis. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 2014.
- [19] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. IEEE, Florence, Italy, 2015.
- [20] T. Busjahn and C. Schulte. The use of code reading in teaching programming. In *Proceedings of the Koli Calling International Conference on Computing Education Research*. ACM Koli, Finland, 2013.
- [21] E. Carter and G. D. Blank. A tutoring system for debugging: Status report. *Journal of Computing Sciences in Colleges (JCSC)*, 28(3), 2013.
- [22] Charles C Chan, MS Tsui, Mandy YC Chan, and Joe H Hong. Applying the structure of the observed learning outcomes (solo) taxonomy on student’s learning outcomes: An empirical study. *Assessment & Evaluation in Higher Education*, 27(6):511–527, 2002.
- [23] M. W. Chen, C. C. Wu, and Y. T. Lin. Novices’ debugging behaviors in vb programming. In *Proceedings of the Learning and Teaching in Computing and Engineering (LaTiCE)*. IEEE, Macau, China, 2013.
- [24] A. Clear, A. Parrish, J. Impagliazzo, P. Wang, P. Ciancarini, E. Cuadros-Vargas, S. Frezza, J. Gal-Ezer, A. Pears, S. Takada, et al. Computing curricula 2020 (cc2020) paradigms for global computing education. *ACM: New York, NY, USA*, 2020.
- [25] D. R. Costa. *Métodos Estatísticos em Testes Adaptativos Informatizados*. PhD thesis, Master’s thesis, UFRJ-Universidade Federal do Rio de Janeiro., 2009.
- [26] Richard Cox, Steven Bird, and Bernd Meyer. Teaching computer science in the victorian certificate of education: A pilot study. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 135–140, 2017.

- [27] D. Cukierman and D. M. Thompson. Learning strategies sessions within the classroom in computing science university courses. In *Proceedings of the 12th annual SIGCSE conference on Innovation and Technology in Computer Science Education*, pages 341–341, 2007.
- [28] Q. Cutts, M. Barr, M. B. Ada, P. Donaldson, S. Draper, J. Parkinson, J. Singer, and L. Sundin. Experience report: Thinkathon – countering an got it working mentality with pencil-and-paper exercises. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Aberdeen, Scotland, United Kingdom, 2019.
- [29] P. Denny, A. Luxton-Reily, and D. Carpenter. Enhancing syntax error messages appears ineffectual. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Uppsala, Sweden, 2014.
- [30] P. Donaldson and Q. Cutts. Flexible low-cost activities to develop novice code comprehension skills in schools. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE)*. ACM Potsdam, Germany, 2018.
- [31] M. Dorodchi, N. Dehbozorgi, and T. K Frevert. “i wish i could rank my exam’s challenge level!”: An algorithm of bloom’s taxonomy in teaching cs1. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, Indianapolis, IN, United States, 2017.
- [32] R. S. Duran, J. Rybicki, A. Hellas, and S. Suoranta. Towards a common instrument for measuring prior programming knowledge. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Aberdeen, Scotland, United Kingdom, 2019.
- [33] S. Elnaffar. Using software metrics to predict the difficulty of code writing questions. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)*. IEEE, Dubai, United Arab Emirates, 2016.
- [34] S. E. Embretson and S. P. Reise. *Item Response Theory*. Psychology Press, 2013.



- [35] K. L. Eranki and K. M. Moudgalya. Application of program slicing technique to improve novice programming competency in spoken tutorial workshops. In *Proceedings of the International Conference on Technology for Education (T4E)*. IEEE, Clappana, India, 2013.
- [36] K. L. Eranki and K. M. Moudgalya. An integrated approach to build programming competencies through spoken tutorial workshops. In *Proceedings of the International Conference on Technology for Education (T4E)*. IEEE, Kharagpur, India, 2013.
- [37] K. L. N. Eranki and K. M. Moudgalya. Evaluation of programming competency using student error patterns. In *Proceedings of the International Conference on Learning and Teaching in Computing and Engineering*. IEEE, 2015.
- [38] K. L. N. Eranki and K. M. Moudgalya. Program slicing technique: A novel approach to improve programming skills in novice learners. In *Proceedings of the Conference on Information Technology Education (SIGITE)*. ACM Boston, MA, United States, 2016.
- [39] M. Erguven. Two approaches to psychometric process: Classical test theory and item response theory. *Journal of Education*, 2(2):23–30, 2013.
- [40] S. Esper, S. R. Foster, W. G. Griswold, C. Herrera, and W. Snyder. Codespells: Bridging educational language features with industry-standard languages. In *Proceedings of the Koli Calling International Conference on Computing Education Research*. ACM Koli, Finland, 2014.
- [41] G. V. F. Fabic, A. Mitrovic, and K. Neshatian. Adaptive problem selection in a mobile python tutor. In *Proceedings of the Conference on User Modeling, Adaptation and Personalization (UMAP)*. ACM, Singapore, Singapore, 2018.
- [42] G. V. F. Fabic, A. Mitrovic, and K. Neshatian. Investigating the effects of learning activities in a mobile python tutor for targeting multiple coding skills. *Research and Practice in Technology Enhanced Learning*, 13(1), 2018.
- [43] A. P. do C. M. Ferraz and R. V. Belhot. Taxonomia de bloom: Revisão teórica e

- apresentação das adequações do instrumento para definição de objetivos instrucionais. *Gestão & Produção*, 17(2):421–431, 2010.
- [44] A. Fink, S. Born, C. Spoden, and A. Frey. A continuous calibration strategy for computerized adaptive testing. *Psychological Test and Assessment Modeling*, 60(3):327–346, 2018.
- [45] J. P. Fox and C. A. W. Glas. Bayesian estimation of a multilevel irt model using gibbs sampling. *Psychometrika*, 66(2), 2001.
- [46] S. Frezza, M. Daniels, A. Pears, A. Cajander, V. Kann, A. Kapoor, R. McDermott, A. Peters, M. Sabin, and C. Wallace. Modelling competencies for computing education beyond 2020: a research based approach to defining competencies in the computing disciplines. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 148–174, 2018.
- [47] I. Fronza, A. Hellas, P. Ihantola, and T. Mikkonen. An exploration of cognitive shifting in writing code. In *Proceedings of the Conference on Global Computing Education (CompEd)*. ACM Chengdu, Sichuan, China, 2019.
- [48] A. C. Gil. *Métodos e técnicas de pesquisa social*. 6. ed. Editora Atlas SA, 2008.
- [49] D. Ginat and E. Menashe. Solo taxonomy for assessing novices algorithmic design. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Kansas City, MO, United States, 2015.
- [50] V. A. A. Gomes. *Investigando a Relação entre um Conjunto de Habilidades Preditoras de Programação e a Escrita/Leitura de Códigos em Iniciantes*. PhD thesis, Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Exatas e Sociais Aplicadas, Universidade Estadual da Paraíba., 2022.
- [51] J. M. Griffin. Learning by taking apart: Deconstructing code by reading, tracing, and debugging. In *Proceedings of the Conference on Information Technology Education (SIGITE)*. ACM Boston, MA, United States, 2016.

- [52] B. Harrington and N. Cheng. Tracing vs. writing code: Beyond the learning hierarchy. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Baltimore, MD, United States, 2018.
- [53] K. V. Hausswolff and A. Eckerdal. Measuring programming knowledge in a research context. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, San Jose, CA, United States, 2018.
- [54] N. S. Herman, S. B. Salam, and E. Noersasongko. A study of tracing and writing performance of novice students in introductory programming. In *Proceedings of the International Conference on Software Engineering and Computer Systems (ICSECS)*. Springer, 2011.
- [55] M. Hertz and M. Jump. Trace-based teaching in early programming courses. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Denver, Colorado, United States, 2013.
- [56] A. D. Hilton, G. M. Lipp, and S. H. Rodger. Translation from problem to code in seven steps. In *Proceedings of the Conference on Global Computing Education (CompEd)*. ACM Chengdu, Sichuan, China, 2019.
- [57] D. M. Hoffman, M. Lu, and T. Pelton. A web-based generation and delivery system for active code reading. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Dallas, Texas, United States, 2011.
- [58] C. S. Hutz, D. R. Bandeira, and C. M. Trentini. *Psicometria*. Artmed Editora, 2015.
- [59] IEEE. Ieee xplore library. <https://ieeexplore.ieee.org/>. Accessed: 2019-08-16.
- [60] V. Isomöttönen, A. Nylén, and V. Tirronen. Writing to learn programming? a single case pilot study. In *Proceedings of the Koli Calling International Conference on Computing Education Research*. ACM Joensuu, Finland, 2016.
- [61] C. Izu, A. Weerasinghe, and C. Pope. A study of code design skills in novice programmers using the solo taxonomy. In *Proceedings of the International Computing Education Research (ICER)*. ACM Melbourne, VIC, Australia, 2016.

- [62] N. Kasto, J. L. Whalley, A. Philpott, and D. Whalley. Solution spaces. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, New Zealand, 2014.
- [63] T. Kelly and J. Buckley. A context-aware analysis scheme for bloom's taxonomy. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 275–284. IEEE, 2006.
- [64] B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33, 2004.
- [65] Terry K Koo and Mae Y Li. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of Chiropractic Medicine*, 15(2):155–163, 2016.
- [66] Özgen Korkmaz and BAĪ Xuemei. Adapting computational thinking scale (cts) for chinese high school students and their thinking scale skills level. *Participatory Educational Research*, 6(1):10–26, 2019.
- [67] J. Kottner, L. Audigé, S. Brorson, A. Donner, B. J. Gajewski, A. Hróbjartsson, C. Roberts, M. Shoukri, and D. L. Streiner. Guidelines for reporting reliability and agreement studies (grras) were proposed. *International Journal of Nursing Studies*, 48(6):661–671, 2011.
- [68] David R Krathwohl. A revision of bloom's taxonomy: An overview. *Theory Into Practice*, 41(4):212–218, 2002.
- [69] A. N. Kumar. A mid-career review of teaching computer science i. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Denver, Colorado, United States, 2013.
- [70] Essi Lahtinen. A categorization of novice programmers: A cluster analysis study. In *PPIG*, volume 16, pages 32–41. Citeseer, 2007.
- [71] A. Lajis, S. A. Baharudin, D. A. Kadir, N. M. Ralim, H. M. Nasir, and N. A. Aziz. A review of techniques in automatic programming assessment for practical skill test.

- Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 10(2), 2018.
- [72] A. Lajis, H. M. Nasir, and N. A. Aziz. Proposed assessment framework based on bloom taxonomy cognitive competency: Introduction to programming. In *Proceedings of the International Conference on Software and Computer Applications (ICSCA)*. ACM, Kuantan, Malaysia, 2018.
- [73] S. Lee, A. Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, and H. Lim. Comparing programming language comprehension between novice and expert programmers using eeg analysis. In *Proceedings of the International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, Taichung, Taiwan, 2016.
- [74] R. L. Linn. Educational measurement: Overview. *International Encyclopedia of Education*, 3, 2010.
- [75] R. Lister, T. Clear, D. J. Bouvier, P. Carter, A. Eckerdal, J. Jacková, M. Lopez, R. McCartney, P. Robbins, O. Seppälä, and E. Thompson. Naturally occurring data as research instrument: Analyzing examination responses to study the novice programmer. *ACM SIGCSE Bulletin*, 41(4), 2010.
- [76] Ursula Lucas and Rosina Mladenovic. The identification of variation in students' understandings of disciplinary concepts: The application of the solo taxonomy within introductory accounting. *Higher Education*, 58(2):257–283, 2009.
- [77] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 55–106, 2018.
- [78] A. Luxton-Reilly, E. McMillan, E. Stevenson, E. Tempero, and P. Denny. Ladebug: An online tool to help novice programmers improve their debugging skills. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Larnaca, Cyprus, 2018.

- [79] R. A. Martins, J. B. P. Mello, and C. H. Turrioni. *Guia para elaboração de monografia e TCC em engenharia de produção*. Editora Atlas SA, 2013.
- [80] R. J. Marzano and J. S. Kendall. *The New Taxonomy of Educational Objectives*. Corwin Press, 2006.
- [81] R. Mathew, S. I. Malik, and R. M. Tawafak. Teaching problem solving skills using an educational game in a computer programming course. *Informatics in Education*, 18(2), 2019.
- [82] R. Matthews, S. H. Hew, and A. C. Koo. Empirical study of multimedia learning object to enhance introductory programming learning. In *Proceedings of the International Conference on Information Technology (ICIT)*. ACM Singapore, Singapore, 2017.
- [83] A. Matthíasdóttir and H. Arnalds. E-assessment: Students point of view. In *Proceedings of the Computer Systems and Technologies (CompSysTech)*. ACM Palermo, Italy, 2016.
- [84] E. McArdle, J. Holdsworth, and I. Lee. Assessing the usability of students object-oriented language with first-year it students: A case study. In *Proceedings of the Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration (OzCHI)*. ACM Adelaide, Australia, 2013.
- [85] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2), 2018.
- [86] B. Mendoza and L. Zavala. An intervention strategy to hone students code understanding skills. *Journal of Computing Sciences in Colleges (JCSC)*, 33(3), 2018.
- [87] T. Michaeli and R. Romeike. Current status and perspectives of debugging in the k12 classroom: A qualitative study. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)*. IEEE, Dubai, United Arab Emirates, 2019.

- [88] L. A. M. Morais. *Mastery Learning: uma abordagem personalizada de ensino no contexto de programação introdutória*. PhD thesis, Dissertação (Mestrado em Ciência da Computação) – Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande., 2015.
- [89] L. Murphy, S. Fitzgerald, R. Lister, and R. McCauley. Ability to explain in plain english linked to proficiency in computer-based programming. In *Proceedings of the International Computing Education Research (ICER)*. ACM Auckland, New Zealand, 2012.
- [90] T. Na, N. Funabiki, K. K. Zaw, N. Ishihara, S. Matsumoto, and W. C. Kao. A fill-in-blank problem workbook for java programming learning assistant system. *International Journal of Web Information Systems*, 13(2), 2017.
- [91] G. Näsström. Interpretation of standards with bloom’s revised taxonomy: a comparison of teachers and assessment experts. *International Journal of Research & Method in Education*, 32(1):39–51, 2009.
- [92] G. L. Nelson, A. Hu, B. Xie, and A. J. Ko. Towards validity for a formative assessment for language-specific program tracing skills. In *Proceedings of the Koli Calling International Conference on Computing Education Research*, 2019.
- [93] G. L. Nelson, B. Xie, and A. J. Ko. Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the International Computing Education Research (ICER)*. ACM Tacoma, WA, United States, 2017.
- [94] N. Nentl and R. Zietlow. Using bloom’s taxonomy to teach critical thinking skills to business students. *College & Undergraduate Libraries*, 15(1-2):159–172, 2008.
- [95] M. Nirgude. Debugger tool to improve conceptual understanding of programming language of novice learners. In *Proceedings of the International Conference on Technology for Education (T4E)*. IEEE, Kharagpur, India, 2013.
- [96] J. C. Nunnally. *Psychometric Theory 3E*. Tata McGraw-Hill Education, 1994.

- [97] C. J. Park and J. S. Hyun. Effects of abstract thinking and familiarity with programming languages on computer programming ability in high schools. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Wellington, New Zealand, 2014.
- [98] C. J. Park, J. S. Hyun, and J. Heuilan. Effects of gender and abstract thinking factors on adolescents' computer program learning. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, El Paso, TX, United States, 2015.
- [99] Eun Jung Park, Greg Light, Su Swarat, and Denise Denise. Understanding learning progression in student conceptualization of atomic structure by variation theory for learning. In *Learning Progressions in Science (LeaPS) Conference*, 2009.
- [100] D. Parsons, K. Wood, and P. Haden. What are we doing when we assess programming? In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Sydney, Australia, 2015.
- [101] L. Pasquali. *Psicometria: Teoria dos Testes na Psicologia e na Educação*. Editora Vozes Limitada, 2017.
- [102] A. Petersen, M. Craig, and D. Zingaro. Reviewing cs1 exam question content. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Dallas, Texas, United States, 2011.
- [103] R. Primi. *Psicometria: Fundamentos matemáticos da teoria clássica dos testes*. *Avaliação Psicológica*, 11(2), 2012.
- [104] T. Raykov and G. A. Marcoulides. *Introduction to Psychometric Theory*. Routledge, 2011.
- [105] Mihaela Sabin, Svetlana Peltsverger, Bill Paterson, Ming Zhang, and Hala Alrumaih. It2017 report: putting it to work. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 95–96, 2017.
- [106] K. Sadeghi and Z. A. Khonbi. An overview of differential item functioning in multistage computer adaptive testing using three-parameter logistic item response theory. *Language Testing in Asia*, 7(1):1–16, 2017.



- [107] José-Manuel Sáez-López, Maria-Luisa Sevillano-García, and Esteban Vazquez-Cano. The effect of programming on primary school students' mathematical and scientific understanding: educational use of mbot. *Educational Technology Research and Development*, 67(6):1405–1425, 2019.
- [108] K. Sanders, M. Ahmadzadeh, T. Clear, S. H. Edwards, M. Goldweber, C. Johnson, R. Lister, R. McCartney, E. Patitsas, and J. Spacco. The canterbury questionbank: Building a repository of multiple-choice cs1 and cs2 questions. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Canterbury, United Kingdom, 2013.
- [109] P. Sands. Addressing cognitive load in the computer science classroom. *ACM Inroads*, 10(1), 2019.
- [110] Jucelio S Santos, Wilkerson L Andrade, João Brunet, and Monilly Ramos Araujo Melo. Applying item response theory to evaluate instruments of introductory programming skills measurement. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE, 2020.
- [111] L. M. A. Sartes and M. L. O. S. Formigoni. Avanços na psicometria: da teoria clássica dos testes à teoria de resposta ao item. *Psicologia: Reflexão e Crítica*, 26(2):241–250, 2013.
- [112] C. Schulte. Block model: An educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the International Workshop on Computing Education Research*, 2008.
- [113] Science. Scencedirect library. <https://www.sciencedirect.com/>. Accessed: 2020-02-10.
- [114] Scopus. Scopus library. <https://www.scopus.com/>. Accessed: 2020-02-10.
- [115] A. Shargabi, S. A. Aljunid, M. Annamalai, S. M. Shuhidan, and A. M. Zin. Tasks that can improve novices' program comprehension. In *Proceedings of the Conference on e-Learning, e-Management and e-Services (IC3e)*. IEEE, Melaka, Malaysia, 2015.

- [116] J. Sheard, J. Dermoudy, D. D'Souza, M. Hu, and D. Parsons. Benchmarking a set of exam questions for introductory programming. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, New Zealand, 2014.
- [117] N. Shi, P. Zhang, and X. Sun. B-learning on novice programmer learning with roles of variables. *International Journal of Simulation Systems, Science & Technology (IJSSST)*, 17(32), 2016.
- [118] D. T. Silveira and F. P. Córdova. A pesquisa científica. *Métodos de pesquisa. Porto Alegre: Editora da UFRGS, 2009. p. 33-44, 2009.*
- [119] Simon and S. Snowdon. Explaining program code: Giving students the answer helps-but only jus. In *Proceedings of the International Computing Education Research (ICER)*. ACM Providence, Rhode Island, United States, 2011.
- [120] J. Skalka and M. Drlík. Educational model for improving programming skills based on conceptual microlearning framework. In *Proceedings of the International Conference on Interactive Collaborative Learning (ICL)*. Springer, Cham, 2018.
- [121] R. Smetsers-Weeda and S. Smetsers. Problem solving and algorithmic development with flowcharts. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE)*. ACM Nijmegen, Netherlands, 2017.
- [122] L. M. Souza, B. M. Ferreira, I. M. Félix, L. Oliveira B., A. A. F. Brandão, and P. A. Pereira. Mathematics and programming: Marriage or divorce? In *Proceedings of the World Conference on Engineering Education (EDUNINE)*. IEEE, Lima, Peru, 2019.
- [123] J. Stamey and S. Sheel. A boot camp approach to learning programming in a cs0 course. *The Journal of Computing Sciences in Colleges (JCSC)*, 25(5), 2010.
- [124] X. Suo. Toward more effective strategies in teaching programming for novice students. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Hong Kong, China, 2012.
- [125] S. M. Taheri, M. Sasaki, and H. T. Ngetha. Evaluating the effectiveness of problem solving techniques and tools in programming. In *Proceedings of the Science and Information Conference (SAI)*. IEEE, Logon, United Kingdom, 2015.

- [126] S. Takada, E. Cuadros-Vargas, J. Impagliazzo, S. Gordon, L. Marshall, H. Topi, G. V. Veer, and L. Waguespack. Toward the visual understanding of computing curricula. *Education and Information Technologies*, 25(5):4231–4270, 2020.
- [127] A. L. Trevisan and R. G. Amaral. A taxonomia revisada de bloom aplicada à avaliação: Um estudo de provas escritas de matemática. *Ciência & Educação (Bauru)*, 22(2):451–464, 2016.
- [128] Y. L. P. Vega, J. C. G. Bolaños, G. M. F. Nieto, and S. M. Baldiris. Application of item response theory (irt) for the generation of adaptive assessments in an introductory course on object-oriented programming. In *Proceedings of the IEEE Frontiers in Education Conference (FIE)*. IEEE, 2012.
- [129] C. M. M. Vendramini and A. S. Dias. Teoria de resposta ao item na análise de uma prova de estatística em universitários. *Psico-USF*, 10(2):201–210, 2005.
- [130] A. Vihavainen, C. S. Miller, and A. Settle. Benefits of self-explanation in introductory programming. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Kansas City, MO, United States, 2015.
- [131] L. Waguespack, H. Topi, S. Frezza, J. Babb, L. Marshall, S. Takada, G. V. Veer, and A. Pears. Adopting competency mindful of professionalism in baccalaureate computing curricula. In *Proceedings of the EDSIG Conference ISSN*, volume 2473, page 3857, 2019.
- [132] S. Wang, Y. Han, W. Wu, and Z. Hu. Modeling student learning outcomes in studying programming language course. In *Proceedings of the Seventh International Conference on Information Science and Technology (ICIST)*. IEEE, 2017.
- [133] X. S. Wang and J. Souders. Improving debugging education through applied learning. *Journal of Computing Sciences in Colleges (JCSC)*, 27(3), 2012.
- [134] E. J. Wartha and D. A. S. de Santana. Construção e validação de instrumento de coleta de dados na pesquisa em ensino de ciências. *Amazônia: Revista de Educação em Ciências e Matemáticas*, 16(36):39–52, 2020.

- 
- [135] J. Whalley, T. Clear, R. Phil, and T. Errol. Salient elements in novice solutions to code writing problems. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, Perth, Australia, 2011.
- [136] J. Whalley and N. Kasto. How difficult are novice code writing tasks? a software metrics approach. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, New Zealand, 2014.
- [137] B. Xie, D. Loksa, G. L. Nelson, M. J. Davidson, D. Dong, H. Kwik, A. H. Tan, L. Hwa, M. Li, and A. J. Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3), 2019.
- [138] P. S. Yanagi, T. S. Indi, and M. A. Nirgude. Enhancing the cognitive level of novice learners using effective program writing skills. In *Proceedings of the International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. IEEE, Logon, United Kingdom, 2016.
- [139] L. Zavala and Mendoza B. Precursor skills to writing code. *Journal of Computing Sciences in Colleges (JCSC)*, 32(3), 2017.
- [140] D. Zingaro, A. Petersen, and Craig M. Stepping up to integrative questions on cs1 exams. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Raleigh, North Carolina, United States, 2012.

# **Appendix A**

## **Consent Term—SPLab**



UNIVERSIDADE FEDERAL DE CAMPINA GRANDE  
LABORATÓRIO DE PRÁTICAS DE SOFTWARE  
Rua.: Aprígio Veloso, nº 882, Bairro Universitário, Campina Grande, PB.  
CEP.: 58429-900 - Tel.: 2101-1429

### DECLARAÇÃO DE ANUÊNCIA

Eu, Dr. Dalton Dario Serey Guerrero, Coordenador do Laboratório de Práticas de Software, autorizo o desenvolvimento da pesquisa intitulada: **“Mensuração de habilidades cognitivas introdutórias de programação por meio de uma avaliação adaptativa informatizada”**. A pesquisa será realizada em parceria com o referido laboratório, nos anos de 2020 a 2022, tendo como orientadores Dr. Wilkerson de Lucena Andrade e Dr. João Arthur Brunet Monteiro e orientando Ms. Jucelio Soares dos Santos.

Campina Grande - PB, 19 de dezembro de 2020.

  
\_\_\_\_\_  
Dr. Dalton Dario Serey Guerrero  
(Coord. Laboratório de Práticas de Software)

## **Appendix B**

### **Consent Term—LabNEUROCIT**



UNIVERSIDADE FEDERAL DE CAMPINA GRANDE  
LABORATÓRIO DE NEUROPSICOLOGIA COGNITIVA E INOVAÇÃO TECNOLÓGICA  
Av. Juvêncio Arruda, 795 - Bodocongó, Campina Grande - PB.  
Cep.: 58429-600

### DECLARAÇÃO DE ANUÊNCIA

Eu, Dra. Monilly Ramos de Araújo Melo, Coordenadora do Laboratório de Neuropsicologia Cognitiva e Inovação Tecnológica, autorizo o desenvolvimento da pesquisa intitulada: **“Mensuração de habilidades cognitivas introdutórias de programação por meio de uma avaliação adaptativa informatizada”**. A pesquisa será realizada em parceria com o referido laboratório, nos anos de 2020 a 2022, tendo como orientadores Dr. Wilkerson de Lucena Andrade e Dr. João Arthur Brunet Monteiro e orientando Ms. Jucelio Soares dos Santos.

Campina Grande - PB, 19 de dezembro de 2020.

A handwritten signature in blue ink, reading 'Monilly Ramos Araujo Melo', is written over a horizontal line.

Dra. Monilly Ramos de Araújo Melo  
(Coord. do LabNEUROCIT/UFCG/CNPQ)



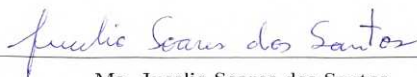
# **Appendix C**

## **Partnership Term**

## Termo de Parceria

Este termo celebra a parceria entre o Laboratório de Neuropsicologia Cognitiva e Inovação Tecnológica - LabNEUROCIT/UFCG/CNPQ, coordenado pela Professora Dr<sup>a</sup> Monilly Ramos Araújo Melo, o Doutorando Jucelio Soares dos Santos e seus Orientadores Dr. Wilkerson Lucena de Andrade e Dr. João Arthur Brunet Monteiro, do Programa Pós-graduação em Ciência da Computação da Universidade Federal de Campina Grande, em que todos assumem o compromisso de registrar nos trabalhos acadêmicos e publicações derivadas dessa parceria, os autores e créditos dos instrumentos psicométricos, softwares, e demais documentos elaborados para os fins necessários ao desenvolvimento dos trabalhos mediante prévia autorização expressa.

Campina Grande - PB, 19 de dezembro de 2020.



Ms. Jucelio Soares dos Santos  
(Doutorando - Universidade Federal de Campina Grande)



Dr. Wilkerson de Lucena Andrade  
(Universidade Federal de Campina Grande)



Dr. João Arthur Brunet Monteiro  
(Universidade Federal de Campina Grande)



Dra. Monilly Ramos de Araújo Melo  
(Coord. do LabNEUROCIT/UFCG/CNPQ)

## **Appendix D**

# **Researchers' Commitment Agreement Term**

## Termo de Compromisso dos Pesquisadores

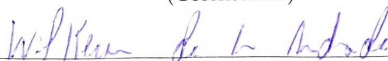
Por este termo de responsabilidade, nós abaixo-assinados, Orientador e Orientando(s) respectivamente, da pesquisa intitulada **“Mensuração de habilidades cognitivas introdutórias de programação por meio de uma avaliação adaptativa informatizada”**, assumimos cumprir fielmente as diretrizes regulamentadoras emanadas da Resolução nº 466, de 12 de Dezembro de 2012 do Conselho Nacional de Saúde/ MS e suas Complementares, homologada nos termos do Decreto de delegação de competências de 12 de novembro de 1991, visando assegurar os direitos e deveres que dizem respeito à comunidade científica, ao (s) sujeito (s) da pesquisa e ao Estado.

Reafirmamos, outros sim, nossa responsabilidade indelegável e intransferível, mantendo em arquivo todas as informações inerentes a presente pesquisa, respeitando a confidencialidade e sigilo das fichas correspondentes a cada sujeito incluído na pesquisa, por um período de 5 (cinco) anos após o término desta. Apresentaremos sempre que solicitado pelo CEP/ CFP/UFCG (Comitê de Ética em Pesquisas/ Centro de Formações de Professores) ou CONEP (Comissão Nacional de Ética em Pesquisa) ou, ainda, as Curadorias envolvidas no presente estudo, relatório sobre o andamento da pesquisa, comunicando ainda ao CEP/CFP/UFCG, qualquer eventual modificação proposta no supracitado projeto.

Campina Grande - PB, 19 de dezembro de 2020.



Ms. Jucelio Soares dos Santos  
(Orientando)



Dr. Wilkerson de Lucena Andrade  
(Orientador)



Dr. João Arthur Brunet Monteiro  
(Orientador)



Dra. Monilly Ramos de Araújo Melo  
(Colaboradora)

## **Appendix E**

### **Results Disclosure Commitment' Term**

## Termo de Compromisso de Divulgação dos Resultados

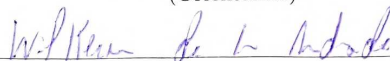
Por este termo de responsabilidade, nós, abaixo – assinados, respectivamente, orientadores, orientando e colaborada da pesquisa intitulada “**Mensuração de habilidades cognitivas introdutórias de programação por meio de uma avaliação adaptativa informatizada**”, assumimos o compromisso de:

- Preservar a privacidade dos participantes da pesquisa cujos dados serão coletados;
- Assegurar que as informações serão utilizadas única e exclusivamente para a execução do projeto em questão;
- Assegurar os benefícios resultantes do projeto retornem aos participantes da pesquisa, seja em termos de retorno social, acesso aos procedimentos, produtos ou agentes da pesquisa;
- Assegurar que as informações somente serão divulgadas de forma anônima, não sendo usadas iniciais ou quaisquer outras indicações que possam identificar o sujeito da pesquisa;
- Assegurar que os resultados da pesquisa serão encaminhados para a publicação, com os devidos créditos aos autores.

Campina Grande - PB, 19 de dezembro de 2020.



Ms. Jucelio Soares dos Santos  
(Orientando)



Dr. Wilkerson de Lucena Andrade  
(Orientador)



Dr. João Arthur Brunet Monteiro  
(Orientador)



Dra. Monilly Ramos de Araújo Melo  
(Colaboradora)

## **Appendix F**

# **Bloom's Revised Taxonomy for Measuring and Fostering Cognitive Programming Skills**

# Taxonomia Revisada de Bloom Adaptada para Mensurar/Fomentar Habilidades Cognitivas de Programação

Este documento apresenta a versão original da Taxonomia Revisada de Bloom, a síntese de sua adaptação para o Ensino de Programação e associação das habilidades cognitivas de programação aos níveis cognitivos da Taxonomia Adaptada.

## Contextualização

Escrever código requer outras habilidades que o professor não ensina tradicionalmente. Em estudos anteriores, identificamos várias linhas de pesquisa sustentam a premissa de que as pessoas aprenderiam a programar de forma mais eficaz e eficiente se gastassem mais tempo rastreando, explicando, entendendo, lendo, depurando e modificando código do que escrevendo código.

Os princípios da Ciência Cognitiva incluem um modelo gráfico que suporta o processo de aprendizagem para resolver problemas. Educadores e designers instrucionais podem usar essa progressão para promover o aprendizado em seus alunos. A progressão começa com atividades desconstrucionistas de baixo risco, como explorar, identificar, comparar e depurar, antes de atividades que exigem a escrita de código. Os educadores de informática devem ajustar o design instrucional da Educação Tradicional e escolher tarefas menos densas para reduzir a carga cognitiva dos alunos.

Paralelamente, educadores em Ciência da Computação (CS) relatam que a satisfação dos alunos com o aprendizado aumenta quando se sentem desafiados. No entanto, desafios levados ao extremo podem desencorajá-los, dificultando seu progresso. Portanto, é essencial fornecer aos alunos o nível de desafio adequado que expanda suas habilidades e compreensão de conceitos, mas não os faça se sentir frustrados. Praticar e classificar os alunos no nível de desafio apropriado é particularmente crítico no Curso Introdutório de Programação (CS1), que historicamente enfrenta problemas de retenção.

Precisamos considerar alguns fatores para determinar o nível de desafio "apropriado" de um teste ou atividade. Em primeiro lugar, é essencial notar que o "nível de desafio" de uma atividade é um conceito relativo que varia de acordo com o contexto e a situação, como repetição de materiais ou exemplos semelhantes. Segundo, deve haver algumas métricas para determinar o nível de desafio relacionado a diferentes contextos. Assim, *como sequenciamos as habilidades cognitivas envolvidas no aprendizado introdutório de programação para determinar o nível de desafio apropriado em um instrumento de avaliação?*

Para começar a responder a esta pergunta, ressaltamos em estudos anteriores o domínio cognitivo da Taxonomia Revisada de Bloom para sequenciar a carga cognitiva e melhorar a qualidade da avaliação. A Taxonomia Revisada de Bloom é amplamente utilizada em diversas áreas do conhecimento para promover e avaliar a aprendizagem. A estrutura hierárquica do domínio cognitivo da Taxonomia Revisada de Bloom representa a profundidade de aprendizado do aluno em um determinado assunto. Em seu domínio cognitivo, esta Taxonomia possui seis níveis, a saber: lembrar, compreender, aplicar, analisar, avaliar e criar.

Em sua versão tradicional, a Taxonomia Revisada de Bloom começa com o nível "lembrar", que indica o nível mais baixo de aprendizado do aluno (ou seja, recordação de fatos e observação e recordação de informações aprendidas). O próximo nível é "entender",



que pode entender as informações e traduzi-las em um contexto diferente. O nível "aplicar" é a capacidade de aplicar informações em uma situação concreta ou em um novo contexto usando as habilidades e conhecimentos que os alunos aprenderam. Então, no nível "analisar", os alunos devem ser capazes de dividir um problema em partes menores e identificar relacionamentos. No nível "avaliar", o aluno avalia possíveis soluções ou ideias para um determinado problema e faz julgamentos. Finalmente, no nível "criar", é a capacidade de combinar as diferentes partes para formar uma nova entidade, relacionando conhecimentos de várias áreas e usando ideias antigas para criar uma nova [1].

Assim, começamos inicialmente com uma hipótese alternativa: *H1-1. Sequenciar as habilidades cognitivas envolvidas no aprendizado introdutório de programação dentro do nível cognitivo da Taxonomia Revisada de Bloom melhora a adoção/medição em CS1.*

Cada habilidade presente no nível cognitivo da Taxonomia Revisada de Bloom tem um objetivo que orienta o professor a desenvolver testes em diferentes áreas. Até onde sabemos, nenhum trabalho explorou a adaptação dos objetivos dessas habilidades no ensino de programação. Com isso em mente, este documento apresenta uma proposta para sequenciar habilidades de programação introdutórias no nível cognitivo da Taxonomia Revisada de Bloom. Para implementar esta proposta, adaptamos o nível cognitivo da Taxonomia Revisada de Bloom no CS1. Associamos as habilidades de programação cognitiva com a Taxonomia Adaptada. Por fim, precisamos da sua ajuda para analisar teoricamente as etapas de adaptação e associação.

## **Adaptação/Associação**

### **Remember/Lembrar**

O primeiro nível cognitivo da Taxonomia Revisada de Bloom envolve lembrar informações e conteúdos previamente cobertos (ou seja, classificações, regras ou procedimentos). Este nível envolve lembrar uma quantidade significativa de informações ou fatos específicos, e o objetivo deste nível é trazer esse conhecimento à consciência [1]. A Tabela 1 apresenta a adaptação do nível cognitivo remember/lembrar para o ensino de programação, bem como as habilidades de programação cognitiva associadas aos níveis cognitivos da Taxonomia Adaptada.

**Tabela 1.** Adaptação do Nível *Remember/Lembrar* e Associação das Habilidades Cognitivas de Programação.

<b>Habilidade</b>	<b>Definição Original</b>	<b>Adaptação</b>	<b>Associação</b>
<i>Recognizing/</i> Reconhecer	<i>Locate knowledge in long-term memory that is consistent with the presented material.</i> Localizar o conhecimento na memória de longo prazo que seja consistente com o material apresentado.	Localizar o conhecimento relevante da memória de longo prazo e compará-lo com a informação apresentada.	-
<i>Recalling/</i> Recordar	<i>Retrieve relevant knowledge from long-term memory.</i> Recuperar conhecimento relevante da memória de longo prazo.	Recuperar o conhecimento relevante da memória de longo prazo quando solicitado.	-

Tradicionalmente, o professor se concentra apenas no aprendizado mecânico; O ensino e avaliação concentram-se apenas na lembrança de elementos ou fragmentos de conhecimento, muitas vezes isolados de seu contexto. No entanto, quando os professores se concentram na aprendizagem significativa, a aprendizagem é integrada com a tarefa mais ampla de construir novos conhecimentos ou resolver novos problemas. Por exemplo, o conhecimento das estruturas de seleção é necessário se o aluno deseja resolver problemas de desvio de fluxo, pois permite executar um ou mais comandos se a condição testada for real ou executar um ou mais comandos se for falsa.

As habilidades cognitivas no nível *Remember/Lembrar* incluem:

- **Recognizing/Reconhecer** envolve recuperar conhecimento relevante da memória de longo prazo para compará-lo com a informação apresentada. Reconhecer a demanda do aluno por informações de memória de longo prazo idênticas às informações apresentadas (conforme representadas na memória de trabalho). Ao apresentar a nova informação, o aluno determina se esta informação corresponde ao conhecimento previamente aprendido;
- **Recalling/Recordar** envolve recuperar conhecimento relevante da memória de longo prazo quando solicitado a fazê-lo. O prompt é muitas vezes uma pergunta. Ao lembrar, o aluno procura a informação na memória de longo prazo e processa parte da informação da memória de trabalho.

Se, por exemplo, um aluno aprendeu estruturas de seleção, um teste de memória pode envolver pedir ao aluno para combinar cada uma das estruturas de seleção com seu respectivo funcionamento em uma segunda lista (ou seja, *recognizing/reconhecendo*) ou escrever o funcionamento correspondente ao lado de cada uma das estruturas de seleção apresentadas na lista (ou seja, *recalling/recordando*).

### ***Understand/Compreender***

O nível cognitivo *Understand/Compreender* aplicar o significado ao conteúdo. Esse nível cognitivo traduz o conteúdo entendido em uma nova forma (ou seja, oral, escrita e diagramas) ou contexto. Existe a capacidade de entender a informação ou fato, capturar seu significado e usá-lo em diferentes contextos [1]. A Tabela 2 apresenta a adaptação do nível cognitivo *Understand/Compreender* para o ensino de programação, bem como as habilidades de programação cognitiva associadas aos níveis cognitivos da Taxonomia Adaptada.

Os alunos entendem quando constroem conexões entre os "novos" conhecimentos adquiridos e seus conhecimentos prévios. O conhecimento que chega é integrado com as estruturas e esquemas cognitivos existentes. Como os conceitos são os blocos de construção desses esquemas e estruturas, o conhecimento conceitual fornece uma base para a compreensão.

**Tabela 2.** Adaptação do Nível *Understand*/Compreender e Associação das Habilidades Cognitivas de Programação.

Habilidade	Definição Original	Adaptação	Associação
<i>Interpreting</i> / Interpretar	<i>Change from one representation form to another.</i> Mudar de uma forma de representação para outra.	Traduzir de uma forma de algoritmo para outra.	Compreensão e Leitura.
<i>Exemplifying</i> / Exemplificar	<i>Find a specific example or concept illustration or principle.</i> Encontrar um exemplo específico de ilustração de conceito ou princípio.	Encontrar um exemplo de um problema específico.	Compreensão e Leitura.
<i>Classifying</i> / Classificar	<i>Determine that something belongs to a category.</i> Determinar que algo pertence a uma categoria.	Determinar que algo pertence a uma categoria.	Compreensão e Leitura.
<i>Summarizing</i> / Sumarizar	<i>Abstract a general theme or significant point(s).</i> Abstrair um tema geral ou ponto(s) significativo(s).	Resumir/comentar sobre trechos/partes do código.	Compreensão e Leitura.
<i>Infering</i> / Inferir	<i>Draw a logical conclusion from the presented information.</i> Tirar uma conclusão lógica das informações apresentadas.	Tirar uma conclusão lógica de um código apresentado.	Compreensão, Leitura e Rastreamento.
<i>Comparing</i> / Comparar	<i>Detect correspondences between two ideas, objects, and the like.</i> Detectar correspondências entre duas ideias, objetos e similares.	Detectar correspondências entre pseudocódigo e um código.	Compreensão e Leitura.
<i>Explaining</i> / Exemplificar	<i>Construct a system' cause-and-effect mode.</i> Construir o modo de causa e efeito de um sistema.	Construir um modelo de causa e efeito de um algoritmo.	Compreensão, Leitura e Explicando.

As habilidades cognitivas do nível *Understand*/Compreender incluem:

- ***Interpreting*/Interpretar** é a capacidade do aluno de traduzir uma forma de algoritmo para outra forma (por exemplo, traduzir um algoritmo para um fluxograma);
- ***Exemplifying*/Exemplificar** envolve identificar as características definidoras do conceito ou princípio geral (por exemplo, as regras para nomes de variáveis) e usar essas características para selecionar ou construir uma instância específica (por exemplo, poder selecionar qual dos nomes de variáveis apresentados é declarado de forma incorreta);
- ***Classifying*/Classificar** ocorre quando um aluno reconhece que algo (por exemplo, uma instância ou exemplo particular) pertence a uma categoria específica (por

exemplo, conceito ou princípio). A classificação envolve a detecção de características ou padrões relevantes que "se ajustam" tanto à instância específica quanto ao conceito ou princípio. A classificação é um processo complementar à exemplificação. Enquanto a exemplificação começa com um conceito ou princípio geral e exige que o aluno encontre uma instância ou exemplo específico, a classificação começa com uma instância ou exemplo específico e exige que o aluno encontre um conceito ou princípio geral;

- **Summarizing/Sumarizar** ocorre quando um aluno sugere uma única afirmação representando a informação apresentada ou resume um tema geral (por exemplo, aprender a resumir os propósitos da sub-rotina em um programa geral);
- **Inferring/Inferir** é a capacidade do aluno de tirar uma conclusão lógica de um algoritmo ou código apresentado (por exemplo, descobrir a saída do código) [2] [23] [24] [27] [28] [30] [47] [52].
- **Comparing/Comparar** é a capacidade do aluno de detectar semelhanças e diferenças entre duas ou mais ideias e determinar o quão conhecido um evento é com um menos familiar (por exemplo, detectar correspondências entre pseudocódigo e um código);
- **Explaining/Exemplificar** é a capacidade do aluno de construir e usar o modelo de causa e efeito de um programa. Várias tarefas podem avaliar a capacidade de explicação de um aluno, incluindo raciocínio, resolução de problemas, redesenho e previsão [6] [10] [42] [44] [52] [55] [61]. Em tarefas de raciocínio, um aluno oferece uma razão para um determinado evento. Na resolução de problemas, um aluno diagnostica o que poderia ter dado errado em um programa falho. No redesenho, um aluno altera o programa para atingir algum objetivo. Na previsão, o aluno pode avaliar que uma mudança em uma parte do programa afetará outra parte do programa.

Várias linhas de pesquisa têm confundido "habilidade cognitiva" com "nível cognitivo". A compreensão não é uma habilidade, mas um sinônimo relacionado ao nível cognitivo "*understand/entender*" [13] [15] [16] [18] [19] [20] [21] [36] [42] [46] [47] [53] [61], bem como "leitura" de código, que envolve o nível cognitivo "entender" [4] [8] [10] [27] [41] [45] [50] [51] [59]. Esses trabalhos não especificaram quais habilidades de nível cognitivo a que se referiam, por isso associamos compressão e leitura em todas as habilidades do nível cognitivo *understand/entender*.

### **Apply/Aplicar**

O nível cognitivo de aplicação utiliza informações, métodos e conteúdos aprendidos em novas situações concretas. Inclua aplicações, métodos, modelos, conceitos, leis e teorias de regras [1]. A Tabela 3 apresenta a adaptação do nível cognitivo *Apply/Aplicar* para o ensino de programação, bem como as habilidades de programação cognitiva associadas aos níveis cognitivos da Taxonomia Adaptada.

**Tabela 3.** Adaptação do Nível *Apply*/Aplicar e Associação das Habilidades Cognitivas de Programação.

Habilidade	Definição Original	Adaptação	Associação
<i>Executing</i> / Executar	<i>Apply a procedure to a familiar task.</i> Aplicar um procedimento a uma tarefa familiar.	Aplicar um procedimento a um problema familiar.	-
<i>Implementing</i> / Implementar	<i>Apply a procedure to an unfamiliar task.</i> Aplicar um procedimento a uma tarefa desconhecida.	Aplicar um procedimento a um problema desconhecido.	-

*Apply*/Aplicar consiste em dois processos cognitivos: Executar e Implementar. Ao executar, o professor apresenta ao aluno um problema familiar e um procedimento para resolvê-la. O aluno pode fornecer uma resposta ou, se for o caso, selecionar um conjunto de respostas possíveis. Na implementação, o aluno recebe um problema desconhecido que deve ser resolvido. Portanto, a maioria dos formatos de avaliação começa com a especificação do problema. Os alunos determinam o procedimento necessário para resolver o problema, usam o procedimento selecionado (fazendo as alterações necessárias) ou geralmente ambos.

As habilidades cognitivas do nível *Apply*/Aplicar incluem:

- ***Executing*/Executar** é a capacidade do aluno de realizar um procedimento de problema familiar (por exemplo, solicitar ao aluno que execute um procedimento no problema da família);
- ***Implementing*/Implementar** é a capacidade do aluno de usar um procedimento de problema desconhecido (por exemplo, solicitar ao aluno que execute um procedimento em um problema desconhecido).

### **Analyse/Analisar**

O nível cognitivo *Analyse*/Analisar subdivide o conteúdo em partes menores para compreender a estrutura final. Esse nível cognitivo pode incluir a identificação das partes, a análise do relacionamento entre as partes e o reconhecimento dos princípios organizacionais envolvidos. Identificar partes e suas inter-relações. Neste ponto, é necessário ter compreendido o conteúdo e a estrutura do objeto de estudo [1]. A Tabela 4 apresenta a adaptação do nível cognitivo *Analyse*/Analisar para o ensino de programação, bem como as habilidades de programação cognitiva associadas aos níveis cognitivos da Taxonomia Adaptada.

**Tabela 4.** Adaptação do Nível *Analyse*/Analisar e Associação das Habilidades Cognitivas de Programação.

Habilidade	Definição Original	Adaptação	Associação
<i>Differentiating</i> / Diferenciar	<i>Distinguish relevant from irrelevant parts or important from unimportant parts of the presented material.</i> Distinguir partes relevantes de irrelevantes ou importantes de partes sem importância do material apresentado.	Distinguir partes relevantes de partes irrelevantes do código/ algoritmo apresentado.	-
<i>Organizing</i> / Organizar	<i>Determine how elements fit or function within a structure.</i> Determinar como os elementos se encaixam ou funcionam dentro de uma estrutura.	Organizar as partes do código para atingir o objetivo do programa.	Modificar
<i>Attributing</i> / Atribuir	<i>Determine a viewpoint, bias, values, or intent underlying the presented material.</i> Determinar um ponto de vista, preconceito, valores ou intenção subjacente ao material apresentado.	Determinar o ponto de vista do código.	-

Embora aprender a analisar possa ser visto como um fim em si mesmo, é provavelmente mais defensável educacionalmente considerar a análise como uma extensão da compreensão ou como um prelúdio para avaliar ou criar. Melhorar as habilidades de análise de código dos alunos é um objetivo em muitas universidades em programas de ensino. Nos cursos introdutórios de programação, os professores fornecem exemplos de código para os alunos "aprenderem a analisar" como objetivos essenciais. Por exemplo, eles podem desejar desenvolver a capacidade de seus alunos de dividir uma tarefa de programação em partes, organizar as partes para atingir um objetivo geral e identificar componentes críticos ou sem importância para o desenvolvimento.

Compreender, Analisar e Avaliar são inter-relacionados e frequentemente usados de forma iterativa na execução de tarefas cognitivas. Ao mesmo tempo, porém, é essencial mantê-los como categorias de processo separadas. Uma pessoa que entende um problema pode não analisá-lo bem e, da mesma forma, alguém hábil em analisar um problema pode avaliá-lo mal.

As habilidades cognitivas do nível *Analyse*/Analisar incluem:

- ***Differentiating*/Diferenciar** é a capacidade do aluno de distinguir partes relevantes de partes irrelevantes do código ou algoritmo apresentado (por exemplo, distinguir partes irrelevantes do código). A diferenciação ocorre quando um aluno discrimina informações relevantes/essenciais de informações irrelevantes/sem importância. Essa habilidade é diferente das habilidades cognitivas associadas à compreensão

porque envolve organização estrutural e, em particular, determina como as partes se encaixam na estrutura geral ou no todo;

- **Organizing/Organizar** é a capacidade do aluno de ordenar as partes de um código para atingir o objetivo do programa (como organizar código ou trechos de algoritmo para que a lógica esteja correta). Ao organizar, o aluno constrói conexões sistemáticas e coerentes entre as informações apresentadas. A organização geralmente ocorre em conjunto com a diferenciação. O aluno primeiro identifica os elementos relevantes ou essenciais e, em seguida, determina a estrutura geral na qual os elementos se encaixam. A organização também pode ocorrer em conjunto com a atribuição, em que o foco está em determinar a intenção ou o ponto de vista do autor;
- **Attributing/Atribuir** é a capacidade do aluno de determinar o ponto de vista de um determinado código (por exemplo, o aluno constrói ou seleciona uma descrição de um determinado código). A atribuição envolve um processo de desconstrução, no qual o aluno determina os objetivos materiais apresentados. Ao contrário da interpretação, em que o aluno procura compreender o significado do material apresentado, a atribuição envolve uma extensão além da compreensão necessária para inferir a intenção ou o ponto de vista subjacente ao material apresentado.

Modificar envolve usar ou modificar o código existente [7] [42] [51] [52] [61]. O professor pode estimular essa habilidade em seus alunos com exercícios, a saber: i) preenchimento de um trecho de código (seja escrevendo o código que falta ou escolhendo-o em um conjunto de opções); ii) escrever chamadas de função para determinadas funções para que um resultado específico; iii) construir um programa a partir de um conjunto de fragmentos de código, nem todos podem fazer parte da solução; iv) *ordenar um programa embaralhado (também conhecido como quebra-cabeças de programação de Parson)*.

### **Evaluate/Avaliar**

O nível cognitivo *Evaluate/Avaliar* envolve julgar o valor material (proposta, pesquisa, projeto) para uma finalidade específica. O julgamento é baseado em critérios que podem ser externos (relevância) ou internos (organização) e podem ser fornecidos ou identificados conjuntamente [1]. A Tabela 5 apresenta a adaptação do nível cognitivo avaliado para o ensino de programação, bem como as habilidades de programação cognitiva associadas aos níveis cognitivos da Taxonomia Adaptada.

Esses critérios podem ser determinados pelo aluno ou por outros. Além disso, os padrões podem ser quantitativos ou qualitativos e, se aplicados aos critérios, por exemplo, esse código ou algoritmo é suficientemente convincente? A categoria de avaliação inclui processos de verificação cognitiva e julgamentos críticos baseados em critérios externos.

As habilidades cognitivas do nível *Evaluate/Avaliar* incluem:

- **Checking/Verificar** é a capacidade do aluno de detectar a eficácia de um programa conforme o aluno o implementa (por exemplo, encontrar/corrigir um erro lógico em um determinado trecho de código) [3] [5] [7] [11] [12] [14] [18] [19] [20] [21] [23] [24] [27] [35] [37] [40] [41] [43] [48] [54] [56].
- **Critiquing/Criticar** é a capacidade do aluno de criticar procedimentos para resolver um problema com base em padrões de codificação (por exemplo, julgar qual dos dois códigos ou algoritmos é a melhor maneira de resolver um determinado problema) [7] [40] [42] [52] [61].

**Tabela 5.** Adaptação do Nível *Evaluate*/Avaliar e Associação das Habilidades Cognitivas de Programação.

Habilidade	Definição Original	Adaptação	Associação
<i>Checking</i> / Verificar	<i>Detect inconsistencies or fallacies within a process or product, determining whether a process or product has internal consistency.</i> Detectar inconsistências ou falácias dentro de um processo ou produto, determinando se um processo ou produto tem consistência interna.	Detectar/corrigir falhas em um programa implementado.	Depurar
<i>Critiquing</i> / Criticar	<i>Detect the procedure suitability for a given problem.</i> Detectar a adequação do procedimento para um determinado problema.	Criticar procedimentos para resolver um problema.	-

Quando combinado com planejamento (uma habilidade cognitiva na categoria Criar) e implementação (uma habilidade cognitiva na categoria Aplicar), a verificação envolve determinar como o plano funciona. Ao criticar, o aluno observa as características positivas e negativas de um código ou algoritmo e faz um julgamento baseado, pelo menos parcialmente, nessas características. A crítica está no cerne do que tem sido chamado de pensamento crítico. Um exemplo crítico é julgar os méritos da solução particular para o problema em termos de sua eficiência.

A computação conhece a verificação como teste ou depuração. A depuração é uma habilidade essencial que é difícil para programadores iniciantes aprenderem e desafia os educadores a ensinar [56]. A depuração é um aspecto necessário da ciência da computação que pode ser difícil para programadores novatos e experientes [11] [14]. A habilidade geralmente é autodidata e adquirida por tentativa e erro, talvez com a ajuda de um professor ou outra figura especialista [11].

A depuração é essencial para todos os programadores, mas especialmente para os novatos, que ainda estão aprendendo a sintaxe e a semântica da linguagem de programação e, portanto, são mais propensos a escrever código incorreto do que programadores mais experientes [38]. Os programadores iniciantes geralmente acham difícil executar tarefas de depuração de forma eficaz [5]. Os alunos precisam entender como o programa funciona e como executá-lo com bugs; conhecer o domínio da aplicação e a linguagem de programação; conhecer bugs (específicos) e métodos de depuração. Além disso, a exposição sistemática dos alunos a diferentes tipos de erros em um ambiente de aprendizagem têm o potencial de melhorar as habilidades de depuração [3].

No entanto, a maioria dos alunos acredita que a capacidade de depuração é devido à aptidão individual e não pode ser desenvolvida através da aprendizagem. A maioria dos alunos não utiliza ferramentas para depuração do ambiente ou sequer conhece essas ferramentas [12]. Alunos especialmente mais fracos geralmente são impotentes e aplicam



uma abordagem de tentativa e erro para erros de programação. Acontece que os erros em tempo de compilação representam um obstáculo significativo para muitos alunos. Os professores estão correndo de computador em computador, tentando ajudar [43]. As mensagens de erro ajudam os novatos a encontrar e corrigir erros, mas as mensagens do compilador geralmente são inadequadas [14].

O depurador melhora a compreensão conceitual dos alunos iniciantes, e os alunos mais baixos são os mais beneficiados com essa experiência. O depurador pode ajudar a minimizar erros lógicos e melhorar as habilidades de escrita do programa [48].

Portanto, a autossuficiência na depuração é essencial e um desafio significativo para aprender a programar. No entanto, os professores se concentram em heurísticas para erros comuns e estratégias de depuração relacionadas ao ensino de habilidades de depuração. Os professores não conduzem nenhum processo sistemático para lidar com erros. Além disso, eles não empregam aulas explícitas de ensino sobre depuração. Os professores não têm uma abordagem sistemática para ensinar depuração, pois existem apenas conceitos e materiais vagos [43].

### **Create/Criar**

O nível cognitivo *Create/Criar* agrega e une partes para criar um novo todo. Esse nível cognitivo envolve uma única produção de comunicação (tema ou discurso), um plano de operações (propostas de pesquisa) ou um conjunto de relações abstratas (esquema de classificação da informação). Combinando partes desorganizadas para formar um "todo" [1]. A Tabela 6 apresenta a adaptação do nível cognitivo *Create/Criar* para o ensino de programação, bem como as habilidades de programação cognitiva associadas aos níveis cognitivos da Taxonomia Adaptada.

**Tabela 6.** Adaptação do Nível *Create/Criar* e Associação das Habilidades Cognitivas de Programação.

<b>Habilidade</b>	<b>Definição Original</b>	<b>Adaptação</b>	<b>Associação</b>
<i>Generating/</i> Hipotetizar	<i>Come up with alternative hypotheses based on criteria.</i> Criar hipóteses alternativas com base em critérios.	Crie hipóteses alternativas com base em critérios.	-
<i>Planning/</i> Planejar	<i>Devise a procedure for accomplishing some task.</i> Elaborar um procedimento para realizar alguma tarefa.	Desenvolver um procedimento para resolver um problema.	-
<i>Producing/</i> Produzir	<i>Invent a product.</i> Inventar um produto.	Criar um programa.	Escrever

Nesse nível, os alunos criam um novo código ou algoritmo organizando mentalmente alguns elementos ou partes em um padrão ou estrutura que não estava presente antes. Para realizar este processo, os alunos recorrem a experiências de aprendizagem anteriores. Embora exija pensamento criativo por parte do aluno, isso não implica livre expressão criativa e não se restringe às demandas da tarefa ou da situação de aprendizagem.

Embora este nível inclua metas que exigem produção exclusiva, também se refere a metas que exigem produção que todos os alunos podem e farão. Ao cumprir esses objetivos, muitos alunos irão criar sua síntese de informações ou materiais para formar um novo todo, como resolver um problema.

Apesar dos níveis de processo Compreender, Aplicar e Analisar possam envolver a detecção de relacionamentos entre os elementos apresentados, Criar é diferente porque envolve a construção de um produto original. Ao contrário do nível Criar, os outros níveis envolvem o trabalho com um determinado conjunto de elementos que fazem parte de um determinado todo; eles são parte de uma estrutura maior que o aluno está tentando entender. Por outro lado, o aluno deve recorrer a elementos de muitas fontes e juntá-los em uma nova estrutura ou padrão em relação ao seu conhecimento anterior. A criação de um novo programa ou algoritmo pode ser observada, ou seja, mais do que o material inicial do aluno. Uma tarefa que requer Criar provavelmente exigirá aspectos de cada uma das categorias de processos cognitivos anteriores até certo ponto, mas não necessariamente na ordem em que estão listados na Taxonomia.

Assim, o processo criativo começa com uma fase divergente em que o aluno pensa em várias soluções possíveis ao tentar entender a tarefa (gerar). Segue-se uma fase convergente, na qual o aluno concebe um método de solução e o transforma em um plano de ação (planejamento). Por fim, o aluno executa o plano enquanto constrói a solução (produzindo).

As habilidades cognitivas do nível *Create/Criar* incluem:

- **Generating/Hipotetizar** é a capacidade do aluno de criar hipóteses alternativas com base em critérios (por exemplo, hipotetizar que uma nova combinação de algoritmos resolverá o problema).
- **Planning/Planejar** é a capacidade do aluno de desenvolver um procedimento para realizar uma tarefa (por exemplo, planejar um algoritmo, processo ou estratégia alternativa para um problema);
- **Producing/Produzir** é a capacidade do aluno de criar um programa (por exemplo, construir um programa usando algoritmos inventados) [2] [7] [8] [9] [10] [16] [17] [22] [23] [24] [25] [26] [28] [29] [31] [32] [33] [34] [35] [39] [40] [41] [42] [44] [45] [47] [49] [50] [52] [53] [58] [57] [59] [60] [61] [62].

## REFERÊNCIAS

[1] Lorin W Anderson and David R Krathwohl. A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives. Longman, 2001.

[2] Affandy, N. S. Herman, S. B. Salam, and E. Noersasongko. A study of tracing and writing performance of novice students in introductory programming. In Proceedings of the International Conference on Software Engineering and Computer Systems (ICSECS). Springer, 2011.

[3] B. S. Alqadi and J. I. Maletic. An empirical study of debugging patterns among novices programmers. In Proceedings of the Special Interest Group on Computer Science Education (SIGCSE). ACM Seattle, WA, United States, 2017.

[4] A. P. Ambrosio, F. M. Costa, L. Almeida, A. Franco, and J. Macedo. Identifying cognitive abilities to improve cs1 outcome. In Proceedings of the Frontiers in Education Conference (FIE). IEEE, Rapid City, SD, United States, 2011.

- [5] P. Ardimento, M. L. Bernardi, M. Cimitile, and G. Ruvo. Reusing bugged source code to support novice programmers in debugging tasks. *ACM Transactions on Computing Education (TOCE)*, 20(1), 2019.
- [6] M. Azadmanesh and M. Hauswirth. Concept-driven generation of intuitive explanations of program execution for a visual tutor. In *Proceedings of the Working Conference on Software Visualization (VISOFT)*. IEEE, Shanghai, China, 2017.
- [7] G. R. Bergersen, D. I. Sjøberg, and T. Dyba. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering*, 40(12), 2014.
- [8] M. Berges and P. Hubwieser. Evaluation of source code with item response theory. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Vilnius, Lithuania, 2015.
- [9] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. IEEE, Florence, Italy, 2015.
- [10] T. Busjahn and C. Schulte. The use of code reading in teaching programming. In *Proceedings of the Koli Calling International Conference on Computing Education Research*. ACM Koli, Finland, 2013.
- [11] E. Carter and G. D. Blank. A tutoring system for debugging: Status report. *Journal of Computing Sciences in Colleges (JCSC)*, 28(3), 2013.
- [12] M. W. Chen, C. C. Wu, and Y. T. Lin. Novices' debugging behaviors in vb programming. In *Proceedings of the Learning and Teaching in Computing and Engineering (LaTiCE)*. IEEE, Macau, China, 2013.
- [13] Q. Cutts, M. Barr, M. B. Ada, P. Donaldson, S. Draper, J. Parkinson, J. Singer, and L. Sundin. Experience report: Thinkathon – countering an got it working mentality with pencil-and-paper exercises. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Aberdeen, Scotland, United Kingdom, 2019.
- [14] P. Denny, A. Luxton-Reilly, and D. Carpenter. Enhancing syntax error messages appears ineffectual. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Uppsala, Sweden, 2014.
- [15] P. Donaldson and Q. Cutts. Flexible low-cost activities to develop novice code comprehension skills in schools. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE)*. ACM Potsdam, Germany, 2018.
- [16] R. S. Duran, J. Rybicki, A. Hellas, and S. Suoranta. Towards a common instrument for measuring prior programming knowledge. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Aberdeen, Scotland, United Kingdom, 2019.
- [17] S. Elnaffar. Using software metrics to predict the difficulty of code writing questions. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)*. IEEE, Dubai, United Arab Emirates, 2016.
- [18] K. L. Eranki and K. M. Moudgalya. Application of program slicing technique to improve novice programming competency in spoken tutorial workshops. In *Proceedings of the International Conference on Technology for Education (T4E)*. IEEE, Clappana, India, 2013.
- [19] K. L. Eranki and K. M. Moudgalya. An integrated approach to build programming competencies through spoken tutorial workshops. In *Proceedings of the International Conference on Technology for Education (T4E)*. IEEE, Kharagpur, India, 2014.
- [20] K. L. N. Eranki and K. M. Moudgalya. Evaluation of programming competency using student error patterns. In *Proceedings of the International Conference on Learning and Teaching in Computing and Engineering*. IEEE, 2015.
- [21] K. L. N. Eranki and K. M. Moudgalya. Program slicing technique: A novel approach to improve programming skills in novice learners. In *Proceedings of the Conference on Information Technology Education (SIGITE)*. ACM Boston, MA, United States, 2016.

- [22] S. Esper, S. R. Foster, W. G. Griswold, C. Herrera, and W. Snyder. Codespells: Bridging educational language features with industry-standard languages. In Proceedings of the Koli Calling International Conference on Computing Education Research. ACM Koli, Finland, 2014.
- [23] G. V. F. Fabric, A. Mitrovic, and K. Neshatian. Adaptive problem selection in a mobile python tutor. In Proceedings of the Conference on User Modeling, Adaptation and Personalization (UMAP). ACM, Singapore, Singapore, 2018.
- [24] G. V. F. Fabric, A. Mitrovic, and K. Neshatian. Investigating the effects of learning activities in a mobile python tutor for targeting multiple coding skills. *Research and Practice in Technology Enhanced Learning*, 13(1), 2018.
- [25] I. Fronza, A. Hellas, P. Ihanola, and T. Mikkonen. An exploration of cognitive shifting in writing code. In Proceedings of the Conference on Global Computing Education (CompEd). ACM Chengdu, Sichuan, China, 2019.
- [26] D. Ginat and E. Menashe. Solo taxonomy for assessing novices algorithmic design. In Proceedings of the Special Interest Group on Computer Science Education (SIGCSE). ACM Kansas City, MO, United States, 2015.
- [27] J. M. Griffin. Learning by taking apart: Deconstructing code by reading, tracing, and debugging. In Proceedings of the Conference on Information Technology Education (SIGITE). ACM Boston, MA, United States, 2016.
- [28] B. Harrington and N. Cheng. Tracing vs. writing code: Beyond the learning hierarchy. In Proceedings of the Special Interest Group on Computer Science Education (SIGCSE). ACM Baltimore, MD, United States, 2018.
- [29] K. V. Hausswolff and A. Eckerdal. Measuring programming knowledge in a research context. In Proceedings of the Frontiers in Education Conference (FIE). IEEE, San Jose, CA, United States, 2018.
- [30] M. Hertz and M. Jump. Trace-based teaching in early programming courses. In Proceedings of the Special Interest Group on Computer Science Education (SIGCSE). ACM Denver, Colorado, United States, 2013.
- [31] D. M. Hoffman, M. Lu, and T. Pelton. A web-based generation and delivery system for active code reading. In Proceedings of the Special Interest Group on Computer Science Education (SIGCSE). ACM Dallas, Texas, United States, 2011.
- [32] V. Isomotton, A. Nylen, and V. Tirronen. Writing to learn programming? a single case pilot study. In Proceedings of the Koli Calling International Conference on Computing Education Research. ACM Joensuu, Finland, 2016.
- [33] C. Izu, A. Weerasinghe, and C. Pope. A study of code design skills in novice programmers using the solo taxonomy. In Proceedings of the International Computing Education Research (ICER). ACM Melbourne, VIC, Australia, 2016.
- [34] N. Kasto, J. L. Whalley, A. Philpott, and D. Whalley. Solution spaces. In Proceedings of the Australasian Computing Education Conference (ACE). ACM Auckland, New Zealand, 2014.
- [35] A. N. Kumar. A mid-career review of teaching computer science i. In Proceedings of the Special Interest Group on Computer Science Education (SIGCSE). ACM Denver, Colorado, United States, 2013.
- [36] S. Lee, A. Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, and H. Lim. Comparing programming language comprehension between novice and expert programmers using eeg analysis. In Proceedings of the International Conference on Bioinformatics and Bioengineering (BIBE). IEEE, Taichung, Taiwan, 2016.
- [37] A. Luxton-Reilly, E. McMillan, E. Stevenson, E. Tempero, and P. Denny. Ladebug: An online tool to help novice programmers improve their debugging skills. In Proceedings of the Innovation and Technology in Computer Science Education (ITICSE). ACM Larnaca, Cyprus, 2018.
- [38] Andrew Luxton-Reilly, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. Introductory programming: a systematic literature review. In Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, pages 55–106, 2018.

- [39] R. Mathew, S. I. Malik, and R. M. Tawafak. Teaching problem solving skills using an educational game in a computer programming course. *Informatics in Education*, 18(2), 2019.
- [40] A. Matthiasdottir and H. Arnalds. E-assessment: Students point of view. In *Proceedings of the Computer Systems and Technologies (CompSysTech)*. ACM Palermo, Italy, 2016.
- [41] E. McArdle, J. Holdsworth, and I. Lee. Assessing the usability of students object-oriented language with first-year it students: A case study. In *Proceedings of the Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration (OzCHI)*. ACM Adelaide, Australia, 2013.
- [42] B. Mendoza and L. Zavala. An intervention strategy to hone students code understanding skills. *Journal of Computing Sciences in Colleges (JCSC)*, 33(3), 2018.
- [43] T. Michaeli and R. Romeike. Current status and perspectives of debugging in the k12 classroom: A qualitative study. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)*. IEEE, Dubai, United Arab Emirates, 2019.
- [44] L. Murphy, S. Fitzgerald, R. Lister, and R. McCauley. Ability to explain in plain english linked to proficiency in computer-based programming. In *Proceedings of the International Computing Education Research (ICER)*. ACM Auckland, New Zealand, 2012.
- [45] T. Na, N. Funabiki, K. K. Zaw, N. Ishihara, S. Matsumoto, and W. C. Kao. A fill-in-blank problem workbook for java programming learning assistant system. *International Journal of Web Information Systems*, 13(2), 2017. [57] Peter Naur. Programming as theory building. *Microprocessing and microprogramming*, 15(5):253–261, 1985.
- [46] G. L. Nelson, A. Hu, B. Xie, and A. J. Ko. Towards validity for a formative assessment for language-specific program tracing skills. In *Proceedings of the Koli Calling International Conference on Computing Education Research*, 2019.
- [47] G. L. Nelson, B. Xie, and A. J. Ko. Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the International Computing Education Research (ICER)*. ACM Tacoma, WA, United States, 2017.
- [48] M. Nirgude. Debugger tool to improve conceptual understanding of programming language of novice learners. In *Proceedings of the International Conference on Technology for Education (T4E)*. IEEE, Kharagpur, India, 2013.
- [49] D. Parsons, K. Wood, and P. Haden. What are we doing when we assess programming? In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Sydney, Australia, 2015.
- [50] A. Petersen, M. Craig, and D. Zingaro. Reviewing cs1 exam question content. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Dallas, Texas, United States, 2011.
- [51] K. Sanders, M. Ahmadzadeh, T. Clear, S. H. Edwards, M. Goldweber, C. Johnson, R. Lister, R. McCartney, E. Patitsas, and J. Spacco. The canterbury questionbank: Building a repository of multiple-choice cs1 and cs2 questions. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Canterbury, United Kingdom, 2013.
- [52] J. Sheard, J. Dermoudy, D. D’Souza, M. Hu, and D. Parsons. Benchmarking a set of exam questions for introductory programming. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, New Zealand, 2014.
- [53] N. Shi, P. Zhang, and X. Sun. B-learning on novice programmer learning with roles of variables. *International Journal of Simulation Systems, Science Technology (IJSSST)*, 17(32), 2016.
- [54] X. Suo. Toward more effective strategies in teaching programming for novice students. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Hong Kong, China, 2012.
- [55] A. Vihavainen, C. S. Miller, and A. Settle. Benefits of self-explanation in introductory programming. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Kansas City, MO, United States, 2015.

- [56] X. S. Wang and J. Souders. Improving debugging education through applied learning. *Journal of Computing Sciences in Colleges (JCSC)*, 27(3), 2012.
- [57] J. Whalley, T. Clear, R. Phil, and T. Errol. Salient elements in novice solutions to code writing problems. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, Perth, Australia, 2011.
- [58] J. Whalley and N. Kasto. How difficult are novice code writing tasks? a software metrics approach. In *Proceedings of the Australasian Computing Education Conference (ACE)*. ACM Auckland, New Zealand, 2014.
- [59] B. Xie, D. Loksa, G. L. Nelson, M. J. Davidson, D. Dong, H. Kwik, A. H. Tan, L. Hwa, M. Li, and A. J. Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3), 2019.
- [60] P. S. Yanagi, T. S. Indi, and M. A. Nirgude. Enhancing the cognitive level of novice learners using effective program writing skills. In *Proceedings of the International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. IEEE, Logon, United Kingdom, 2016.
- [61] L. Zavala and Mendoza B. Precursor skills to writing code. *Journal of Computing Sciences in Colleges (JCSC)*, 32(3), 2017.
- [62] D. Zingaro, A. Petersen, and Craig M. Stepping up to integrative questions on cs1 exams. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Raleigh, North Carolina, United States, 2012.

## Appendix G

# Inspection of Bloom's Revised Taxonomy for Measuring and Fostering Cognitive Programming Skills

Prezado(a) juiz(a),

Solicitamos-lhe a avaliação da adaptação do nível cognitivo da Taxonomia Revisada de Bloom para ensino de programação. O mesmo foi criado para sequenciar as habilidades de programação, no domínio cognitivo, a fim melhorar o aprendizado dos alunos. Sua avaliação é muito importante para a validação desta etapa e continuidade da pesquisa. Para o processo de inspeção deste artefato é necessário a leitura do material em anexo. Em seguida, considere as seguintes definições para cada item da escala seleccionada para a realização da avaliação. Apresentamos uma escala do tipo LIKERT, com as opções para você realizar sua avaliação, considerando 1 a pior nota e 5 a melhor nota a ser atribuída em cada uma das habilidades.

- 1—Não adequado: Nada adequado, não adaptado, não correspondendo em nada ao objetivo proposto para o ensino de programação;
- 2—Pouco adequado: 25% adequado, adaptado, correspondendo muito pouco ao objetivo proposto para o ensino de programação;
- 3—Moderadamente adequado: 50% adequado, adaptado, moderadamente

correspondente ao objetivo proposto para o ensino de programação;

- 4—Muito adequado: 75% adequado, adaptado, correspondendo intensamente ao objetivo proposto para o ensino de programação.
- 5—Completamente adequado: 100% adequado, adaptado, correspondendo perfeitamente ao objetivo proposto para o ensino de programação.

Em relação a adaptação, indique o seu grau de concordância com a adaptação do objetivo da habilidade do nível cognitivo da Taxonomia Revisada de Bloom para ensino de programação.

<b>Skill</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Reconhecer					
Recordar					
Interpretar					
Exemplificar					
Classificar					
Resumir					
Inferir					
Comparar					
Explicar					
Executar					
Implementar					
Diferenciar					
Organizar					
Atribuir					
Verificar					
Criticar					
Hipotetizar					
Planejar					
Produzir					

Indique o seu grau de concordância com a associação (se houver) da habilidade cognitiva de programação levantada na literatura com a definição adaptativa da habilidade do Nível Cognitivo da Taxonomia Revisada de Bloom.

<b>Skill</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Interpretar					
Exemplificar					
Classificar					
Resumir					
Inferir					
Comparar					
Explicar					
Organizar					
Verificar					
Produzir					



# **Appendix H**

## **Informed Consent Form**

# Termo de Consentimento Livre Esclarecido

Você está sendo convidado(a) a participar do projeto de pesquisa “**Mensuração de habilidades cognitivas introdutórias de programação por meio de uma avaliação adaptativa informatizada**” coordenado pelos professores Dr. Wilkerson de Lucena Andrade e Dr. João Arthur Brunet Monteiro vinculados ao Programa de Pós Graduação da Universidade Federal de Campina Grande, a quem poderar contatar/consultar a qualquer momento que julgar necessário através dos e-mails {wilkerson, joao.arthur}@computacao.ufcg.edu.br. O documento abaixo contém todas as informações necessárias sobre a pesquisa que estamos fazendo. Sua colaboração neste estudo será de muita importância para nós, mas se desistir a qualquer momento, isso não causará nenhum prejuízo a você.

Eu, ....., portador da Cédula de identidade, RG ....., e inscrito no CPF....., abaixo assinado(a), concordo de livre e espontânea vontade em participar como voluntário(a) deste estudo. Declaro que obtive todas as informações necessárias, bem como todos os eventuais esclarecimentos quanto às dúvidas por mim apresentadas.

Estou ciente que:

- Este estudo tem por objetivo mensurar as habilidades cognitivas introdutórias relacionadas à programação de alunos em cursos técnicos e superiores com intuito de analisar suas limitações e descrever métodos para aprimorar o processo de ensino-aprendizagem.
- Serei submetido aos seguintes procedimentos: i) avaliar a usabilidade de instrumentos psicométricos; ii) elencar requisitos de instrumentos por meio de entrevistasemi-estruturada; iii) elaborar itens para instrumentos psicométricos; e iv) avaliar a reação dos indivíduos no decorrer da aplicação de instrumentos psicométricos, medindo assim o grau de satisfação ao utilizá-los. Na qual serei beneficiado(a) na colaboração/produção de instrumentos que poderão me auxiliar futuramente em sala de aula.
- Caso sinta riscos por está intimidado(a) ou desconfortado(a) durante a participação na pesquisa, poderei desistir a qualquer momento, retirando meu consentimento, sem que isso me traga nenhum prejuízo ou penalidade e receberei apoio necessário a questão apresentada.
- Todas as informações obtidas serão sigilosas e meu nome não será identificado em nenhum momento. Os dados serão guardados em local seguro e a divulgação dos resultados será feita de maneira que não permita a minha identificação.
- Se eu tiver algum gasto decorrente de minha participação na pesquisa, serei ressarcido, caso solicite. Em qualquer momento, se eu sofrer algum dano comprovadamente decorrente desta pesquisa, serei indenizado.
- Caso me sinta prejudicado (a) por participar desta pesquisa, poderei recorrer ao Comitê de Ética em Pesquisas com Seres Humanos – CEP, do Hospital Universitário Alcides Carneiro - HUAC, situado a Rua: Dr. Carlos Chagas, s/ n, São José, CEP: 58401 – 490, Campina Grande-PB, Tel: 2101 – 5545, E-mail: cep@huac.ufcg.edu.br; Conselho Regional de Medicina da Paraíba e a Delegacia Regional de Campina Grande.
- Atesto recebimento de uma via assinada deste Termo de Consentimento Livre e Esclarecido, conforme recomendações da Comissão Nacional de Ética em Pesquisa (CONEP). Outros esclarecimentos sobre esta pesquisa, poderei entrar em contato com o pesquisador principal Ms. Jucelio Soares dos Santos, Rua Severino Pimentel, n. 785D - Liberdade - Campina Grande - PB, tel. (83) 99694.0954.

Campina Grande - PB, .... de ..... de 20 ....

---

(Assinatura do participante)

---

(Testemunha 1 | Nome/RG/Telefone)

---

(Testemunha 2 | Nome/RG/Telefone)

---

Ms. Jucelio Soares dos Santos  
(Responsável pelo projeto)

# Appendix I

## Item Bank Inspection

Prezado(a) juiz(a),

A seguir, apresentamos um exemplo dos formulários de inspeção.

Solicitamos-lhe a avaliação dos itens a seguir a fim de melhorar a sua qualidade. Os itens foram desenvolvidos a fim de mensurar habilidades cognitivas dos alunos em programação. Sua avaliação é muito importante para a validação desta etapa. Para o processo de inspeção das questões é necessário assistir o vídeo que contém a definição da habilidade, bem como avaliar se cada item mensura esta habilidade. Para tanto, deverá considerar as seguintes definições para cada item da escala seleccionada para a realização da avaliação. Apresentamos uma escala do tipo LIKERT, com as opções para você realizar sua avaliação, considerando 1 a pior nota e 5 a melhor nota a ser atribuída em cada uma das habilidades.

- 1—Não adequado: Nada adequado, não adaptado, não correspondendo em nada ao objetivo proposto para a habilidade inspeccionada;
- 2—Pouco adequado: 25% adequado, adaptado, correspondendo muito pouco ao objetivo proposto para a habilidade inspeccionada;
- 3—Moderadamente adequado: 50% adequado, adaptado, moderadamente correspondente ao objetivo proposto para a habilidade inspeccionada;
- 4—Muito adequado: 75% adequado, adaptado, correspondendo intensamente ao objetivo proposto para a habilidade inspeccionada;

- 5—Completamente adequado: 100% adequado, adaptado, correspondendo perfeitamente ao objetivo proposto para a habilidade inspeccionada.

Algumas observações:

- Caso você avalie o item em notas 3 e 4, por favor informe o que deve ser ajustado. Os itens podem conter os seguintes erros de: i) sintaxe em relação a linguagem Python; ii) escrita, concordância ou até mesmo má compreensão no que foi apresentado; iii) contém mais de uma alternativa correta; iv) não contém alternativa correta; e v) demais sugestões;
- Caso você avalie o item em notas 1 e 2, compreenderemos que o item não está adequado para avaliar esta habilidade e não tem como ser ajustado;
- Ao atribuir nota 5, compreenderemos que você está de acordo com todas as especificações do item, bem como concorda com a alternativa correta entre as opções. Assim, você certifica que este item está totalmente apto para ser utilizado em avaliações.

**Definição da habilidade e exemplo de item:** <https://youtu.be/7Z-jSFAnQxE>

**Informe o avaliador:** .....

Item	1	2	3	4	5
#REMREC01ES					
#REMREC02ES					
#REMREC03ES					
#REMREC04ES					
#REMREC05ES					
#REMREC06ES					
#REMREC07ES					
#REMREC08ES					
#REMREC09ES					
#REMREC10ES					

Algum item precisa de ajuste? Se você avaliou algum item em notas 3 e 4 indica que o item apresenta algum erro e precisa ser ajustado. Informe o item e indique o ajuste que ele precisa obter.

# **Appendix J**

## **Informed Consent Form**

# Termo de Consentimento Livre Esclarecido

Você está sendo convidado(a) a participar do projeto de pesquisa “**Mensuração de habilidades cognitivas introdutórias de programação por meio de uma avaliação adaptativa informatizada**” coordenado pelos professores Dr. Wilkerson de Lucena Andrade e Dr. João Arthur Brunet Monteiro vinculados ao Programa de Pós Graduação da Universidade Federal de Campina Grande, a quem poderar contatar/consultar a qualquer momento que julgar necessário através dos e-mails {wilkerson, joao.arthur}@computacao.ufcg.edu.br. O documento abaixo contém todas as informações necessárias sobre a pesquisa que estamos fazendo. Sua colaboração neste estudo será de muita importância para nós, mas se desistir a qualquer momento, isso não causará nenhum prejuízo a você.

Eu, ....., portador da Cédula de identidade, RG ....., e inscrito no CPF....., abaixo assinado(a), concordo de livre e espontânea vontade em participar como voluntário(a) deste estudo. Declaro que obtive todas as informações necessárias, bem como todos os eventuais esclarecimentos quanto às dúvidas por mim apresentadas.

Estou ciente que:

- Este estudo tem por objetivo mensurar as habilidades cognitivas introdutórias relacionadas à programação de alunos em cursos técnicos e superiores com intuito de analisar suas limitações e descrever métodos para aprimorar o processo de ensino-aprendizagem.
- Serei submetido aos seguintes procedimentos: i) fornecer respostas à exercícios durante a aplicação de instrumentos psicométricos; e, ii) avaliar o seu nível de satisfação ao usar os instrumentos psicométricos. Na qual serei beneficiado por avaliar minhas habilidades introdutórias em Programação.
- Caso sinta riscos por está intimidado(a) ou desconfortado(a) durante a participação na pesquisa, poderei desistir a qualquer momento, retirando meu consentimento, sem que isso me traga nenhum prejuízo ou penalidade e receberei apoio necessário a questão apresentada.
- Todas as informações obtidas serão sigilosas e meu nome não será identificado em nenhum momento. Os dados serão guardados em local seguro e a divulgação dos resultados será feita de maneira que não permita a minha identificação.
- Se eu tiver algum gasto decorrente de minha participação na pesquisa, serei ressarcido, caso solicite. Em qualquer momento, se eu sofrer algum dano comprovadamente decorrente desta pesquisa, serei indenizado.
- Caso me sinta prejudicado (a) por participar desta pesquisa, poderei recorrer ao Comitê de Ética em Pesquisas com Seres Humanos – CEP, do Hospital Universitário Alcides Carneiro - HUAC, situado a Rua: Dr. Carlos Chagas, s/ n, São José, CEP: 58401 – 490, Campina Grande-PB, Tel: 2101 – 5545, E-mail: cep@huac.ufcg.edu.br; Conselho Regional de Medicina da Paraíba e a Delegacia Regional de Campina Grande.
- Atesto recebimento de uma via assinada deste Termo de Consentimento Livre e Esclarecido, conforme recomendações da Comissão Nacional de Ética em Pesquisa (CONEP). Outros esclarecimentos sobre esta pesquisa, poderei entrar em contato com o pesquisador principal Ms. Jucelio Soares dos Santos, Rua Severino Pimentel, n. 785D - Liberdade - Campina Grande - PB, tel. (83) 99694.0954.

Campina Grande - PB, .... de ..... de 20 ....

---

(Assinatura do participante)

---

(Testemunha 1 | Nome/RG/Telefone)

---

(Testemunha 2 | Nome/RG/Telefone)

---

Ms. Jucelio Soares dos Santos  
(Responsável pelo projeto)

# **Appendix K**

## **Items Psychometric Properties**

# 1 Recognizing

In Table 1, we present the items that make up the recognizing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
REMREC01ES	0.912	0.619	0.200	0.470	0.234
REMREC02ES	1.702	0.000	0.200	0.380	0.044
REMREC03ES	0.847	-0.189	0.152	0.560	0.296
REMREC04ES	0.953	0.350	0.167	0.480	0.289
REMREC05ES	1.346	1.139	0.187	0.340	0.225
REMREC06ES	1.193	-2.311	0.161	0.900	0.280
REMREC07ES	1.331	-0.466	0.160	0.620	0.336
REMREC08ES	1.007	0.218	0.157	0.490	0.318
REMREC09ES	1.181	-1.083	0.192	0.750	0.315
REMREC10ES	1.016	-1.587	0.149	0.790	0.272
REMREC01CS	1.746	0.765	0.180	0.360	0.331
REMREC02CS	1.530	0.403	0.178	0.440	0.319
REMREC03CS	1.497	0.027	0.175	0.520	0.368
REMREC04CS	1.811	-0.084	0.162	0.530	0.405
REMREC05CS	1.783	-0.382	0.166	0.610	0.410
REMREC06CS	1.130	-0.952	0.166	0.710	0.332
REMREC07CS	1.057	-1.397	0.176	0.780	0.282
REMREC08CS	1.963	1.177	0.144	0.250	0.277
REMREC09CS	1.748	-2.087	0.156	0.920	0.330
REMREC10CS	0.890	0.237	0.164	0.500	0.289
REMREC01CI	2.785	1.359	0.329	0.410	0.068
REMREC02CI	1.932	-1.910	0.153	0.910	0.359
REMREC03CI	1.782	0.070	0.109	0.450	0.476
REMREC04CI	1.064	-1.342	0.148	0.760	0.331
REMREC05CI	0.926	-0.054	0.157	0.540	0.322
REMREC06CI	1.434	-0.793	0.152	0.690	0.383
REMREC07CI	1.533	0.778	0.157	0.350	0.316
REMREC08CI	0.695	0.256	0.174	0.520	0.187
REMREC09CI	0.795	0.540	0.159	0.460	0.243
REMREC10CI	1.495	-0.342	0.214	0.630	0.298
REMREC01FU	1.336	-1.255	0.189	0.790	0.329
REMREC02FU	1.565	-0.626	0.156	0.660	0.392
REMREC03FU	1.333	-0.254	0.139	0.560	0.366
REMREC04FU	1.063	-1.752	0.179	0.830	0.256
REMREC05FU	1.074	-0.022	0.158	0.530	0.320
REMREC06FU	2.512	0.892	0.174	0.300	0.305
REMREC07FU	1.366	2.005	0.246	0.320	0.090
REMREC08FU	1.591	0.553	0.159	0.390	0.383
REMREC09FU	1.161	0.072	0.159	0.510	0.325
REMREC10FU	1.364	0.275	0.135	0.440	0.412
<b>Mean</b>	1.386	-0.179	0.171	0.561	0.303
<b>Standard deviation</b>	0.445	0.997	0.034	0.179	0.088

Table 1: Recognizing skill - all items calibrated.



## 2 Recalling

In Table 2, we present the items that make up the recalling skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
REMREL01ES	1.036	-1.020	0.185	0.433	0.308
REMREL02ES	1.215	0.499	0.133	0.496	0.404
REMREL03ES	1.589	0.755	0.159	0.488	0.391
REMREL04ES	1.189	1.122	0.236	0.492	0.057
REMREL05ES	0.812	-0.010	0.151	0.496	0.298
REMREL06ES	1.191	-0.313	0.158	0.485	0.328
REMREL07ES	1.680	0.188	0.212	0.499	0.271
REMREL08ES	0.975	0.416	0.158	0.500	0.323
REMREL09ES	0.995	-1.447	0.149	0.407	0.279
REMREL10ES	1.204	-2.155	0.161	0.300	0.284
REMREL01CS	0.988	0.574	0.154	0.498	0.309
REMREL02CS	1.267	0.225	0.169	0.500	0.364
REMREL03CS	2.184	0.949	0.195	0.480	0.339
REMREL04CS	1.024	-1.264	0.176	0.414	0.293
REMREL05CS	1.828	0.123	0.170	0.499	0.418
REMREL06CS	1.553	-0.212	0.167	0.488	0.406
REMREL07CS	1.084	-0.807	0.166	0.454	0.336
REMREL08CS	1.060	0.445	0.158	0.500	0.337
REMREL09CS	1.293	1.392	0.188	0.474	0.228
REMREL10CS	1.754	-1.952	0.156	0.271	0.333
REMREL01CI	1.652	0.275	0.110	0.497	0.478
REMREL02CI	1.373	-0.637	0.154	0.462	0.384
REMREL03CI	1.890	1.450	0.146	0.433	0.281
REMREL04CI	1.307	-0.185	0.210	0.483	0.301
REMREL05CI	0.806	0.454	0.162	0.500	0.288
REMREL06CI	1.932	-1.778	0.153	0.286	0.361
REMREL07CI	0.946	0.125	0.158	0.498	0.328
REMREL08CI	1.340	1.035	0.150	0.477	0.318
REMREL09CI	1.011	-1.222	0.148	0.427	0.337
REMREL10CI	1.417	0.634	0.178	0.496	0.320
REMREL01FU	0.994	-1.674	0.178	0.376	0.255
REMREL02FU	1.224	-0.061	0.143	0.496	0.366
REMREL03FU	1.180	0.297	0.170	0.500	0.329
REMREL04FU	2.937	1.121	0.182	0.458	0.311
REMREL05FU	0.672	0.454	0.174	0.500	0.188
REMREL06FU	1.085	0.679	0.195	0.499	0.288
REMREL07FU	0.986	0.182	0.159	0.499	0.320
REMREL08FU	0.695	2.654	0.195	0.466	0.091
REMREL09FU	1.277	-1.130	0.188	0.407	0.338
REMREL10FU	1.442	-0.474	0.157	0.474	0.397
<b>Mean</b>	1.302	-0.007	0.168	0.460	0.315
<b>Standard deviation</b>	0.437	1.038	0.023	0.060	0.077

Table 2: Recalling skill - all items calibrated.

### 3 Interpreting

In Table 3, we present the items that make up the interpreting skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDINT01ES	1.046	-0.985	0.186	0.750	0.307
UNDINT02ES	1.694	-1.972	0.156	0.920	0.330
UNDINT03ES	0.626	0.515	0.173	0.520	0.187
UNDINT04ES	0.785	0.033	0.152	0.560	0.299
UNDINT05ES	1.414	-0.123	0.218	0.630	0.304
UNDINT06ES	1.169	1.506	0.184	0.340	0.225
UNDINT07ES	1.655	0.308	0.108	0.450	0.477
UNDINT08ES	0.926	0.474	0.158	0.490	0.321
UNDINT09ES	1.502	0.824	0.159	0.390	0.392
UNDINT10ES	1.212	-1.148	0.185	0.790	0.340
UNDINT01CS	1.863	-1.794	0.153	0.910	0.358
UNDINT02CS	1.005	-1.414	0.149	0.790	0.278
UNDINT03CS	1.189	-0.273	0.160	0.620	0.331
UNDINT04CS	3.453	1.165	0.246	0.360	0.287
UNDINT05CS	1.255	0.254	0.167	0.520	0.364
UNDINT06CS	1.973	1.039	0.194	0.360	0.341
UNDINT07CS	0.874	0.180	0.159	0.540	0.331
UNDINT08CS	1.041	0.496	0.159	0.480	0.341
UNDINT09CS	1.468	0.681	0.183	0.440	0.324
UNDINT10CS	1.032	-0.802	0.166	0.710	0.338
UNDINT01CI	0.963	-1.297	0.174	0.780	0.291
UNDINT02CI	1.138	-2.216	0.161	0.900	0.282
UNDINT03CI	1.145	0.329	0.167	0.510	0.332
UNDINT04CI	1.381	-0.604	0.155	0.690	0.382
UNDINT05CI	1.285	0.524	0.135	0.440	0.407
UNDINT06CI	1.021	0.218	0.161	0.530	0.320
UNDINT07CI	1.057	0.736	0.196	0.470	0.292
UNDINT08CI	2.566	1.206	0.179	0.300	0.314
UNDINT09CI	0.936	0.640	0.227	0.520	0.246
UNDINT10CI	1.543	-0.181	0.167	0.610	0.408
UNDINT01FU	1.224	-0.031	0.142	0.560	0.368
UNDINT02FU	1.481	-0.435	0.158	0.660	0.398
UNDINT03FU	0.757	0.517	0.162	0.500	0.286
UNDINT04FU	1.026	-1.609	0.180	0.830	0.254
UNDINT05FU	1.844	0.163	0.172	0.530	0.417
UNDINT06FU	0.896	0.662	0.154	0.460	0.310
UNDINT07FU	1.019	-1.190	0.148	0.760	0.336
UNDINT08FU	2.073	1.438	0.146	0.250	0.288
UNDINT09FU	1.114	1.169	0.232	0.410	0.056
UNDINT10FU	1.384	1.069	0.152	0.350	0.320
<b>Mean</b>	1.326	0.002	0.170	0.566	0.320
<b>Standard deviation</b>	0.526	0.990	0.027	0.175	0.069

Table 3: Interpreting skill - all items calibrated.

## 4 Exemplifying

In Table 4, we present the items that make up the exemplifying skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDEXE01ES	0.973	-1.036	0.182	0.750	0.307
UNDEXE02ES	1.248	-1.116	0.185	0.790	0.350
UNDEXE03ES	1.041	-0.006	0.145	0.560	0.355
UNDEXE04ES	1.742	-1.931	0.157	0.920	0.322
UNDEXE05ES	1.528	0.235	0.169	0.520	0.385
UNDEXE06ES	1.171	-0.176	0.202	0.630	0.303
UNDEXE07ES	1.536	0.790	0.152	0.390	0.400
UNDEXE08ES	1.134	1.562	0.185	0.340	0.223
UNDEXE09ES	0.968	0.459	0.156	0.490	0.329
UNDEXE10ES	1.642	0.314	0.107	0.450	0.482
UNDEXE01CS	1.915	-1.756	0.154	0.910	0.349
UNDEXE02CS	0.983	-1.421	0.150	0.790	0.286
UNDEXE03CS	1.127	-0.285	0.156	0.620	0.333
UNDEXE04CS	1.120	0.118	0.151	0.540	0.377
UNDEXE05CS	1.456	0.693	0.183	0.440	0.333
UNDEXE06CS	1.166	0.284	0.168	0.520	0.359
UNDEXE07CS	1.051	0.488	0.156	0.480	0.346
UNDEXE08CS	1.899	1.050	0.190	0.360	0.347
UNDEXE09CS	1.301	1.175	0.181	0.370	0.300
UNDEXE10CS	1.046	-0.787	0.165	0.710	0.340
UNDEXE01CI	1.028	0.229	0.161	0.530	0.325
UNDEXE02CI	1.159	-2.177	0.161	0.900	0.275
UNDEXE03CI	1.014	0.449	0.215	0.530	0.279
UNDEXE04CI	1.327	-0.601	0.157	0.690	0.380
UNDEXE05CI	2.759	1.252	0.188	0.300	0.315
UNDEXE06CI	1.527	-0.160	0.171	0.610	0.408
UNDEXE07CI	1.163	0.715	0.200	0.470	0.299
UNDEXE08CI	1.212	0.544	0.132	0.440	0.407
UNDEXE09CI	1.073	0.364	0.169	0.510	0.328
UNDEXE10CI	1.001	-1.251	0.174	0.780	0.302
UNDEXE01FU	0.943	-1.703	0.177	0.830	0.248
UNDEXE02FU	1.030	-1.173	0.147	0.760	0.335
UNDEXE03FU	1.217	-0.011	0.145	0.560	0.375
UNDEXE04FU	1.922	0.177	0.175	0.530	0.417
UNDEXE05FU	1.216	0.508	0.148	0.460	0.378
UNDEXE06FU	0.775	0.519	0.162	0.500	0.295
UNDEXE07FU	1.247	1.153	0.151	0.350	0.309
UNDEXE08FU	1.868	1.529	0.146	0.250	0.279
UNDEXE09FU	1.393	-0.443	0.156	0.660	0.392
UNDEXE10FU	1.204	1.158	0.240	0.410	0.066
<b>Mean</b>	1.303	-0.007	0.167	0.566	0.331
<b>Standard deviation</b>	0.378	0.988	0.023	0.175	0.066

Table 4: Exemplifying skill - all items calibrated.

## 5 Classifying

In Table 5, we present the items that make up the classifying skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDCLA01ES	1.476	0.205	0.169	0.520	0.393
UNDCLA02ES	0.829	-0.001	0.150	0.560	0.303
UNDCLA03ES	1.304	-0.196	0.204	0.630	0.307
UNDCLA04ES	1.646	0.272	0.105	0.450	0.477
UNDCLA05ES	0.936	0.444	0.156	0.490	0.329
UNDCLA06ES	1.100	1.551	0.182	0.340	0.223
UNDCLA07ES	1.399	0.813	0.153	0.390	0.394
UNDCLA08ES	1.474	-0.810	0.211	0.760	0.353
UNDCLA09ES	1.842	-1.890	0.156	0.920	0.321
UNDCLA10ES	1.313	-1.101	0.188	0.790	0.343
UNDCLA01CS	1.826	-1.119	0.150	0.810	0.392
UNDCLA02CS	2.037	-1.721	0.153	0.910	0.348
UNDCLA03CS	1.015	-0.834	0.164	0.710	0.342
UNDCLA04CS	1.621	1.056	0.181	0.360	0.346
UNDCLA05CS	1.370	0.657	0.176	0.440	0.331
UNDCLA06CS	0.915	0.149	0.158	0.540	0.340
UNDCLA07CS	1.146	0.242	0.164	0.520	0.361
UNDCLA08CS	1.047	0.472	0.159	0.480	0.348
UNDCLA09CS	1.419	1.144	0.190	0.370	0.308
UNDCLA10CS	1.210	-0.303	0.157	0.620	0.339
UNDCLA01CI	1.525	-0.198	0.170	0.610	0.407
UNDCLA02CI	1.185	0.545	0.137	0.440	0.413
UNDCLA03CI	1.190	-2.155	0.161	0.900	0.275
UNDCLA04CI	1.118	0.681	0.195	0.470	0.302
UNDCLA05CI	1.313	-1.101	0.188	0.790	0.343
UNDCLA06CI	1.521	0.262	0.187	0.520	0.372
UNDCLA07CI	1.328	-0.639	0.154	0.690	0.378
UNDCLA08CI	1.734	0.227	0.223	0.530	0.281
UNDCLA09CI	1.040	0.193	0.161	0.530	0.327
UNDCLA10CI	2.470	1.259	0.184	0.300	0.316
UNDCLA01FU	1.464	-0.466	0.156	0.660	0.391
UNDCLA02FU	1.869	0.121	0.168	0.530	0.415
UNDCLA03FU	0.720	0.530	0.163	0.500	0.289
UNDCLA04FU	1.003	-1.652	0.178	0.830	0.249
UNDCLA05FU	1.131	-0.050	0.142	0.560	0.367
UNDCLA06FU	2.545	1.788	0.330	0.410	0.075
UNDCLA07FU	1.931	1.502	0.147	0.250	0.294
UNDCLA08FU	1.320	1.084	0.152	0.350	0.318
UNDCLA09FU	1.425	-1.041	0.148	0.770	0.398
UNDCLA10FU	1.163	0.498	0.149	0.460	0.377
<b>Mean</b>	1.398	0.010	0.170	0.568	0.337
<b>Standard deviation</b>	0.403	0.974	0.034	0.177	0.065

Table 5: Classifying skill - all items calibrated.

## 6 Summarizing

In Table 6, we present the items that make up the summarizing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDSUM01ES	1.854	-1.895	0.156	0.920	0.322
UNDSUM02ES	1.334	-1.086	0.191	0.790	0.344
UNDSUM03ES	1.281	-0.189	0.205	0.630	0.306
UNDSUM04ES	1.490	0.211	0.171	0.520	0.392
UNDSUM05ES	0.938	0.448	0.157	0.490	0.327
UNDSUM06ES	0.994	-1.044	0.183	0.750	0.312
UNDSUM07ES	0.832	0.001	0.150	0.560	0.305
UNDSUM08ES	1.164	1.505	0.184	0.340	0.225
UNDSUM09ES	1.618	0.281	0.106	0.450	0.477
UNDSUM10ES	1.470	0.793	0.154	0.390	0.395
UNDSUM01CS	1.820	-1.126	0.150	0.810	0.394
UNDSUM02CS	1.362	0.653	0.174	0.440	0.329
UNDSUM03CS	0.936	0.149	0.159	0.540	0.339
UNDSUM04CS	1.185	-0.309	0.155	0.620	0.339
UNDSUM05CS	1.042	0.475	0.159	0.480	0.347
UNDSUM06CS	1.198	0.252	0.169	0.520	0.362
UNDSUM07CS	1.015	-0.832	0.165	0.710	0.342
UNDSUM08CS	1.516	1.149	0.198	0.370	0.308
UNDSUM09CS	2.047	-1.728	0.152	0.910	0.350
UNDSUM10CS	1.822	1.040	0.189	0.360	0.346
UNDSUM01CI	1.334	-1.086	0.191	0.790	0.344
UNDSUM02CI	1.211	-2.137	0.161	0.900	0.276
UNDSUM03CI	1.086	0.328	0.168	0.510	0.329
UNDSUM04CI	1.017	0.202	0.161	0.530	0.326
UNDSUM05CI	1.148	0.708	0.203	0.470	0.300
UNDSUM06CI	1.765	0.225	0.225	0.530	0.279
UNDSUM07CI	1.200	0.543	0.137	0.440	0.411
UNDSUM08CI	2.490	1.264	0.185	0.300	0.312
UNDSUM09CI	1.454	-0.201	0.169	0.610	0.407
UNDSUM10CI	1.328	-0.638	0.154	0.690	0.379
UNDSUM01FU	0.965	-1.703	0.177	0.830	0.250
UNDSUM02FU	1.173	0.502	0.151	0.460	0.376
UNDSUM03FU	1.099	-0.044	0.143	0.560	0.367
UNDSUM04FU	1.924	1.493	0.146	0.250	0.289
UNDSUM05FU	1.440	-0.469	0.155	0.660	0.391
UNDSUM06FU	1.824	0.120	0.165	0.530	0.414
UNDSUM07FU	0.745	0.517	0.163	0.500	0.290
UNDSUM08FU	2.365	1.787	0.329	0.410	0.071
UNDSUM09FU	1.306	1.096	0.152	0.350	0.315
UNDSUM10FU	0.984	-1.232	0.149	0.760	0.336
<b>Mean</b>	1.369	0.001	0.170	0.567	0.333
<b>Standard deviation</b>	0.406	0.986	0.033	0.176	0.064

Table 6: Summarizing skill - all items calibrated.

## 7 Inferring

In Table 7, we present the items that make up the inferring skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDINF01ES	1.857	-1.915	0.157	0.920	0.316
UNDINF02ES	1.236	-1.187	0.184	0.790	0.345
UNDINF03ES	0.992	0.379	0.157	0.490	0.328
UNDINF04ES	1.585	0.703	0.155	0.390	0.399
UNDINF05ES	1.614	0.246	0.109	0.450	0.479
UNDINF06ES	1.592	0.141	0.167	0.520	0.390
UNDINF07ES	0.871	-0.052	0.149	0.560	0.302
UNDINF08ES	1.197	-0.248	0.201	0.630	0.303
UNDINF09ES	1.003	-1.076	0.183	0.750	0.313
UNDINF10ES	1.153	1.453	0.184	0.340	0.223
UNDINF01CS	1.101	-0.831	0.164	0.710	0.343
UNDINF02CS	1.323	1.055	0.177	0.370	0.301
UNDINF03CS	2.045	-1.749	0.154	0.910	0.344
UNDINF04CS	1.013	0.078	0.156	0.540	0.337
UNDINF05CS	1.862	0.947	0.186	0.360	0.343
UNDINF06CS	1.072	0.409	0.157	0.480	0.345
UNDINF07CS	1.180	-0.350	0.155	0.620	0.336
UNDINF08CS	1.190	0.204	0.168	0.520	0.360
UNDINF09CS	1.454	0.596	0.179	0.440	0.335
UNDINF10CS	0.977	-1.486	0.151	0.790	0.276
UNDINF01CI	1.099	0.281	0.168	0.510	0.327
UNDINF02CI	1.248	-2.126	0.161	0.900	0.270
UNDINF03CI	1.135	0.621	0.194	0.470	0.297
UNDINF04CI	1.572	-0.227	0.173	0.610	0.406
UNDINF05CI	1.358	-0.662	0.156	0.690	0.377
UNDINF06CI	1.231	0.467	0.133	0.440	0.407
UNDINF07CI	1.021	0.150	0.160	0.530	0.323
UNDINF08CI	0.986	-1.328	0.174	0.780	0.297
UNDINF09CI	1.702	0.000	0.200	0.550	0.191
UNDINF10CI	2.630	1.163	0.185	0.300	0.314
UNDINF01FU	1.397	-0.513	0.155	0.660	0.390
UNDINF02FU	1.026	-1.237	0.148	0.760	0.334
UNDINF03FU	1.864	1.443	0.146	0.250	0.280
UNDINF04FU	0.960	-1.744	0.176	0.830	0.252
UNDINF05FU	1.241	0.417	0.147	0.460	0.377
UNDINF06FU	1.229	-0.083	0.146	0.560	0.377
UNDINF07FU	1.259	1.061	0.151	0.350	0.308
UNDINF08FU	1.111	1.155	0.229	0.410	0.070
UNDINF09FU	0.794	0.434	0.162	0.500	0.293
UNDINF10FU	1.968	0.104	0.178	0.530	0.413
<b>Mean</b>	1.329	-0.083	0.166	0.567	0.326
<b>Standard deviation</b>	0.381	0.962	0.021	0.175	0.068

Table 7: Inferring skill - all items calibrated.

## 8 Comparing

In Table 8, we present the items that make up the comparing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDCOM01ES	1.282	-0.268	0.200	0.630	0.306
UNDCOM02ES	0.840	-0.061	0.149	0.560	0.301
UNDCOM03ES	1.573	-0.827	0.215	0.760	0.353
UNDCOM04ES	1.264	-1.189	0.185	0.790	0.343
UNDCOM05ES	1.453	0.141	0.167	0.520	0.396
UNDCOM06ES	0.970	0.372	0.156	0.490	0.330
UNDCOM07ES	1.110	1.469	0.180	0.340	0.226
UNDCOM08ES	1.356	0.762	0.152	0.390	0.394
UNDCOM09ES	1.594	0.221	0.106	0.450	0.478
UNDCOM10ES	1.927	-1.895	0.156	0.920	0.319
UNDCOM01CS	2.132	-1.733	0.153	0.910	0.346
UNDCOM02CS	1.887	-1.155	0.150	0.810	0.391
UNDCOM03CS	1.022	-0.885	0.164	0.710	0.343
UNDCOM04CS	1.195	-0.363	0.157	0.620	0.339
UNDCOM05CS	1.720	0.978	0.184	0.360	0.353
UNDCOM06CS	1.310	0.607	0.174	0.440	0.328
UNDCOM07CS	3.147	0.420	0.196	0.410	0.412
UNDCOM08CS	1.067	0.407	0.159	0.480	0.350
UNDCOM09CS	0.975	0.081	0.158	0.540	0.344
UNDCOM10CS	1.101	0.183	0.162	0.520	0.358
UNDCOM01CI	1.532	-0.260	0.169	0.610	0.409
UNDCOM02CI	1.393	-0.680	0.154	0.690	0.379
UNDCOM03CI	1.264	-1.189	0.185	0.790	0.343
UNDCOM04CI	1.018	0.139	0.160	0.530	0.325
UNDCOM05CI	1.201	0.606	0.200	0.470	0.304
UNDCOM06CI	1.818	0.248	0.231	0.530	0.285
UNDCOM07CI	1.177	0.484	0.136	0.440	0.409
UNDCOM08CI	1.503	0.197	0.185	0.520	0.376
UNDCOM09CI	1.228	-2.161	0.161	0.900	0.274
UNDCOM10CI	2.719	1.201	0.189	0.300	0.325
UNDCOM01FU	1.881	0.058	0.169	0.530	0.417
UNDCOM02FU	1.122	-0.110	0.142	0.560	0.367
UNDCOM03FU	1.930	1.441	0.146	0.250	0.298
UNDCOM04FU	1.175	0.437	0.150	0.460	0.382
UNDCOM05FU	2.372	1.780	0.330	0.410	0.073
UNDCOM06FU	1.020	0.236	0.169	0.520	0.364
UNDCOM07FU	1.426	-0.532	0.154	0.660	0.392
UNDCOM08FU	1.228	1.070	0.150	0.350	0.316
UNDCOM09FU	1.003	-1.706	0.177	0.830	0.250
UNDCOM10FU	1.424	-1.095	0.147	0.770	0.395
<b>Mean</b>	1.459	-0.064	0.171	0.569	0.342
<b>Standard deviation</b>	0.493	0.952	0.034	0.175	0.066

Table 8: Comparing skill - all items calibrated.

## 9 Explaining

In Table 9, we present the items that make up the explaining skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
UNDEXP01ES	1.600	0.177	0.170	0.520	0.386
UNDEXP02ES	1.057	-0.061	0.144	0.560	0.352
UNDEXP03ES	0.956	-1.099	0.180	0.750	0.306
UNDEXP04ES	1.280	-1.140	0.186	0.790	0.350
UNDEXP05ES	0.996	0.398	0.156	0.490	0.329
UNDEXP06ES	1.648	0.262	0.107	0.450	0.482
UNDEXP07ES	1.561	0.718	0.151	0.390	0.399
UNDEXP08ES	1.170	-0.227	0.201	0.630	0.304
UNDEXP09ES	1.774	-1.949	0.157	0.920	0.323
UNDEXP10ES	1.202	1.448	0.186	0.340	0.226
UNDEXP01CS	0.976	-1.470	0.150	0.790	0.285
UNDEXP02CS	1.049	0.433	0.155	0.480	0.343
UNDEXP03CS	1.079	-0.818	0.165	0.710	0.339
UNDEXP04CS	1.165	0.058	0.149	0.540	0.375
UNDEXP05CS	1.407	0.629	0.178	0.440	0.326
UNDEXP06CS	0.950	1.793	0.182	0.330	0.184
UNDEXP07CS	1.953	-1.776	0.154	0.910	0.350
UNDEXP08CS	1.162	0.234	0.168	0.520	0.359
UNDEXP09CS	1.130	-0.335	0.155	0.620	0.331
UNDEXP10CS	2.015	0.965	0.192	0.360	0.345
UNDEXP01CI	1.117	0.651	0.194	0.470	0.292
UNDEXP02CI	1.098	0.312	0.170	0.510	0.328
UNDEXP03CI	1.751	0.216	0.223	0.530	0.275
UNDEXP04CI	1.189	-2.179	0.161	0.900	0.275
UNDEXP05CI	1.347	-0.642	0.157	0.690	0.382
UNDEXP06CI	1.191	0.495	0.131	0.440	0.404
UNDEXP07CI	3.187	1.138	0.189	0.300	0.322
UNDEXP08CI	1.038	-1.261	0.175	0.780	0.305
UNDEXP09CI	1.506	-0.216	0.169	0.610	0.407
UNDEXP10CI	1.001	0.180	0.160	0.530	0.322
UNDEXP01FU	1.201	-0.060	0.145	0.560	0.373
UNDEXP02FU	0.924	-1.775	0.176	0.830	0.248
UNDEXP03FU	1.002	-1.238	0.148	0.760	0.331
UNDEXP04FU	1.311	0.428	0.148	0.460	0.380
UNDEXP05FU	2.001	0.125	0.177	0.530	0.418
UNDEXP06FU	1.295	1.138	0.248	0.410	0.063
UNDEXP07FU	1.422	-0.483	0.157	0.660	0.398
UNDEXP08FU	0.799	0.452	0.162	0.500	0.295
UNDEXP09FU	1.854	1.476	0.147	0.250	0.278
UNDEXP10FU	1.290	1.081	0.153	0.350	0.313
<b>Mean</b>	1.341	-0.048	0.167	0.565	0.328
<b>Standard deviation</b>	0.437	0.999	0.024	0.176	0.070

Table 9: Explaining skill - all items calibrated.



## 10 Executing

In Table 10, we present the items that make up the executing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
APPEXE01ES	1.217	-1.239	0.182	0.790	0.350
APPEXE02ES	1.193	-0.287	0.200	0.630	0.303
APPEXE03ES	0.991	-1.124	0.181	0.750	0.302
APPEXE04ES	1.476	0.244	0.112	0.450	0.483
APPEXE05ES	1.683	-2.049	0.158	0.920	0.317
APPEXE06ES	1.089	-0.119	0.143	0.560	0.357
APPEXE07ES	1.531	0.129	0.171	0.520	0.391
APPEXE08ES	1.056	0.321	0.156	0.490	0.333
APPEXE09ES	1.206	1.400	0.186	0.340	0.229
APPEXE10ES	1.446	0.700	0.150	0.390	0.395
APPEXE01CS	1.051	0.376	0.155	0.480	0.350
APPEXE02CS	1.876	-1.858	0.155	0.910	0.349
APPEXE03CS	1.326	0.609	0.179	0.440	0.330
APPEXE04CS	0.910	-1.596	0.151	0.790	0.277
APPEXE05CS	2.022	0.928	0.195	0.360	0.356
APPEXE06CS	3.397	0.421	0.203	0.410	0.410
APPEXE07CS	1.161	0.175	0.167	0.520	0.356
APPEXE08CS	1.169	-0.383	0.156	0.620	0.335
APPEXE09CS	1.185	-0.006	0.148	0.540	0.382
APPEXE10CS	1.082	-0.871	0.165	0.710	0.344
APPEXE01CI	1.143	-2.287	0.162	0.900	0.272
APPEXE02CI	2.007	-0.239	0.177	0.610	0.411
APPEXE03CI	1.394	-0.685	0.157	0.690	0.380
APPEXE04CI	1.113	0.236	0.166	0.510	0.335
APPEXE05CI	0.976	-1.376	0.173	0.780	0.303
APPEXE06CI	1.244	0.419	0.131	0.440	0.405
APPEXE07CI	1.189	0.604	0.202	0.470	0.301
APPEXE08CI	0.999	0.125	0.160	0.530	0.326
APPEXE09CI	2.172	0.291	0.260	0.530	0.283
APPEXE10CI	2.936	1.128	0.189	0.300	0.324
APPEXE01FU	2.488	-0.165	0.162	0.580	0.461
APPEXE02FU	1.086	-1.236	0.146	0.760	0.335
APPEXE03FU	0.919	-1.839	0.176	0.830	0.250
APPEXE04FU	1.192	1.089	0.153	0.350	0.308
APPEXE05FU	1.746	1.806	0.322	0.410	0.063
APPEXE06FU	2.141	1.249	0.146	0.250	0.284
APPEXE07FU	1.224	0.392	0.147	0.460	0.384
APPEXE08FU	1.343	-0.554	0.156	0.660	0.392
APPEXE09FU	1.267	0.170	0.145	0.500	0.429
APPEXE10FU	1.990	0.060	0.175	0.530	0.421
<b>Mean</b>	1.466	-0.126	0.170	0.568	0.340
<b>Standard deviation</b>	0.558	0.995	0.035	0.174	0.071

Table 10: Executing skill - all items calibrated.

## 11 Implementing

In Table 11, we present the items that make up the implementing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
APPIMP01ES	1.293	0.429	0.146	0.460	0.383
APPIMP02ES	1.217	1.445	0.185	0.340	0.227
APPIMP03ES	1.638	-2.039	0.158	0.920	0.319
APPIMP04ES	1.295	0.222	0.144	0.500	0.426
APPIMP05ES	2.398	1.747	0.330	0.410	0.065
APPIMP06ES	1.168	-0.229	0.201	0.630	0.303
APPIMP07ES	1.308	-1.129	0.187	0.790	0.350
APPIMP08ES	1.932	0.253	0.241	0.530	0.279
APPIMP09ES	0.950	-1.105	0.181	0.750	0.301
APPIMP10ES	1.062	-0.064	0.142	0.560	0.357
APPIMP01CS	1.560	1.110	0.199	0.370	0.303
APPIMP02CS	1.156	0.052	0.147	0.540	0.379
APPIMP03CS	0.940	-1.512	0.151	0.790	0.283
APPIMP04CS	1.031	0.392	0.157	0.490	0.335
APPIMP05CS	1.532	0.190	0.171	0.520	0.390
APPIMP06CS	1.078	-0.824	0.164	0.710	0.343
APPIMP07CS	1.542	0.738	0.152	0.390	0.396
APPIMP08CS	1.584	0.277	0.108	0.450	0.483
APPIMP09CS	1.826	-1.844	0.155	0.910	0.351
APPIMP10CS	1.101	-0.341	0.155	0.620	0.336
APPIMP01CI	1.125	-2.267	0.162	0.900	0.272
APPIMP02CI	1.309	-0.655	0.157	0.690	0.379
APPIMP03CI	1.369	0.648	0.178	0.440	0.327
APPIMP04CI	0.976	0.185	0.160	0.530	0.322
APPIMP05CI	1.149	0.249	0.171	0.520	0.353
APPIMP06CI	2.134	0.972	0.196	0.360	0.353
APPIMP07CI	1.524	-0.221	0.166	0.610	0.413
APPIMP08CI	1.062	-1.245	0.176	0.780	0.306
APPIMP09CI	1.026	0.446	0.156	0.480	0.348
APPIMP10CI	2.973	1.109	0.178	0.300	0.322
APPIMP01FU	0.996	-1.249	0.147	0.760	0.335
APPIMP02FU	1.371	-0.494	0.157	0.660	0.391
APPIMP03FU	1.834	1.491	0.145	0.250	0.279
APPIMP04FU	0.894	-1.826	0.176	0.830	0.250
APPIMP05FU	1.201	-0.065	0.143	0.560	0.377
APPIMP06FU	1.257	1.121	0.155	0.350	0.309
APPIMP07FU	1.901	0.121	0.172	0.530	0.418
APPIMP08FU	1.158	0.682	0.203	0.470	0.296
APPIMP09FU	1.082	0.312	0.168	0.510	0.334
APPIMP10FU	1.179	0.501	0.131	0.440	0.403
<b>Mean</b>	1.378	-0.060	0.169	0.566	0.335
<b>Standard deviation</b>	0.435	1.013	0.035	0.175	0.068

Table 11: Implementing skill - all items calibrated.

## 12 Differentiating

In Table 12, we present the items that make up the differentiating skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
ANADIF01CS	1.576	0.045	0.146	0.560	0.428
ANADIF02CS	1.215	0.362	0.175	0.520	0.326
ANADIF03CS	1.079	0.282	0.162	0.530	0.330
ANADIF04CS	0.979	0.500	0.153	0.490	0.328
ANADIF05CS	1.549	-2.017	0.159	0.920	0.325
ANADIF06CS	1.071	0.611	0.147	0.460	0.358
ANADIF07CS	0.979	-0.207	0.184	0.630	0.294
ANADIF08CS	0.885	-0.843	0.162	0.710	0.314
ANADIF09CS	1.092	-1.115	0.180	0.780	0.327
ANADIF10CS	0.953	-1.644	0.178	0.830	0.256
ANADIF01CI	1.297	-0.557	0.156	0.690	0.396
ANADIF02CI	1.803	0.223	0.166	0.530	0.424
ANADIF03CI	0.990	0.052	0.144	0.560	0.364
ANADIF04CI	1.104	-0.236	0.155	0.620	0.352
ANADIF05CI	0.907	0.413	0.156	0.510	0.303
ANADIF06CI	0.994	0.590	0.161	0.480	0.317
ANADIF07CI	1.324	-1.013	0.192	0.790	0.366
ANADIF08CI	1.203	-1.016	0.146	0.760	0.402
ANADIF09CI	1.732	-1.811	0.155	0.910	0.378
ANADIF10CI	1.795	0.358	0.105	0.450	0.518
ANADIF01FU	0.970	-2.419	0.163	0.900	0.235
ANADIF02FU	0.970	0.190	0.150	0.540	0.357
ANADIF03FU	1.148	0.390	0.152	0.500	0.380
ANADIF04FU	1.048	-1.313	0.150	0.790	0.331
ANADIF05FU	0.917	0.795	0.180	0.470	0.258
ANADIF06FU	1.536	0.786	0.194	0.440	0.308
ANADIF07FU	1.055	-0.910	0.188	0.750	0.317
ANADIF08FU	1.344	-0.395	0.156	0.660	0.401
ANADIF09FU	1.824	-0.077	0.175	0.610	0.440
ANADIF10FU	1.462	0.318	0.173	0.520	0.370
<b>Mean</b>	1.227	-0.322	0.162	0.630	0.350
<b>Standard deviation</b>	0.298	0.891	0.018	0.150	0.059

Table 12: Differentiating skill - all items calibrated.

## 13 Organizing

In Table 13, we present the items that make up the organizing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
ANAORG01ES	1.004	-1.018	0.186	0.750	0.307
ANAORG02ES	1.482	0.221	0.170	0.520	0.391
ANAORG03ES	1.994	1.854	0.326	0.410	0.061
ANAORG04ES	1.156	1.483	0.180	0.340	0.228
ANAORG05ES	1.377	-1.050	0.192	0.790	0.354
ANAORG06ES	1.619	-2.019	0.157	0.920	0.318
ANAORG07ES	1.018	-0.021	0.146	0.560	0.365
ANAORG08ES	1.323	0.551	0.201	0.460	0.206
ANAORG09ES	1.006	0.411	0.153	0.490	0.340
ANAORG10ES	1.249	-0.198	0.200	0.630	0.302
ANAORG01CS	2.151	-0.958	0.139	0.790	0.471
ANAORG02CS	1.006	-0.827	0.164	0.710	0.338
ANAORG03CS	1.101	-0.304	0.157	0.620	0.331
ANAORG04CS	1.271	0.262	0.172	0.520	0.362
ANAORG05CS	1.816	-1.817	0.153	0.910	0.354
ANAORG06CS	1.068	0.475	0.159	0.480	0.354
ANAORG07CS	1.475	0.782	0.151	0.390	0.386
ANAORG08CS	1.785	0.266	0.104	0.450	0.484
ANAORG09CS	1.091	0.103	0.151	0.540	0.383
ANAORG10CS	1.058	1.915	0.204	0.330	0.184
ANAORG01CI	1.093	-1.184	0.178	0.780	0.310
ANAORG02CI	1.057	0.322	0.163	0.510	0.328
ANAORG03CI	1.049	0.190	0.157	0.530	0.329
ANAORG04CI	1.349	-0.615	0.156	0.690	0.383
ANAORG05CI	1.409	0.659	0.177	0.440	0.322
ANAORG06CI	1.883	1.033	0.191	0.360	0.348
ANAORG07CI	3.034	1.189	0.188	0.300	0.315
ANAORG08CI	1.083	-2.295	0.162	0.900	0.271
ANAORG09CI	1.492	-0.198	0.165	0.610	0.413
ANAORG10CI	1.611	0.795	0.252	0.470	0.301
ANAORG01FU	1.822	1.511	0.144	0.250	0.271
ANAORG02FU	1.285	1.134	0.156	0.350	0.302
ANAORG03FU	1.255	0.491	0.152	0.460	0.387
ANAORG04FU	1.313	0.256	0.146	0.500	0.434
ANAORG05FU	1.213	0.521	0.131	0.440	0.413
ANAORG06FU	1.673	-1.240	0.166	0.830	0.382
ANAORG07FU	1.145	-0.032	0.144	0.560	0.368
ANAORG08FU	1.464	-0.452	0.155	0.660	0.391
ANAORG09FU	1.716	0.120	0.159	0.530	0.410
ANAORG10FU	1.018	-1.203	0.146	0.760	0.345
<b>Mean</b>	1.400	0.028	0.169	0.564	0.339
<b>Standard deviation</b>	0.407	1.015	0.035	0.177	0.077

Table 13: Organizing skill - all items calibrated.

## 14 Attributing

In Table 14, we present the items that make up the attributing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
ANAATT01ES	1.001	0.423	0.154	0.490	0.339
ANAATT02ES	1.455	0.227	0.168	0.520	0.390
ANAATT03ES	1.797	0.268	0.102	0.450	0.483
ANAATT04ES	1.211	1.443	0.181	0.340	0.230
ANAATT05ES	1.277	-0.181	0.202	0.630	0.299
ANAATT06ES	1.365	-1.049	0.193	0.790	0.353
ANAATT07ES	1.052	-0.971	0.189	0.750	0.305
ANAATT08ES	0.986	-0.008	0.147	0.560	0.359
ANAATT09ES	2.067	1.879	0.329	0.410	0.057
ANAATT10ES	1.638	-2.006	0.157	0.920	0.323
ANAATT01CS	2.150	-0.956	0.139	0.790	0.474
ANAATT02CS	1.091	-0.297	0.157	0.620	0.331
ANAATT03CS	0.991	-0.830	0.165	0.710	0.340
ANAATT04CS	1.034	0.494	0.159	0.480	0.352
ANAATT05CS	1.431	0.279	0.182	0.520	0.364
ANAATT06CS	1.042	0.121	0.152	0.540	0.380
ANAATT07CS	1.408	0.659	0.176	0.440	0.323
ANAATT08CS	2.308	1.821	0.258	0.330	0.186
ANAATT09CS	1.524	0.787	0.154	0.390	0.384
ANAATT10CS	1.836	-1.808	0.153	0.910	0.359
ANAATT01CI	1.088	-1.182	0.179	0.780	0.309
ANAATT02CI	1.473	-0.190	0.165	0.610	0.411
ANAATT03CI	1.321	-0.620	0.155	0.690	0.381
ANAATT04CI	3.026	1.172	0.187	0.300	0.314
ANAATT05CI	1.040	0.334	0.163	0.510	0.322
ANAATT06CI	1.216	0.531	0.132	0.440	0.412
ANAATT07CI	1.774	1.057	0.188	0.360	0.342
ANAATT08CI	1.054	0.199	0.158	0.530	0.331
ANAATT09CI	1.071	-2.309	0.162	0.900	0.275
ANAATT10CI	1.201	0.743	0.211	0.470	0.296
ANAATT01FU	1.837	1.485	0.142	0.250	0.276
ANAATT02FU	1.393	0.267	0.151	0.500	0.440
ANAATT03FU	1.259	0.504	0.154	0.460	0.386
ANAATT04FU	1.677	0.122	0.156	0.530	0.410
ANAATT05FU	1.528	-0.431	0.157	0.660	0.395
ANAATT06FU	1.131	-0.026	0.144	0.560	0.364
ANAATT07FU	1.002	-1.211	0.146	0.760	0.343
ANAATT08FU	1.312	1.118	0.155	0.350	0.304
ANAATT09FU	1.688	-1.232	0.166	0.830	0.385
ANAATT10FU	1.383	0.920	0.235	0.420	0.129
<b>Mean</b>	1.428	0.039	0.171	0.563	0.336
<b>Standard deviation</b>	0.430	1.013	0.038	0.178	0.081

Table 14: Attributing skill - all items calibrated.

## 15 Checking

In Table 15, we present the items that make up the checking skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
EVACHE01ES	1.311	-0.133	0.207	0.630	0.300
EVACHE02ES	1.773	0.311	0.103	0.450	0.488
EVACHE03ES	1.036	-0.958	0.188	0.750	0.310
EVACHE04ES	1.564	-2.041	0.157	0.920	0.326
EVACHE05ES	1.419	0.263	0.167	0.520	0.394
EVACHE06ES	0.968	0.026	0.147	0.560	0.362
EVACHE07ES	1.310	-1.048	0.193	0.790	0.351
EVACHE08ES	1.245	1.468	0.182	0.340	0.232
EVACHE09ES	0.963	0.468	0.153	0.490	0.340
EVACHE10ES	2.233	1.950	0.334	0.410	0.054
EVACHE01CS	0.987	0.540	0.157	0.480	0.350
EVACHE02CS	0.994	0.152	0.150	0.540	0.377
EVACHE03CS	1.537	0.833	0.156	0.390	0.384
EVACHE04CS	1.443	-0.683	0.146	0.710	0.452
EVACHE05CS	2.069	-0.946	0.140	0.790	0.475
EVACHE06CS	1.355	0.712	0.175	0.440	0.321
EVACHE07CS	1.091	-0.267	0.157	0.620	0.335
EVACHE08CS	1.373	0.316	0.180	0.520	0.365
EVACHE09CS	2.121	1.837	0.254	0.330	0.188
EVACHE10CS	1.753	-1.833	0.153	0.910	0.362
EVACHE01CI	1.026	0.231	0.157	0.530	0.329
EVACHE02CI	1.047	0.366	0.162	0.510	0.322
EVACHE03CI	1.778	1.120	0.192	0.360	0.340
EVACHE04CI	1.037	-2.343	0.162	0.900	0.278
EVACHE05CI	1.058	-1.173	0.180	0.780	0.310
EVACHE06CI	1.480	-0.163	0.164	0.610	0.414
EVACHE07CI	1.060	0.774	0.197	0.470	0.294
EVACHE08CI	1.196	0.571	0.132	0.440	0.409
EVACHE09CI	1.314	-0.595	0.154	0.690	0.384
EVACHE10CI	3.117	1.208	0.185	0.300	0.315
EVACHE01FU	1.859	1.505	0.140	0.250	0.274
EVACHE02FU	1.327	0.299	0.147	0.500	0.440
EVACHE03FU	1.664	-1.218	0.166	0.830	0.386
EVACHE04FU	1.223	0.546	0.152	0.460	0.389
EVACHE05FU	1.553	-0.398	0.157	0.660	0.398
EVACHE06FU	1.148	0.010	0.144	0.560	0.365
EVACHE07FU	1.324	1.173	0.158	0.350	0.302
EVACHE08FU	0.989	-1.196	0.146	0.760	0.344
EVACHE09FU	0.797	1.508	0.226	0.420	0.120
EVACHE10FU	1.684	0.173	0.161	0.530	0.407
<b>Mean</b>	1.406	0.084	0.169	0.563	0.340
<b>Standard deviation</b>	0.446	1.044	0.037	0.178	0.084

Table 15: Checking skill - all items calibrated.

## 16 Critiquing

In Table 16, we present the items that make up the critiquing skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
EVACRI01ES	1.313	-1.088	0.192	0.790	0.350
EVACRI02ES	1.065	-0.982	0.188	0.750	0.312
EVACRI03ES	1.287	-0.192	0.204	0.630	0.302
EVACRI04ES	1.670	-1.993	0.157	0.920	0.325
EVACRI05ES	1.018	-0.033	0.146	0.560	0.363
EVACRI06ES	1.528	0.189	0.166	0.520	0.395
EVACRI07ES	1.804	0.249	0.103	0.450	0.486
EVACRI08ES	1.010	0.384	0.151	0.490	0.338
EVACRI09ES	1.057	1.232	0.233	0.410	0.054
EVACRI10ES	1.246	1.382	0.180	0.340	0.234
EVACRI01CS	2.119	-0.969	0.139	0.790	0.473
EVACRI02CS	1.115	1.833	0.205	0.330	0.187
EVACRI03CS	1.545	0.746	0.153	0.390	0.386
EVACRI04CS	1.391	0.632	0.174	0.440	0.323
EVACRI05CS	1.130	-0.314	0.156	0.620	0.336
EVACRI06CS	1.871	-1.797	0.153	0.910	0.360
EVACRI07CS	1.088	0.082	0.149	0.540	0.379
EVACRI08CS	1.377	0.247	0.177	0.520	0.366
EVACRI09CS	1.493	-0.716	0.145	0.710	0.453
EVACRI10CS	1.060	0.449	0.156	0.480	0.351
EVACRI01CI	3.019	1.143	0.186	0.300	0.314
EVACRI02CI	1.527	-0.206	0.165	0.610	0.415
EVACRI03CI	1.047	-1.226	0.178	0.780	0.306
EVACRI04CI	1.246	0.488	0.130	0.440	0.410
EVACRI05CI	1.069	0.307	0.163	0.510	0.324
EVACRI06CI	1.146	0.677	0.199	0.470	0.296
EVACRI07CI	1.110	-2.266	0.161	0.900	0.277
EVACRI08CI	1.816	1.016	0.189	0.360	0.342
EVACRI09CI	1.379	-0.622	0.154	0.690	0.386
EVACRI10CI	1.041	0.172	0.156	0.530	0.330
EVACRI01FU	1.171	-0.042	0.145	0.560	0.366
EVACRI02FU	1.321	0.782	0.204	0.420	0.179
EVACRI03FU	1.517	-0.453	0.156	0.660	0.400
EVACRI04FU	1.310	1.098	0.156	0.350	0.301
EVACRI05FU	1.266	0.463	0.150	0.460	0.387
EVACRI06FU	1.896	1.423	0.141	0.250	0.273
EVACRI07FU	1.395	0.226	0.146	0.500	0.441
EVACRI08FU	1.668	-1.249	0.166	0.830	0.385
EVACRI09FU	1.711	0.120	0.163	0.530	0.406
EVACRI10FU	1.010	-1.219	0.146	0.760	0.343
<b>Mean</b>	1.396	-0.001	0.165	0.563	0.341
<b>Standard deviation</b>	0.391	0.971	0.024	0.178	0.081

Table 16: Critiquing skill - all items calibrated.

## 17 Generating

In Table 17, we present the items that make up the generating skill with the respective parameters.

Item	Slope	Threshold	Asymptote	Hit ratio	Point-biserial
CREGEN01ES	0.966	-0.019	0.148	0.560	0.360
CREGEN02ES	0.953	0.421	0.153	0.490	0.336
CREGEN03ES	1.281	-0.168	0.212	0.630	0.302
CREGEN04ES	1.327	-1.065	0.198	0.790	0.349
CREGEN05ES	1.742	0.250	0.101	0.450	0.485
CREGEN06ES	1.046	-1.003	0.184	0.750	0.318
CREGEN07ES	1.430	0.170	0.157	0.520	0.394
CREGEN08ES	3.309	-1.634	0.154	0.930	0.375
CREGEN09ES	0.995	1.283	0.229	0.410	0.051
CREGEN10ES	1.221	1.444	0.184	0.340	0.234
CREGEN01CS	3.770	-1.515	0.148	0.920	0.407
CREGEN02CS	0.932	1.866	0.184	0.330	0.188
CREGEN03CS	2.122	-0.975	0.138	0.790	0.470
CREGEN04CS	1.000	0.488	0.158	0.480	0.349
CREGEN05CS	1.336	0.634	0.169	0.440	0.327
CREGEN06CS	1.580	0.747	0.153	0.390	0.392
CREGEN07CS	1.353	0.254	0.177	0.520	0.371
CREGEN08CS	1.010	0.124	0.156	0.540	0.378
CREGEN09CS	1.169	-0.320	0.152	0.620	0.342
CREGEN10CS	1.403	-0.726	0.150	0.710	0.454
CREGEN01CI	1.065	0.177	0.158	0.530	0.333
CREGEN02CI	1.022	0.334	0.165	0.510	0.324
CREGEN03CI	1.214	0.509	0.131	0.440	0.412
CREGEN06CI	3.131	1.156	0.187	0.300	0.320
CREGEN05CI	1.851	1.003	0.186	0.360	0.346
CREGEN04CI	1.111	0.649	0.189	0.470	0.300
CREGEN07CI	1.245	-2.098	0.160	0.900	0.278
CREGEN08CI	1.481	-0.986	0.183	0.780	0.401
CREGEN09CI	1.343	-0.632	0.154	0.690	0.388
CREGEN10CI	1.571	-0.176	0.176	0.610	0.417
CREGEN01FU	1.828	-0.877	0.145	0.760	0.491
CREGEN02FU	1.983	1.403	0.140	0.250	0.276
CREGEN03FU	1.191	-0.026	0.149	0.560	0.369
CREGEN04FU	1.205	1.127	0.148	0.350	0.297
CREGEN05FU	1.534	-0.460	0.153	0.660	0.404
CREGEN06FU	1.715	-1.235	0.164	0.830	0.383
CREGEN07FU	1.745	0.126	0.166	0.530	0.407
CREGEN08FU	1.397	0.281	0.161	0.500	0.442
CREGEN09FU	1.283	0.487	0.156	0.460	0.390
CREGEN10FU	1.285	0.809	0.204	0.420	0.177
<b>Mean</b>	1.504	0.046	0.165	0.563	0.351
<b>Standard deviation</b>	0.627	0.921	0.024	0.179	0.085

Table 17: Generating skill - all items calibrated.



## 18 Planning

In Table 18, we present the items that make up the planning skill with the respective parameters.

Item	Slope	Threshold	Hit ratio	Point-biserial
CREPLA01ES	0.829	-0.724	0.630	0.309
CREPLA02ES	0.921	-1.371	0.750	0.328
CREPLA03ES	1.579	-2.042	0.920	0.326
CREPLA04ES	1.410	0.609	0.340	0.467
CREPLA05ES	0.913	0.052	0.490	0.337
CREPLA06ES	1.037	-0.282	0.560	0.378
CREPLA07ES	1.182	-0.093	0.520	0.409
CREPLA08ES	1.075	-1.467	0.790	0.358
CREPLA09ES	1.394	0.173	0.450	0.480
CREPLA10ES	0.964	0.446	0.410	0.361
CREPLA01CS	1.736	-1.836	0.910	0.361
CREPLA02CS	0.972	0.096	0.480	0.370
CREPLA03CS	1.059	-0.187	0.540	0.397
CREPLA04CS	1.927	-1.018	0.790	0.470
CREPLA05CS	1.027	-0.570	0.620	0.349
CREPLA06CS	0.947	-0.099	0.520	0.362
CREPLA07CS	1.002	0.531	0.390	0.389
CREPLA08CS	1.624	-0.768	0.710	0.458
CREPLA09CS	0.775	0.358	0.440	0.323
CREPLA10CS	1.014	0.731	0.350	0.374
CREPLA01CI	1.137	-2.295	0.900	0.279
CREPLA02CI	1.349	-1.200	0.780	0.415
CREPLA03CI	1.678	-0.056	0.510	0.471
CREPLA04CI	0.946	-0.149	0.530	0.337
CREPLA05CI	0.811	0.174	0.470	0.317
CREPLA06CI	1.152	0.251	0.440	0.421
CREPLA07CI	1.230	-0.817	0.690	0.387
CREPLA08CI	1.638	-0.396	0.610	0.412
CREPLA09CI	1.210	0.877	0.300	0.410
CREPLA10CI	0.868	0.770	0.360	0.344
CREPLA01FU	2.361	-0.809	0.760	0.504
CREPLA02FU	0.812	0.875	0.350	0.305
CREPLA03FU	2.641	-0.192	0.560	0.541
CREPLA04FU	0.795	1.579	0.250	0.277
CREPLA05FU	1.285	-0.662	0.660	0.399
CREPLA06FU	1.554	-1.365	0.830	0.379
CREPLA07FU	1.256	-0.132	0.530	0.403
CREPLA08FU	1.185	-0.009	0.500	0.438
CREPLA09FU	0.784	0.472	0.420	0.305
CREPLA10FU	0.986	0.191	0.460	0.387
<b>Mean</b>	1.227	-0.259	0.563	0.383
<b>Standard deviation</b>	0.420	0.856	0.177	0.062

Table 18: Planning skill - all items calibrated.

## 19 Producing

In Table 19, we present the items that make up the producing skill with the respective parameters.

<b>Item</b>	<b>Slope</b>	<b>Threshold</b>	<b>Hit ratio</b>	<b>Point-biserial</b>
CREPRO01ES	0.843	-0.715	0.630	0.313
CREPRO02ES	0.929	-1.363	0.750	0.331
CREPRO03ES	1.530	-2.089	0.920	0.318
CREPRO04ES	1.028	0.725	0.350	0.376
CREPRO05ES	0.903	0.054	0.490	0.342
CREPRO06ES	1.047	-0.279	0.560	0.379
CREPRO07ES	1.208	-0.091	0.520	0.412
CREPRO08ES	1.056	-1.488	0.790	0.356
CREPRO09ES	1.427	0.171	0.450	0.484
CREPRO10ES	0.980	0.441	0.410	0.363
CREPRO01CS	1.696	-1.869	0.910	0.356
CREPRO02CS	0.972	0.097	0.480	0.374
CREPRO03CS	1.064	-0.185	0.540	0.403
CREPRO04CS	1.946	-1.014	0.790	0.471
CREPRO05CS	1.008	-0.577	0.620	0.351
CREPRO06CS	0.921	-0.100	0.520	0.359
CREPRO07CS	1.034	0.520	0.390	0.396
CREPRO08CS	1.689	-0.751	0.710	0.463
CREPRO09CS	0.784	0.355	0.440	0.326
CREPRO10CS	1.445	0.600	0.340	0.473
CREPRO01CI	1.102	-2.353	0.900	0.272
CREPRO02CI	1.321	-1.218	0.780	0.411
CREPRO03CI	1.661	-0.054	0.510	0.473
CREPRO04CI	0.942	-0.149	0.530	0.337
CREPRO05CI	1.129	0.257	0.440	0.419
CREPRO06CI	0.828	0.171	0.470	0.320
CREPRO07CI	1.233	-0.816	0.690	0.392
CREPRO08CI	1.543	-0.407	0.610	0.410
CREPRO09CI	1.197	0.887	0.300	0.409
CREPRO10CI	0.895	0.753	0.360	0.353
CREPRO01FU	2.299	-0.817	0.760	0.506
CREPRO02FU	1.395	0.573	0.350	0.457
CREPRO03FU	2.459	-0.194	0.560	0.538
CREPRO04FU	2.034	0.774	0.260	0.542
CREPRO05FU	1.260	-0.670	0.660	0.395
CREPRO06FU	1.519	-1.387	0.830	0.377
CREPRO07FU	1.212	-0.133	0.530	0.406
CREPRO08FU	1.190	-0.008	0.500	0.440
CREPRO09FU	0.894	0.423	0.420	0.358
CREPRO10FU	0.973	0.195	0.460	0.386
<b>Mean</b>	1.265	-0.293	0.563	0.396
<b>Standard deviation</b>	0.403	0.819	0.177	0.063

Table 19: Producing skill - all items calibrated.

# **Appendix L**

## **Items Characteristic Curves**

# 1 Recognizing

In Figure 1, we present the ICC's graphic representation for the recognizing skill. The slope, threshold, and asymptote' parameters extreme values.

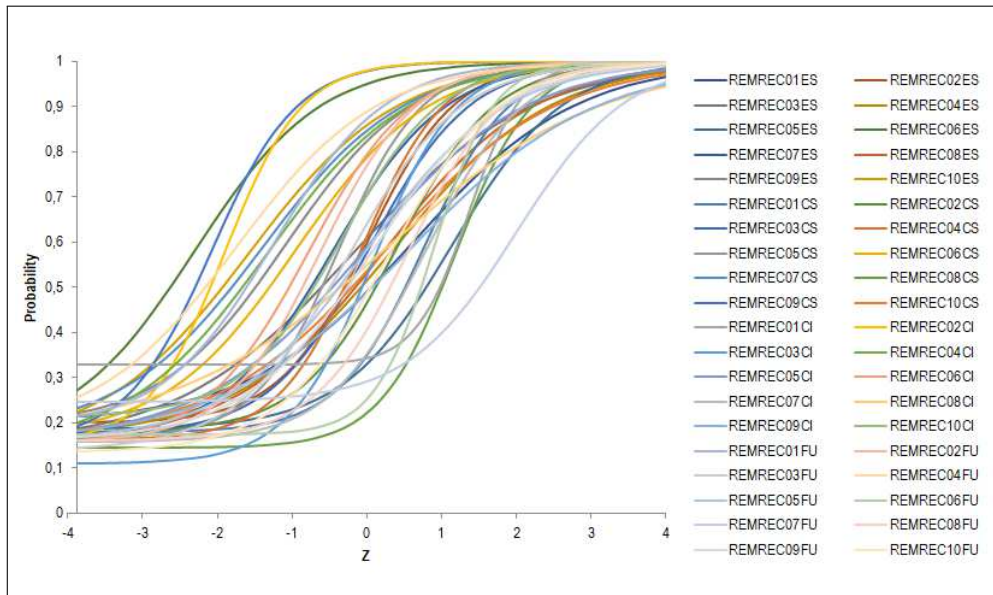


Figure 1: Recognizing skill, all ICC's.

# 2 Recalling

In Figure 2, we present the ICC's graphic representation for the recalling skill. The slope, threshold, and asymptote' parameters extreme values.

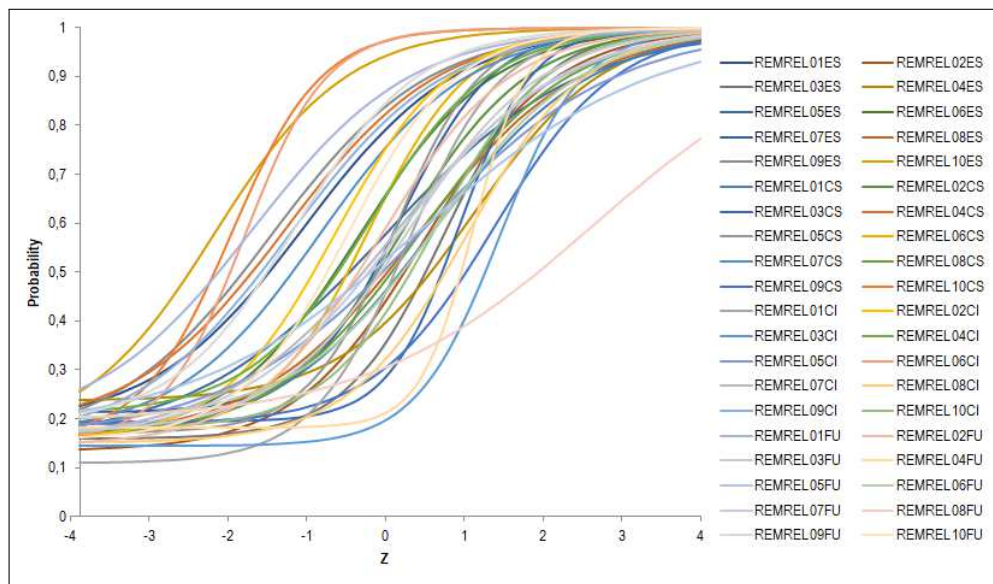


Figure 2: Recalling skill, all ICC's.

### 3 Interpreting

In Figure 3, we present the ICC's graphic representation for the interpreting skill. The slope, threshold, and asymptote' parameters extreme values.

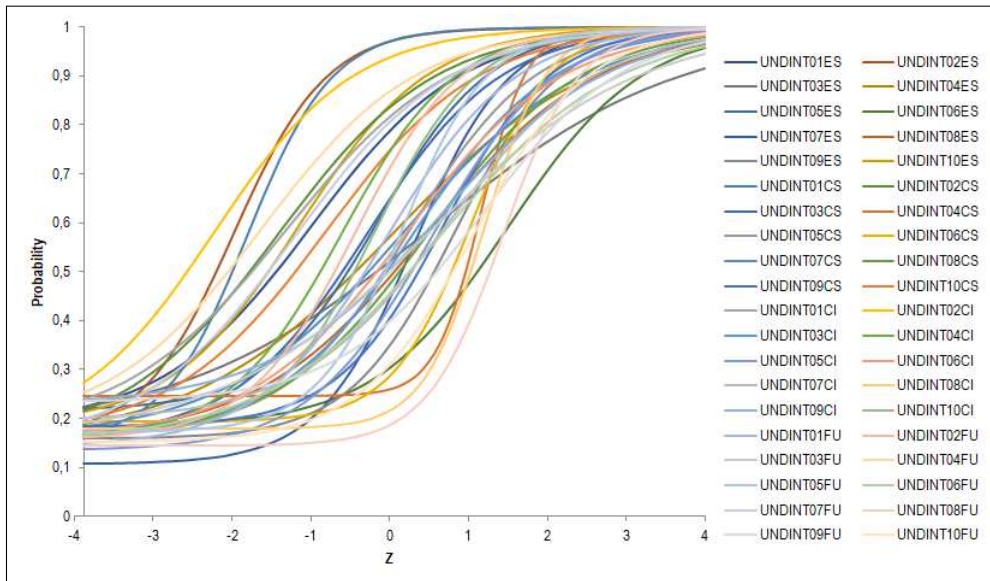


Figure 3: Interpreting skill, all ICC's.

### 4 Exemplifying

In Figure 4, we present the ICC's graphic representation for the exemplifying skill. The slope, threshold, and asymptote' parameters extreme values.

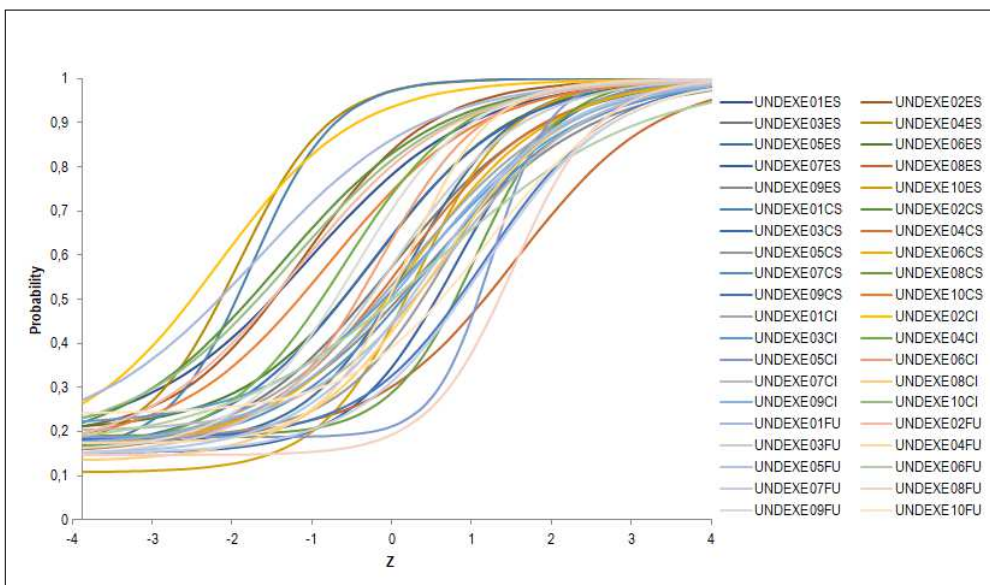


Figure 4: Exemplifying skill, all ICC's.

## 5 Classifying

In Figure 5, we present the ICC's graphic representation for the classifying skill. The slope, threshold, and asymptote' parameters extreme values.

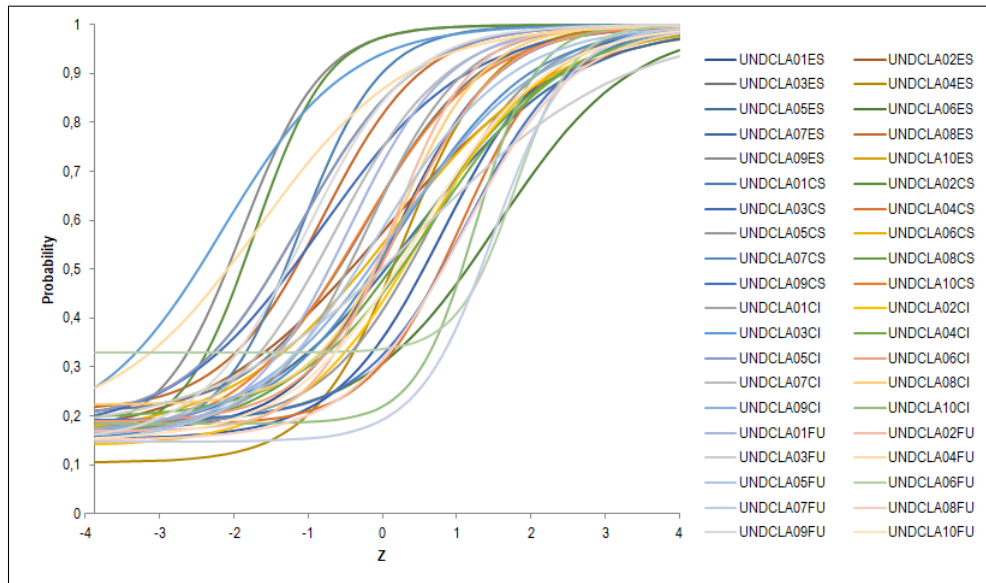


Figure 5: Classifying skill, all ICC's.

## 6 Summarizing

In Figure 6, we present the ICC's graphic representation for the summarizing skill. The slope, threshold, and asymptote' parameters extreme values.

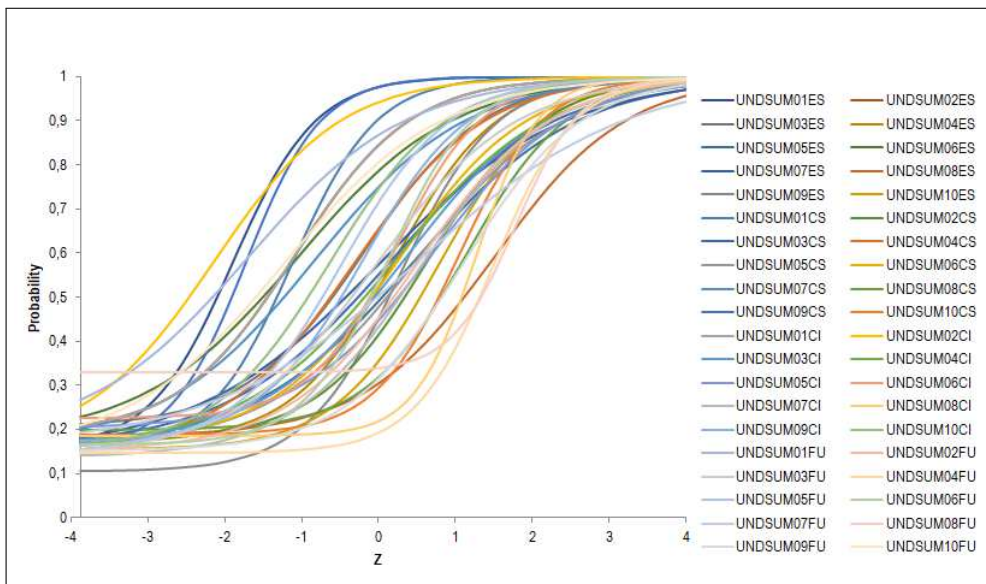


Figure 6: Summarizing skill, all ICC's.



## 7 Inferring

In Figure 7, we present the ICC's graphic representation for the inferring skill. The slope, threshold, and asymptote' parameters extreme values.

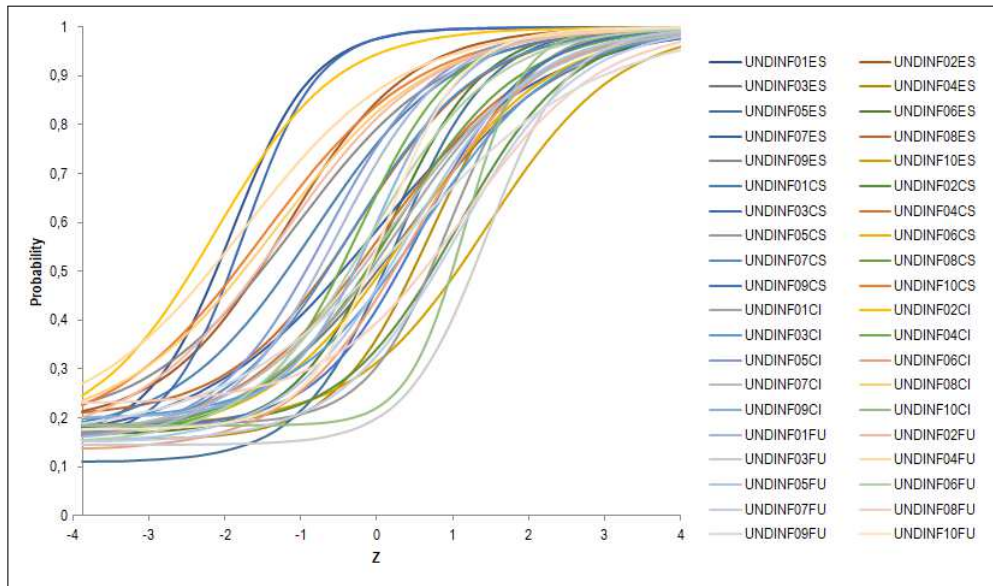


Figure 7: Inferring skill, all ICC's.

## 8 Comparing

In Figure 8, we present the ICC's graphic representation for the comparing skill. The slope, threshold, and asymptote' parameters extreme values.

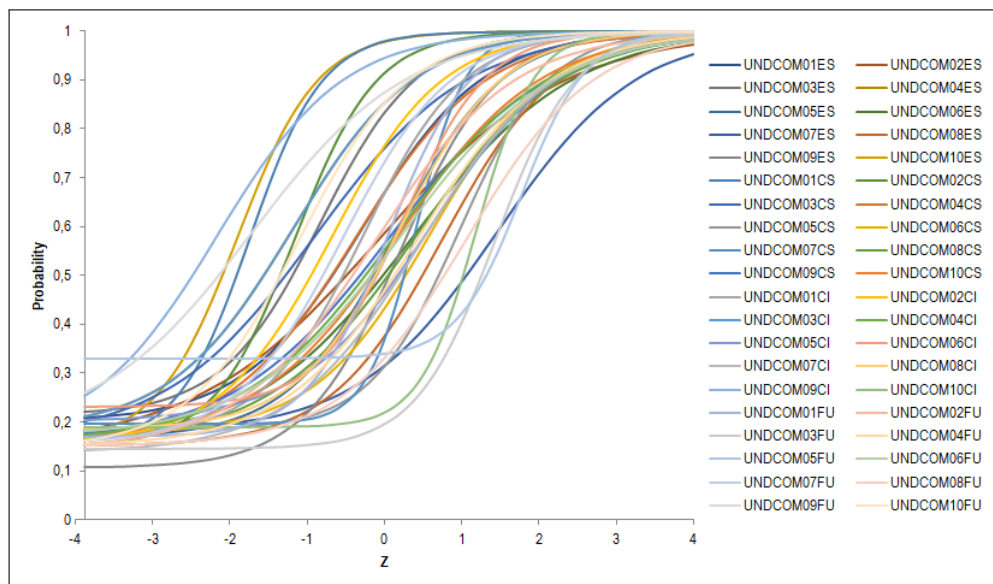


Figure 8: Inferring skill, all ICC's.

## 9 Explaining

In Figure 9, we present the ICC's graphic representation for the explaining skill. The slope, threshold, and asymptote' parameters extreme values.

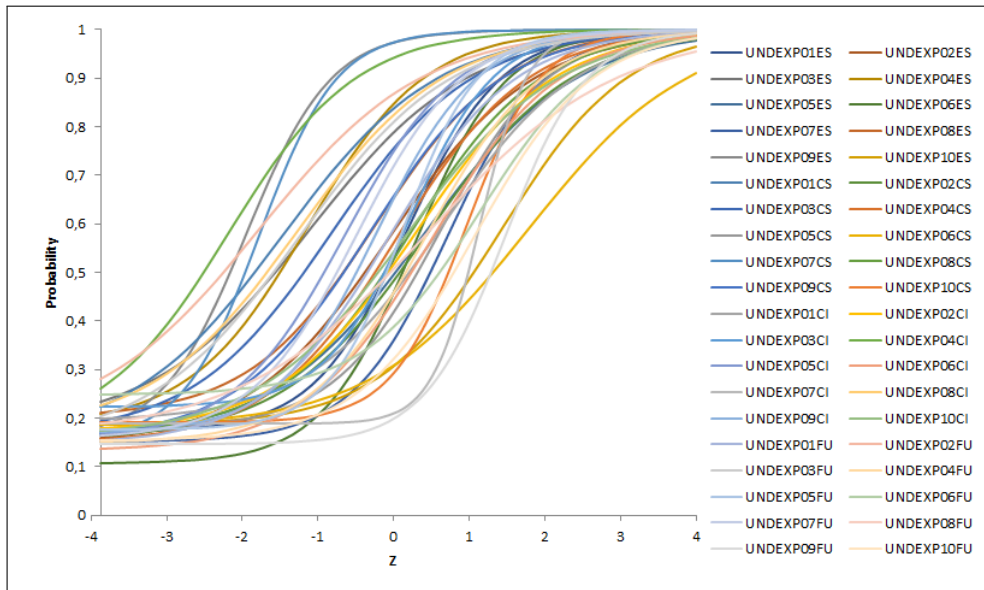


Figure 9: Explaining skill, all ICC's.

## 10 Executing

In Figure 10, we present the ICC's graphic representation for the executing skill. The slope, threshold, and asymptote' parameters extreme values.

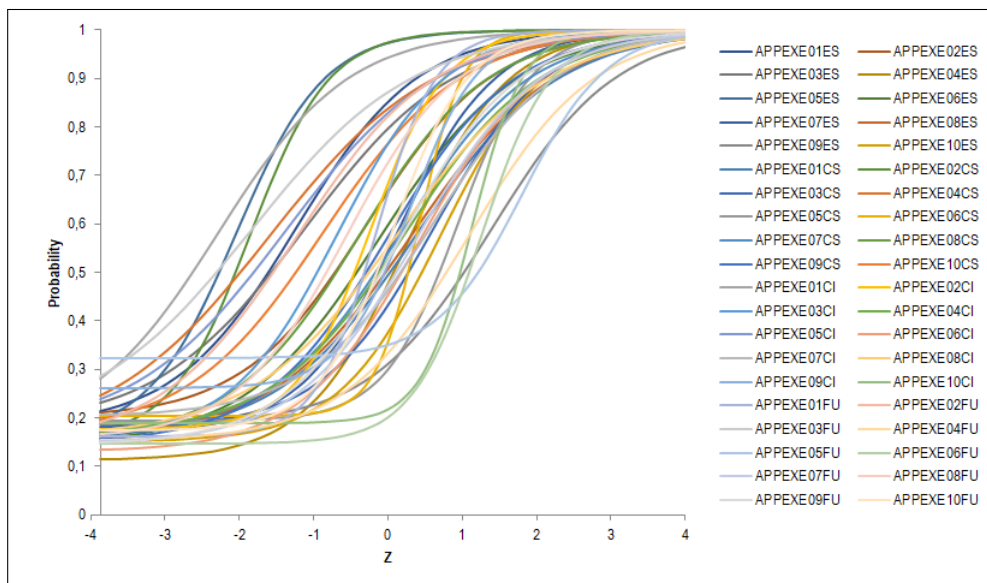


Figure 10: Executing skill, all ICC's.



## 11 Implementing

In Figure 11, we present the ICC's graphic representation for the implementing skill. The slope, threshold, and asymptote' parameters extreme values.

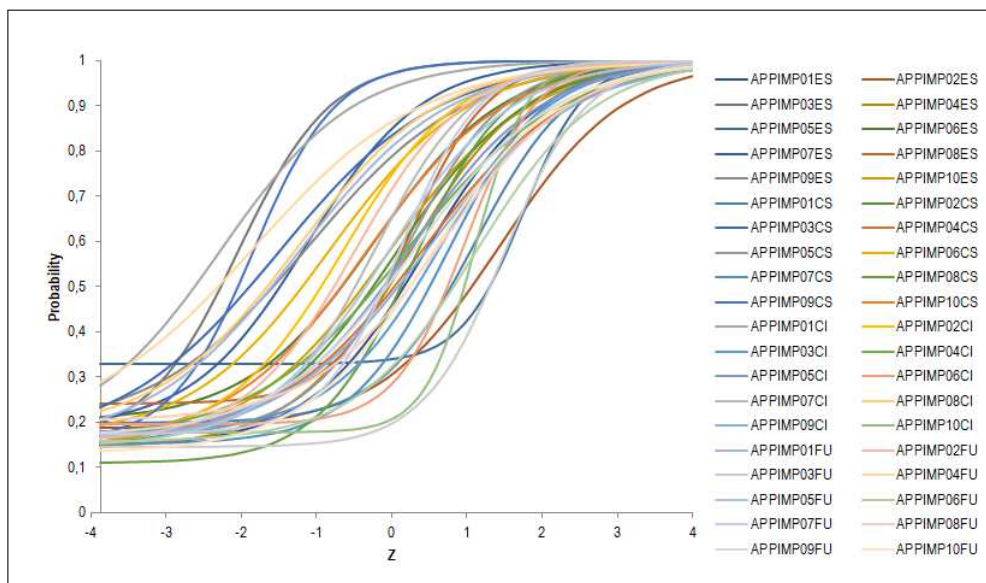


Figure 11: Implementing skill, all ICC's.

## 12 Differentiating

In Figure 12, we present the ICC's graphic representation for the differentiating skill. The slope, threshold, and asymptote' parameters extreme values.

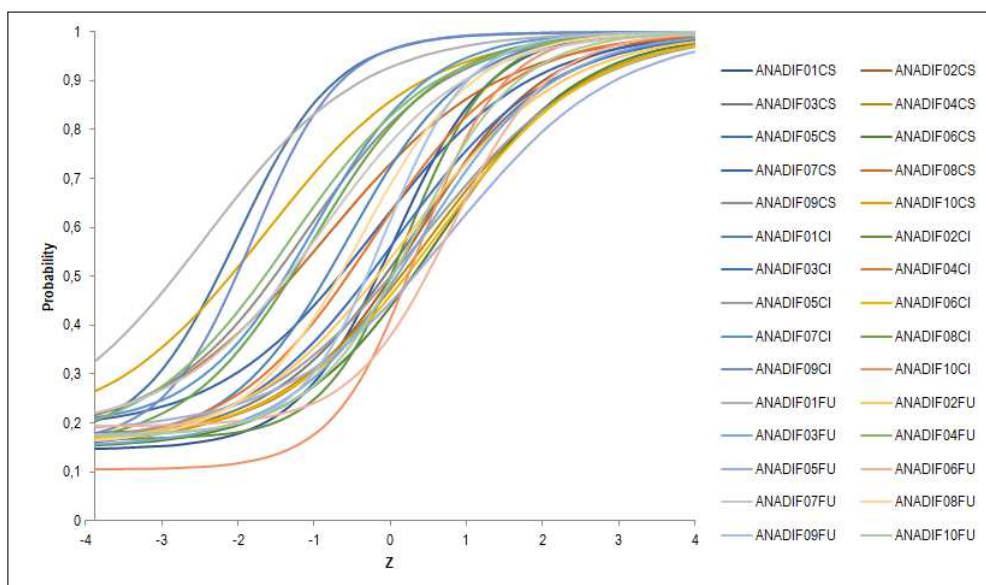


Figure 12: Differentiating skill, all ICC's.

## 13 Organizing

In Figure 13, we present the ICC's graphic representation for the organizing skill. The slope, threshold, and asymptote' parameters extreme values.

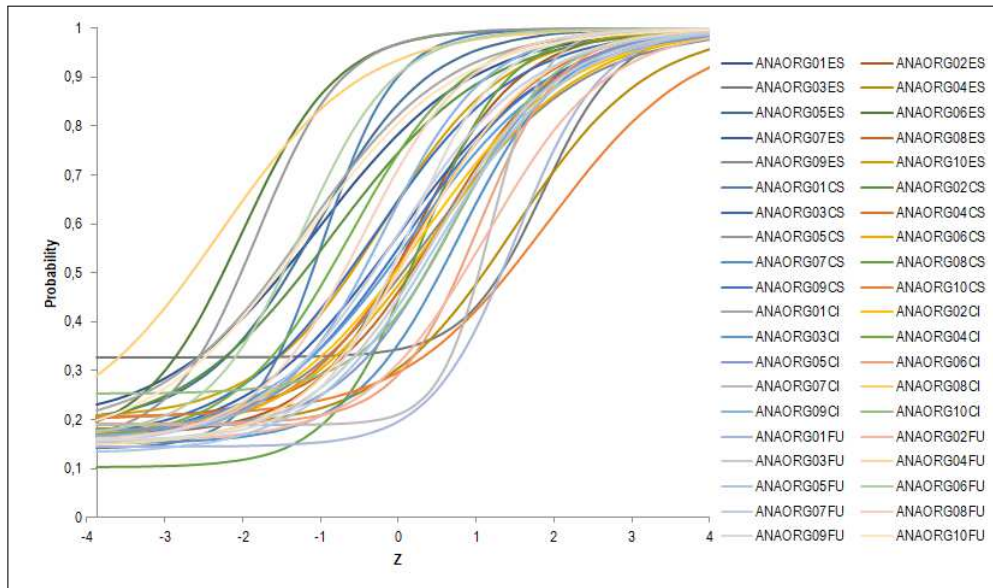


Figure 13: Organizing skill, all ICC's.

## 14 Attributing

In Figure 14, we present the ICC's graphic representation for the attributing skill. The slope, threshold, and asymptote' parameters extreme values.

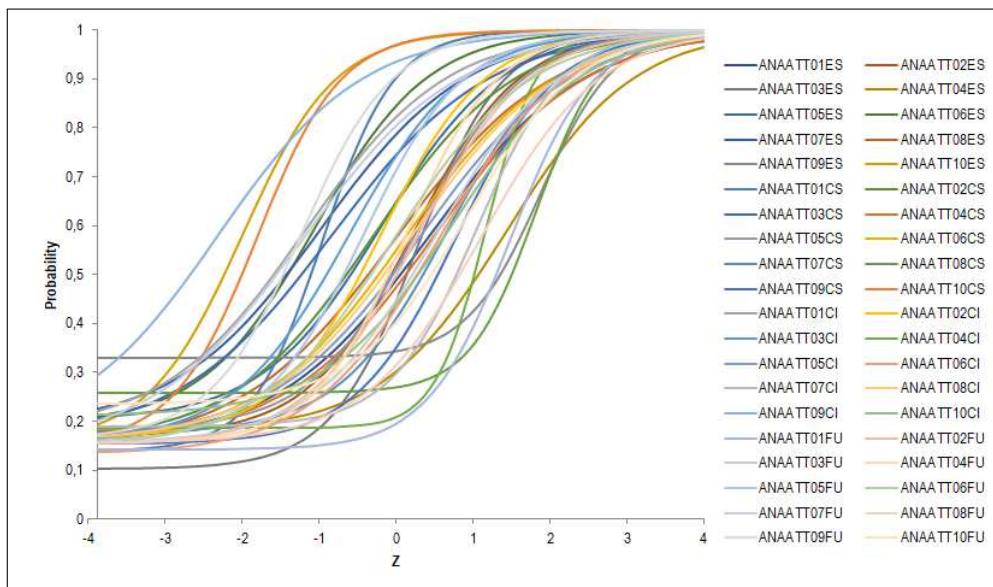


Figure 14: Attributing skill, all ICC's.

## 15 Checking

In Figure 15, we present the ICC's graphic representation for the checking skill. The slope, threshold, and asymptote' parameters extreme values.

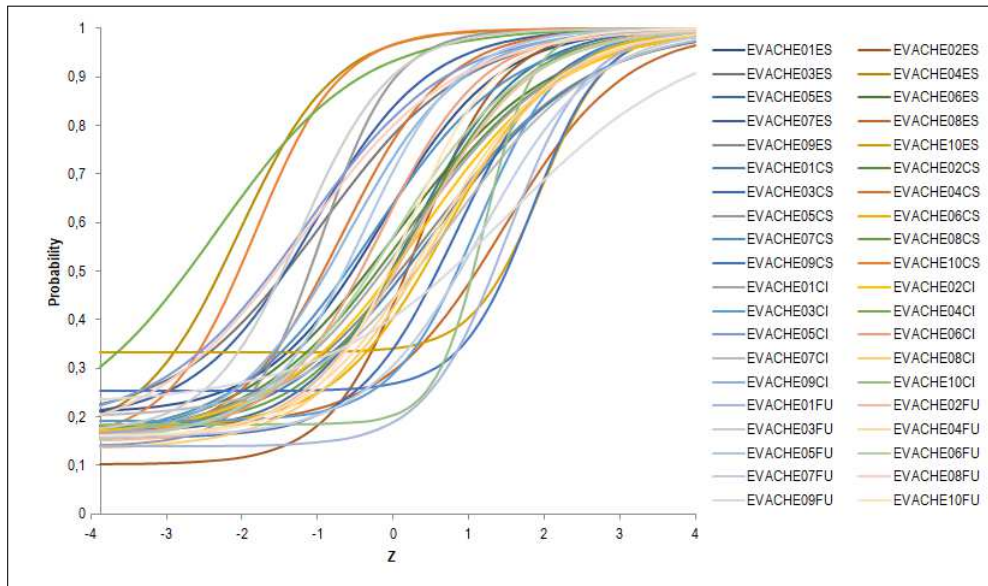


Figure 15: Checking skill, all ICC's.

## 16 Critiquing

In Figure 16, we present the ICC's graphic representation for the critiquing skill. The slope, threshold, and asymptote' parameters extreme values.

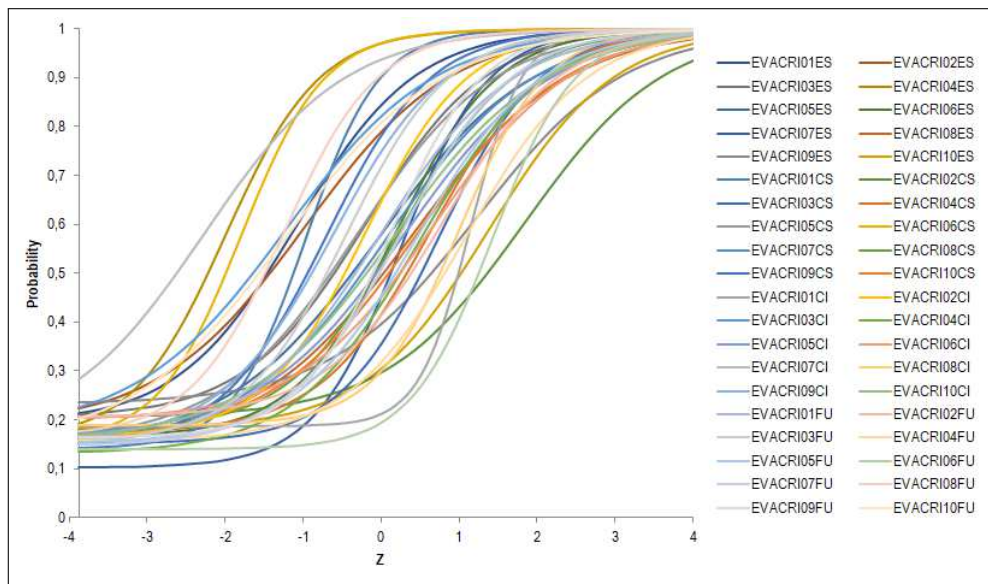


Figure 16: Critiquing skill, all ICC's.



## 17 Generating

In Figure 17, we present the ICC's graphic representation for the generating skill. The slope, threshold, and asymptote' parameters extreme values.

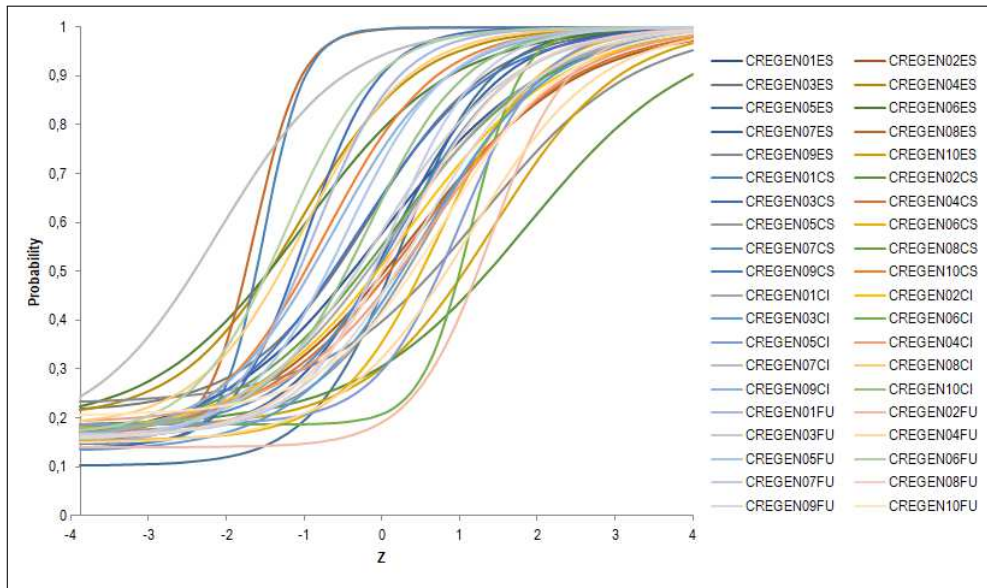


Figure 17: Generating skill, all ICC's.

## 18 Planning

In Figure 18, we present the ICC's graphic representation for the planning skill. The slope, threshold, and asymptote' parameters extreme values.

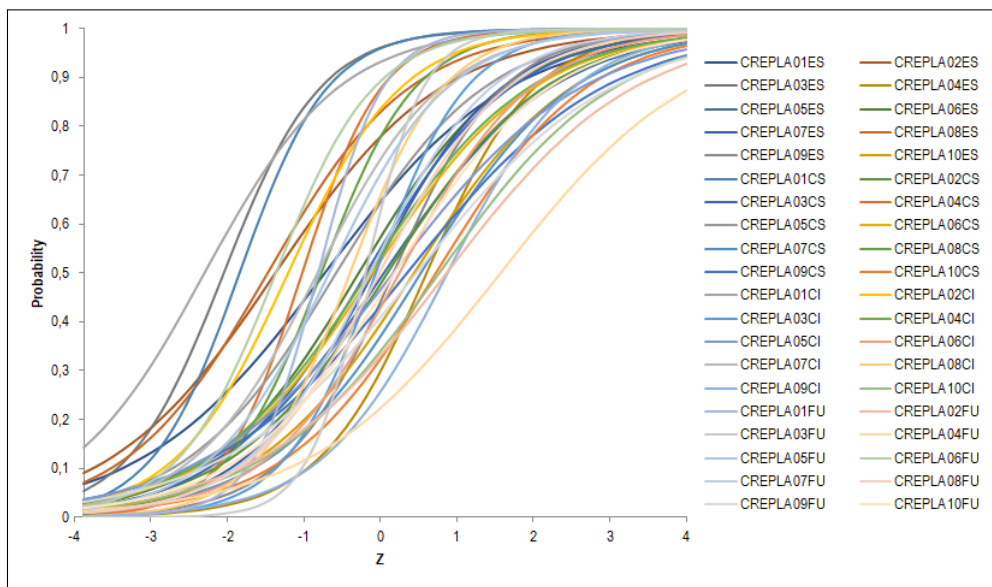


Figure 18: Planning skill, all ICC's.

## 19 Producing

In Figure 19, we present the ICC's graphic representation for the producing skill. The slope, threshold, and asymptote' parameters extreme values.

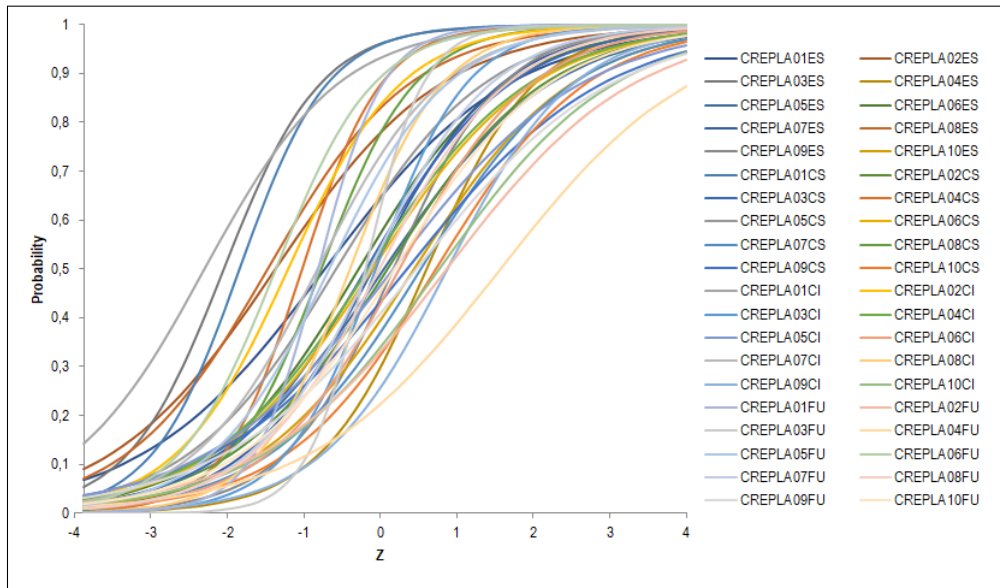


Figure 19: Producing skill, all ICC's.

# **Appendix M**

## **Items Information Function**

# 1 Recognizing

In Figure 1, we present the IIF's graphic representation for the recognizing skill, in which each item provides information in a specific latent trait region.

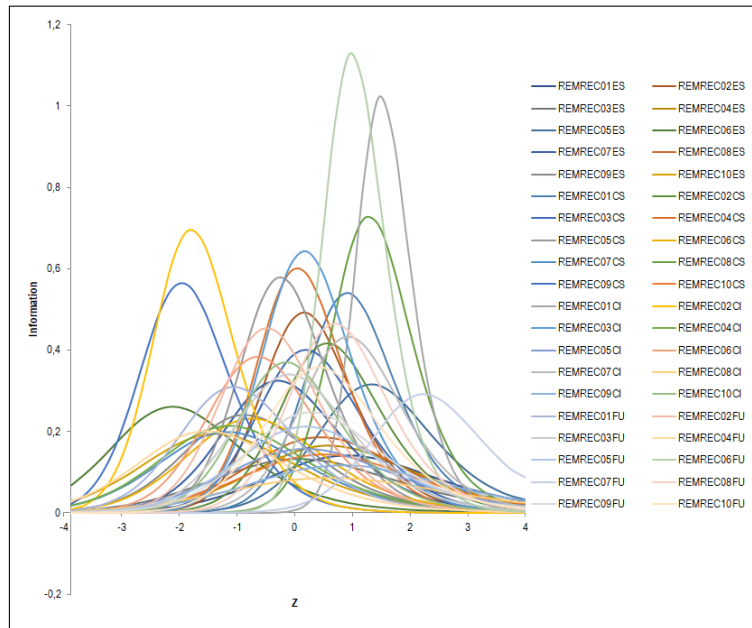


Figure 1: Recognizing skill, all IIF's.

# 2 Recalling

In Figure 2, we present the IIF's graphic representation for the recalling skill, in which each item provides information in a specific latent trait region.

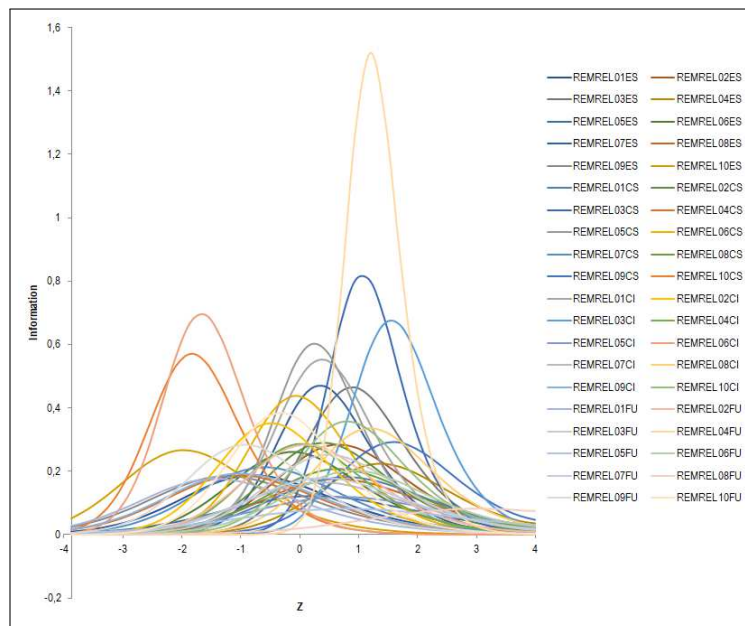


Figure 2: Recalling skill, all IIF's.

### 3 Interpreting

In Figure 3, we present the IIF's graphic representation for the interpreting skill, in which each item provides information in a specific latent trait region.

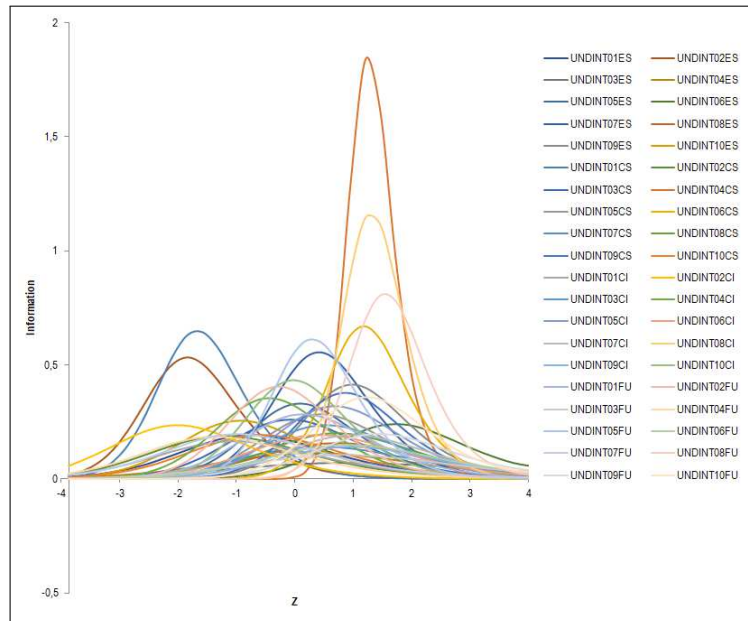


Figure 3: Interpreting skill, all IIF's.

### 4 Exemplifying

In Figure 4, we present the IIF's graphic representation for the exemplifying skill, in which each item provides information in a specific latent trait region.

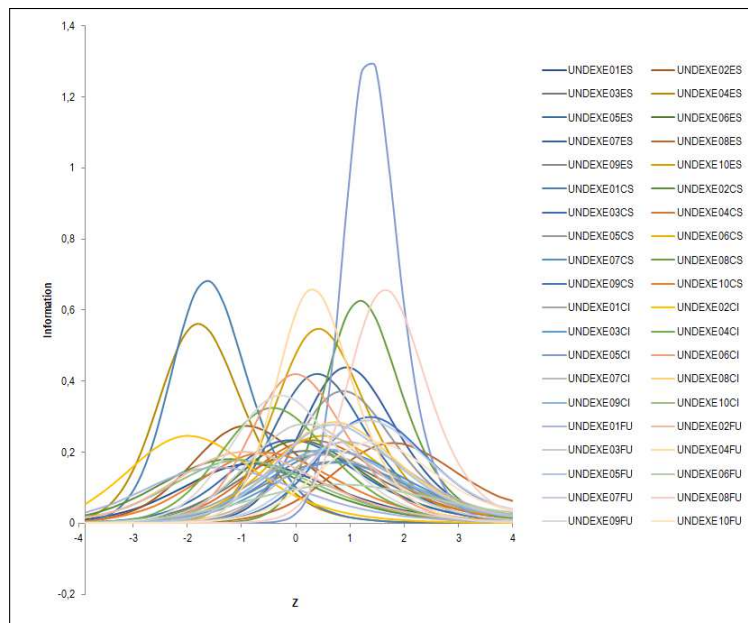


Figure 4: Exemplifying skill, all IIF's.



## 5 Classifying

In Figure 5, we present the IIF's graphic representation for the classifying skill, in which each item provides information in a specific latent trait region.

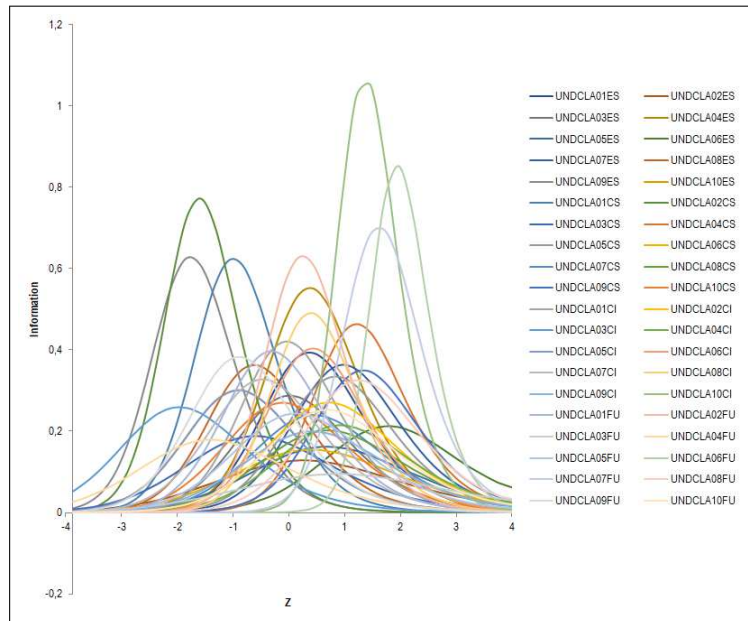


Figure 5: Classifying skill, all IIF's.

## 6 Summarizing

In Figure 6, we present the IIF's graphic representation for the summarizing skill, in which each item provides information in a specific latent trait region.

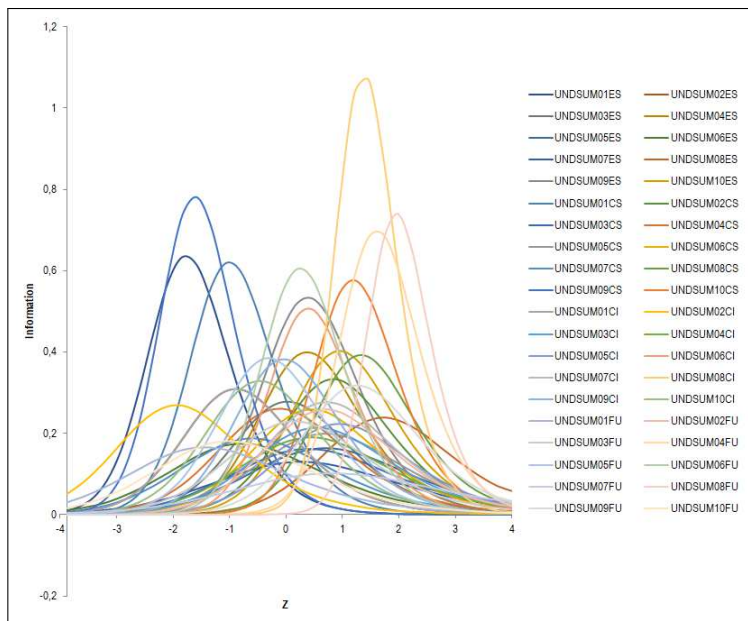


Figure 6: Summarizing skill, all IIF's.

## 7 Inferring

In Figure 7, we present the IIF's graphic representation for the inferring skill, in which each item provides information in a specific latent trait region.

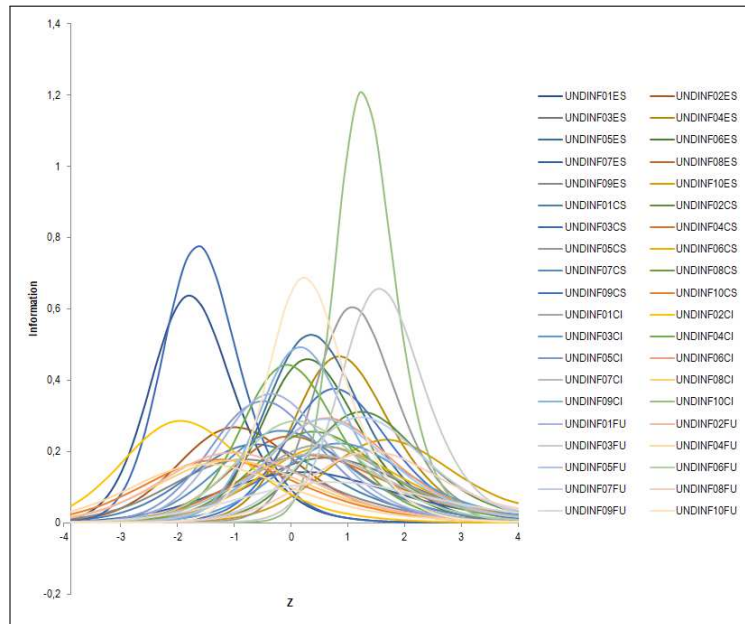


Figure 7: Inferring skill, all IIF's.

## 8 Comparing

In Figure 8, we present the IIF's graphic representation for the comparing skill, in which each item provides information in a specific latent trait region.

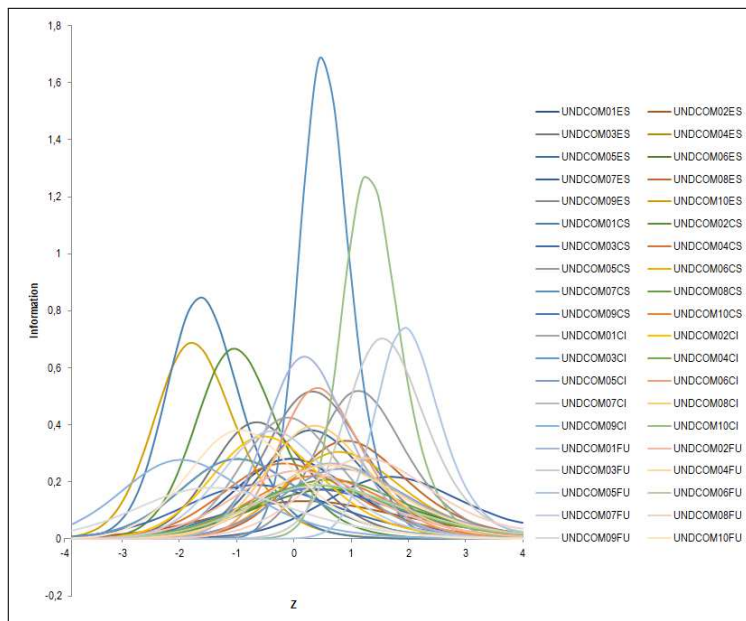


Figure 8: Comparing skill, all IIF's.

## 9 Explaining

In Figure 9, we present the IIF's graphic representation for the explaining skill, in which each item provides information in a specific latent trait region.

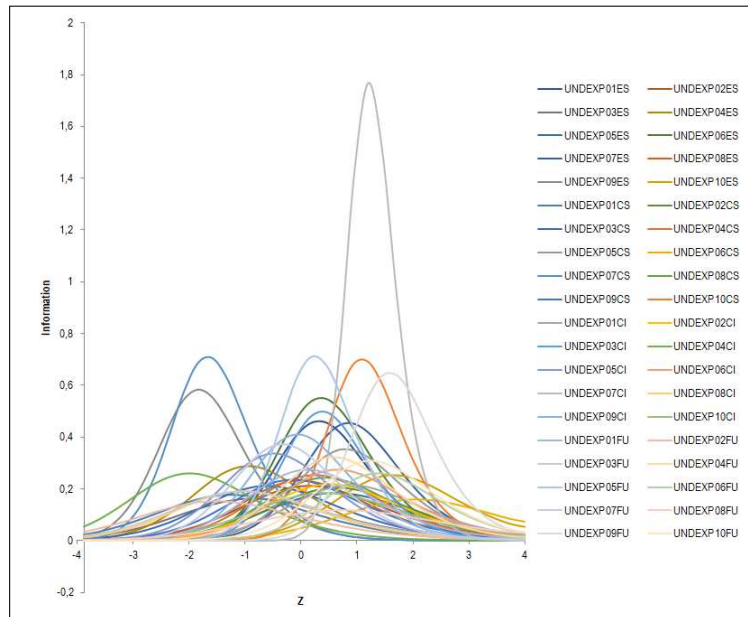


Figure 9: Explaining skill, all IIF's.

## 10 Executing

In Figure 10, we present the IIF's graphic representation for the executing skill, in which each item provides information in a specific latent trait region.

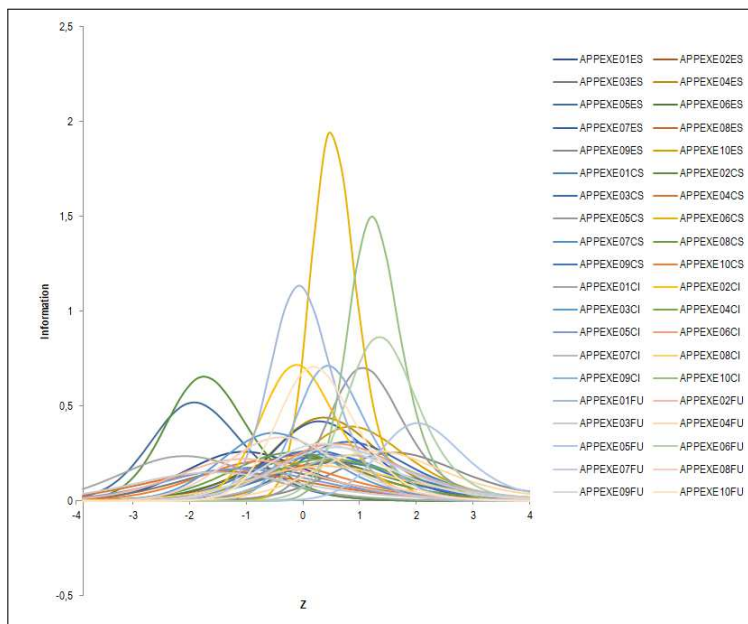


Figure 10: Executing skill, all IIF's.

## 11 Implementing

In Figure 11, we present the IIF's graphic representation for the implementing skill, in which each item provides information in a specific latent trait region.

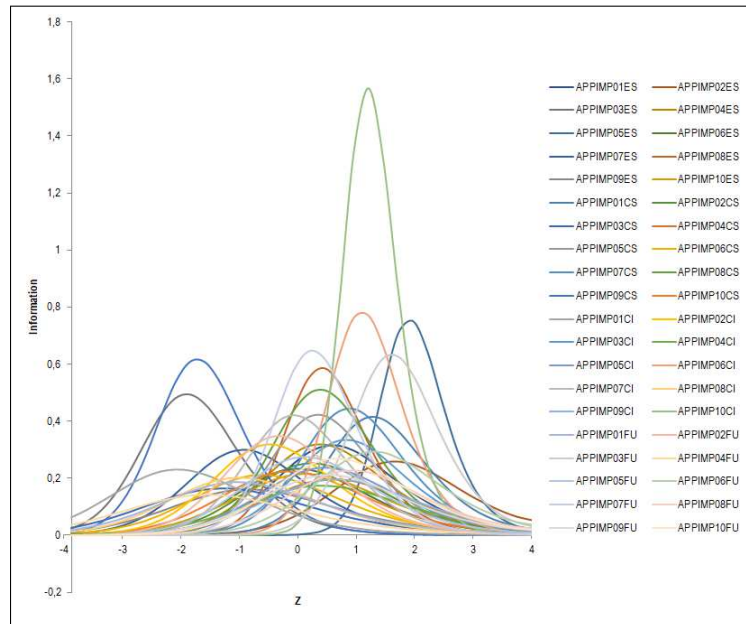


Figure 11: Implementing skill, all IIF's.

## 12 Differentiating

In Figure 12, we present the IIF's graphic representation for the differentiating skill, in which each item provides information in a specific latent trait region.

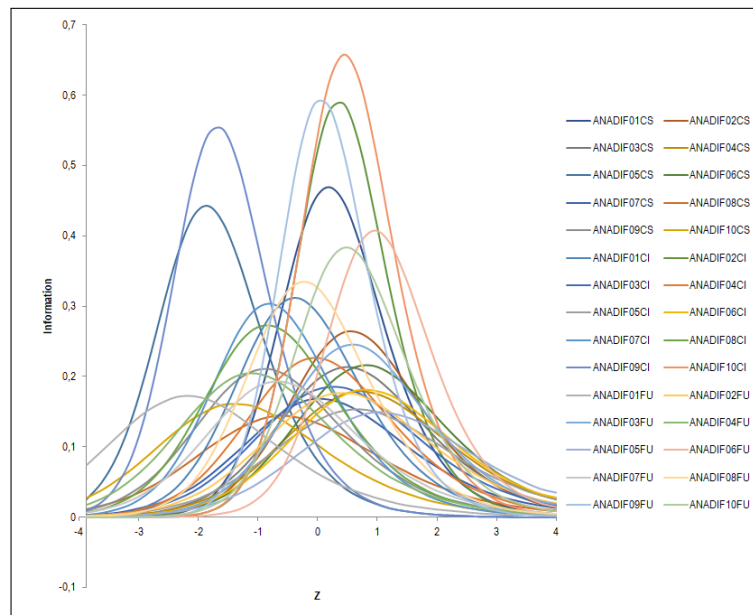


Figure 12: Differentiating skill, all IIF's.

## 13 Organizing

In Figure 13, we present the IIF's graphic representation for the organizing skill, in which each item provides information in a specific latent trait region.

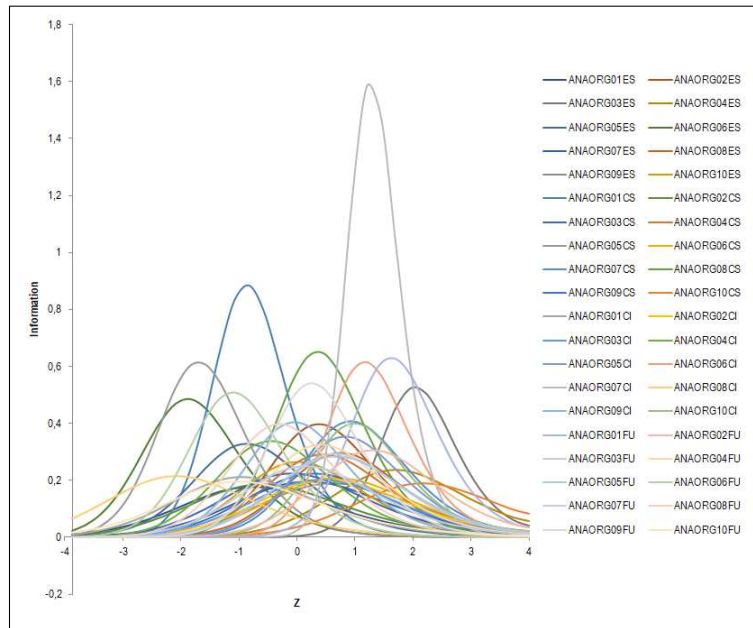


Figure 13: Organizing skill, all IIF's.

## 14 Attributing

In Figure 14, we present the IIF's graphic representation for the attributing skill, in which each item provides information in a specific latent trait region.

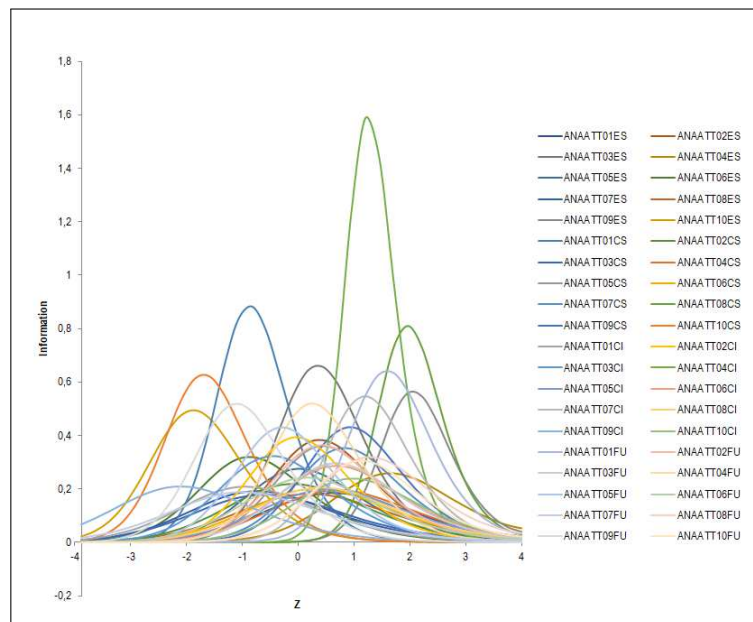


Figure 14: Attributing skill, all IIF's.

## 15 Checking

In Figure 15, we present the IIF's graphic representation for the checking skill, in which each item provides information in a specific latent trait region.

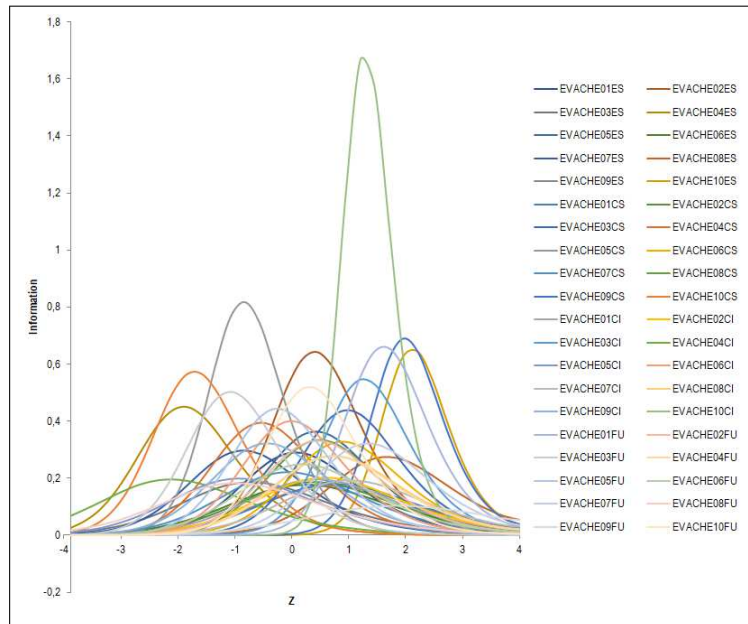


Figure 15: Checking skill, all IIF's.

## 16 Critiquing

In Figure 16, we present the IIF's graphic representation for the critiquing skill, in which each item provides information in a specific latent trait region.

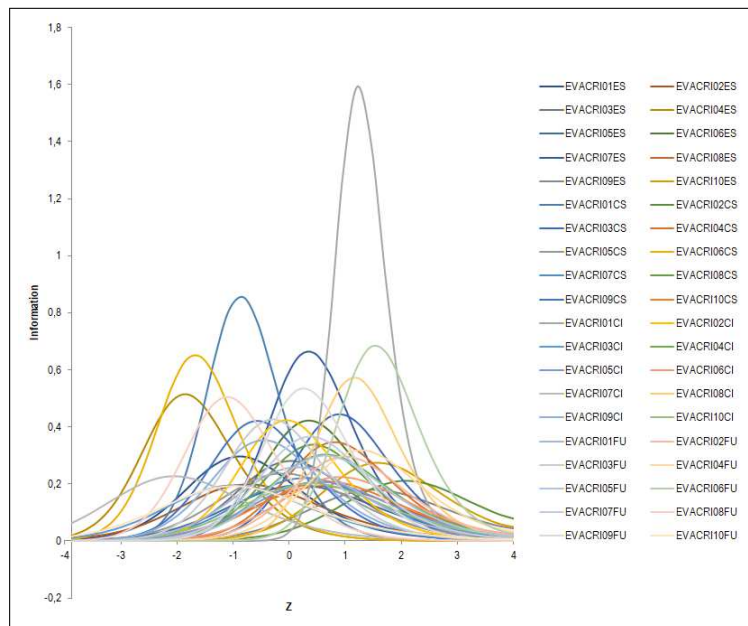


Figure 16: Critiquing skill, all IIF's.



## 17 Generating

In Figure 17, we present the IIF's graphic representation for the generating skill, in which each item provides information in a specific latent trait region.

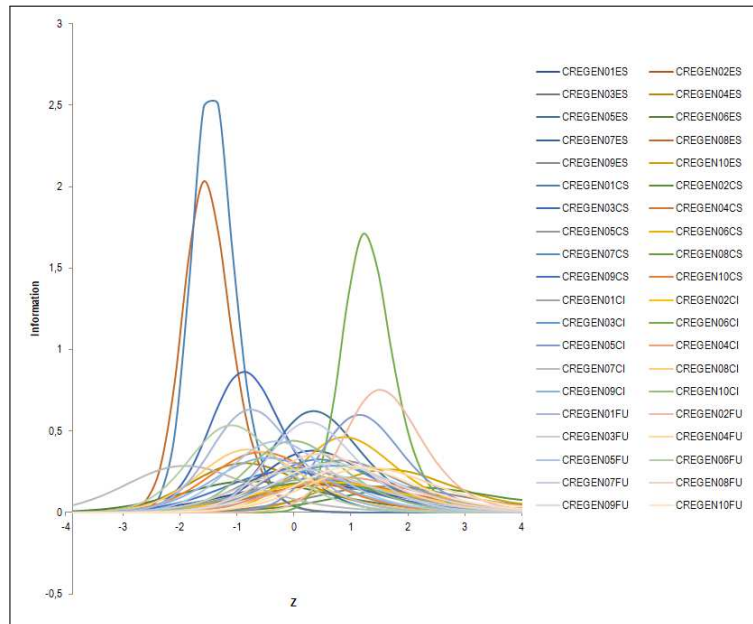


Figure 17: Generating skill, all IIF's.

## 18 Planning

In Figure 18, we present the IIF's graphic representation for the planning skill, in which each item provides information in a specific latent trait region.

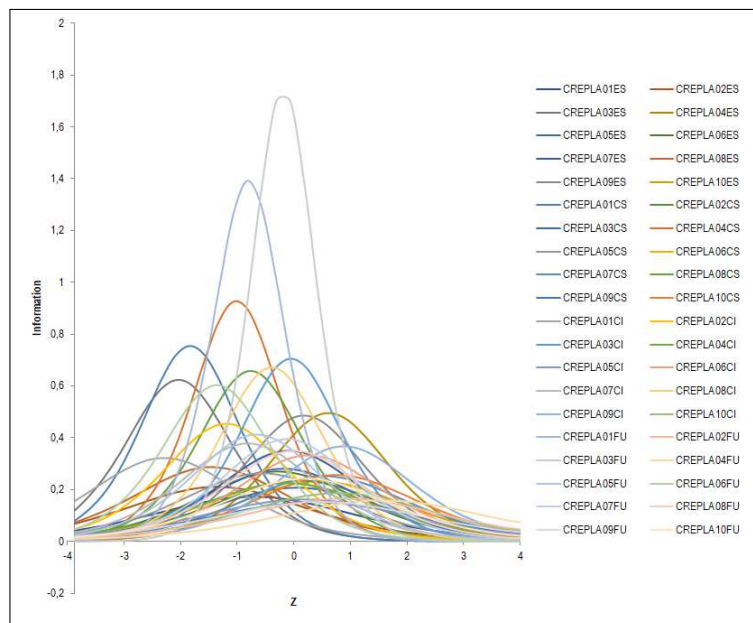


Figure 18: Planning skill, all IIF's.

## 19 Producing

In Figure 19, we present the IIF's graphic representation for the producing skill, in which each item provides information in a specific latent trait region.

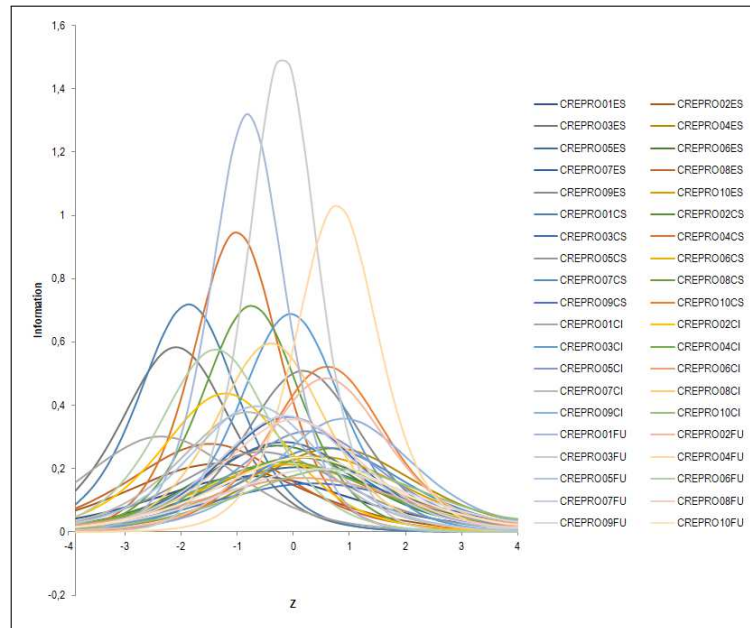


Figure 19: Producing skill, all IIF's.



# Appendix N

## Script for Adaptive Item Selection and Skill Estimation

---

```
1 public class Script{
2
3     private static int iS, id, idStudent, idmirror, max = 10, admint;
4
5     private static float difErro, erro, erro1, erro2, erroAux, theta, numTheta, denTh
6
7     //Insert the calibrated items for the slope parameter
8     private static float[] a = new float[] {};
9
10    //Insert the calibrated items for the threshold parameter
11    private static float[] b = new float[] {};
12
13    //Insert the calibrated items for the asymptote parameter
14    private static float[] c = new float[] {};
15
16    //Probability to hit the item
17    private static float[] p = new float[10];
18
19    //Item Information Function
20    private static float[] i = new float[10];
21
22    //Item Exposure Control
23    public static int[] ic = new int[] {0,0,0,0,0,0,0,0,0,0};
```

```
24
25     private static int[][] mirror = new int[][] {
26         {0,0,0,0,0,0,0,0,0,0,0},
27         {1,1,1,1,1,1,1,1,1,1,1},
28         //Insert database in this format
29             {99,1,0,0,0,0,0,0,0,0,0}
30     };
31
32     public static void main(String[] args) {
33         iS = 0;
34         idStudant = mirror [iS] [0];
35     do{
36         admint = 0;
37         theta = 0.0f;
38         do{
39             newItem();
40             ic[id] = 1;
41             admint = admint + 1;
42             //Stopping criterion for Z2
43             //} while(admint<10);
44             // Stopping criterion for Z3
45             //} while(admint<10 && difErro >0.01); //Stopping criterion for Z4
46         }while(admint<11|| (admint<15 && difErro >0.01));
47         System.out.printf("%.3f\n", theta);
48         //System.out.printf("%d \n", admint);
49         ic = new int[] {0,0,0,0,0,0,0,0,0,0,0};
50             iS = iS + 1;
51         if(iS<mirror.length){
52             idStudant = mirror [iS] [0];
53         }
54         denTheta = 0.0f;
55             numTheta = 0.0f;
56             erro = 0.0f;
57     } while(iS<mirror.length);
58 }
59
60     public static void newItem(){
```

```
61         probI();
62         infoI();
63         newTheta();
64     }
65
66     public static void probI(){
67         for (int index = 0; index < p.length; index++){
68             //3 - Parameter Logistic Model.
69             float den = (float) (1.0f + Math.exp(-a[index] * (theta - b[i[index]])));
70             float pTheta = c[index]+((1.0f-c[index])/den);
71
72             //2 - Parameter Logistic Model.
73             //float den = (float) (1.0f + Math.exp(-a[index] * (theta - b[i[index]])));
74             //float pTheta = (1.0f/ den);
75
76             p[index] = pTheta;
77         }
78     }
79
80     public static void infoI(){
81         float max = 0.0f;
82         for (int index = 0; index < i.length; index++){
83             //3-parameter Logistic Model.
84             float iltem = (a[index]*a[index])*((1.0f-p[index])/p[index])*((p[index]-c[index])/p[index]);
85
86             //2-Parameter Logistic Model.
87             //float iltem = (a[index]*a[index])*p[index]*(1.0f-p[index]);
88
89             i[index] = iltem;
90         }
91
92         for (int index = 0; index < i.length; index++){
93             if (i[index] > max){
94                 if (ic[index] == 0){
95                     max = i[index];
96                     id = index;
97                     idmirror = id + 1;
```

---

```
98         }
99     }
100 }
101 }
102
103 public static void newTheta(){
104     numTheta = numTheta+(a[id]*(mirror[iS][idmirror]-p[id]));
105     denTheta = denTheta + ((a[id]*a[id])*p[id]*(1-p[id]));
106     theta = theta + (numTheta/denTheta);
107     if (theta < -3.0f){
108         theta = -3.0f;
109     }
110     else if (theta > 3.0f){
111         theta = 3.0f;
112     }
113     erro = (float) (1/Math.sqrt(denTheta));
114
115     if (admint == 0){
116         erro1 = 0;
117         erro2 = erro;
118         difErro = erro2 - erro1;
119     } else if (admint > 0){
120         erro1 = erro;
121         erro2 = erroAux;
122         difErro = erro2 - erro1;
123     }
124     erroAux = erro;
125 }
126 }
```

---

# Appendix O

## Trust in Python

Prezado participante,

Solicitamo-lhe a avaliação da sua confiança na sintaxe e semântica da linguagem de programação Python. Apresentamos uma escala do tipo LIKERT, com as opções para você realizar sua avaliação, considerando 1 (de maneira alguma confiante) a 7 (absolutamente confiante). Se um termo ou tarefa específica não lhe for totalmente familiar, marque 1.

Item	1	2	3	4	5	6	7
Escrevo instruções Python sintaticamente corretas							
Entendo a estrutura da linguagem do Python e o uso das palavras reservadas							
Escrevo blocos de código logicamente corretos usando Python							
Escrevo um programa Python que exiba uma mensagem de saudação							
Escrevo um programa Python que calcule a média de três números							
Uso funções internas disponíveis nas várias bibliotecas Python							
Construo minhas próprias bibliotecas Python							
Escrevo um pequeno programa Python, devido a um pequeno problema que me é familiar							
Escrevo um programa Python de tamanho razoável que possa resolver um problema que é apenas vagamente familiar para mim							
Escrevo um programa Python longo e complexo para resolver qualquer problema, desde que as especificações estejam claramente definidas							
Organizo e projeto meu programa de maneira modular							

Item	1	2	3	4	5	6	7
Compreendo o paradigma orientado a objetos							
Identifico os objetos no domínio do problema e declare, defina e use-os							
Faço uso de uma função pré-escrita, dada uma declaração claramente rotulada da função							
Faço uso de uma classe que já está definida, dada uma declaração claramente rotulada da classe							
Depuro (corrijo todos os erros) um programa longo e complexo que eu havia escrito e faça com que funcione							
Compreendo um programa longo e complexo de múltiplos arquivos							
Concluo um projeto de programação se alguém me mostrar como resolver o problema primeiro							
Concluo um projeto de programação se eu tiver apenas o manual de referência do idioma para obter ajuda							
Concluo um projeto de programação se eu puder pedir ajuda a alguém, se eu ficar preso							
Concluo um projeto de programação quando outra pessoa me ajudar a começar							
Concluo um projeto de programação se eu tiver muito tempo para concluir o programa							
Concluo um projeto de programação se eu tivesse apenas o recurso de ajuda interno para obter assistência							
Encontro maneiras de superar o problema se eu ficar preso em um ponto enquanto estiver trabalhando em um projeto de programação							
Crio uma estratégia adequada para um determinado projeto de programação em pouco tempo							
Gerencio meu tempo de forma eficiente se eu tivesse um prazo premente em um projeto de programação							
Rastreio mentalmente a execução de um programa longo e complexo que me foi dado							
Reescrevo longas partes confusas do código para ficar mais legível e claro							
Encontro uma maneira de me concentrar no meu programa, mesmo quando houver muitas distrações ao redor de mim							
Encontro maneiras de me motivar a programar, mesmo que a área problemática não me interesse							