

University of Alabama in Huntsville

LOUIS

Dissertations

UAH Electronic Theses and Dissertations

2011

Using a visual programming language to bridge the cognitive gap between a novice's mental model and program code

Bryan J. Smith

Follow this and additional works at: <https://louis.uah.edu/uah-dissertations>

Recommended Citation

Smith, Bryan J., "Using a visual programming language to bridge the cognitive gap between a novice's mental model and program code" (2011). *Dissertations*. 291.
<https://louis.uah.edu/uah-dissertations/291>

This Dissertation is brought to you for free and open access by the UAH Electronic Theses and Dissertations at LOUIS. It has been accepted for inclusion in Dissertations by an authorized administrator of LOUIS.

**USING A VISUAL PROGRAMMING LANGUAGE TO BRIDGE THE
COGNITIVE GAP BETWEEN A NOVICE'S MENTAL MODEL AND
PROGRAM CODE**

by

BRYAN J. SMITH

A DISSERTATION

**Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
The Department of Computer Science
to
The School of Graduate Studies
of
The University of Alabama in Huntsville**

**HUNTSVILLE, ALABAMA
2011**

In presenting this proposal in partial fulfillment of the requirements for a doctoral degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his absence, by the Chair of the Department or the Dean of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this dissertation.

Brya J. Hill
Student Signature

3/10/11
Date

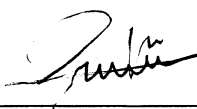
DISSERTATION APPROVAL FORM

Submitted by Bryan J. Smith in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and accepted on behalf of the Faculty of the School of Graduate Studies by the dissertation committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

 Committee Chair

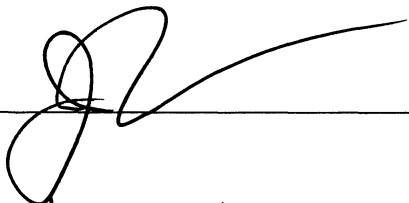
 3/9/11

 3/9/11

 3/9/11

 3/9/2011

 3-9-11
Department Chair

 College Dean

 5-13-11 Graduate Dean

ABSTRACT

The School of Graduate Studies
The University of Alabama in Huntsville

Degree Doctor of Philosophy College/Dept. Science/Computer Science

Name of Candidate Bryan J. Smith

Title Using a Visual Programming Language to Bridge the Cognitive Gap Between a Novice's Mental Model and Program Code

Current research suggests that many students do not know how to program very well at the conclusion of their introductory programming course. We believe that a reason novices have such difficulties learning programming is because engineering novices often learn through a lecture format where someone with programming knowledge lectures to novices, the novices attempt to absorb the content, and then reproduce it during exams. By primarily appealing to programming novices who prefer to understand visually, we research whether programming novices understand programming better if computer science concepts are presented using a visual programming language than if these programs are presented using a text-based programming language. This method builds upon previous research that suggests that most engineering students are visual learners, and we propose that using a flow-based visual programming language will address some of the most important and difficult topics to novices of programming. We use an existing flow-model tool, RAPTOR, to test this method, and share the program understanding results using this theory.

Abstract Approval: Committee Chair

Department Chair

Graduate Dean

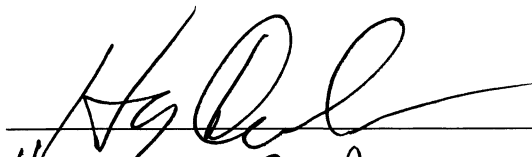

Hegger S. Campanato
Rhonda Kay Shede 5-13-11

TABLE OF CONTENTS

	Page
LIST OF FIGURES	x
LIST OF TABLES	xxiv
LIST OF ACRONYMS	xxix
CHAPTER	
I. INTRODUCTION	1
1.1 Who is a Novice?	3
1.2 Types of Novices	6
1.2.1 The Kolb Cycle.....	9
1.2.2 The Myers-Brigg Type Indicator (MBTI)	10
1.2.3 Lateralization of Brain Function.....	12
1.2.4 The Index of Learning Styles	12
1.3 The Most Underserved Novices: Addressing the Visual/Verbal Dimension	17
1.3.1 Visual Programming (VP)	17
1.3.2 Limits of Visual Programming	20
1.3.3 Empirical Evidence	23
1.4 Secondly Addressing the Sequential/Global Dimension	23
1.5 Secondly Addressing the Active/Reflective Dimension	24
1.6 Considering the Sensing/Intuitive Dimension	25
1.7 Reducing Experimental Bias.....	26
1.8 Summary	28
II. RELATED WORK.....	29
2.1 Technologies that Support Learning Programming	29
2.1.1 Narrative Tools.....	32
2.1.2 Visual Programming Tools.....	32
2.1.3 Flow-Model Tools	33

2.1.4	Specialized Output Realizations	37
2.1.5	Tiered Language Tools	37
2.2	Studies using Technologies that Support Learning	39
2.2.1	Using Multimedia Recordings	39
2.2.2	Using Alice	40
2.2.3	Using RAPTOR at the U.S. Air Force Academy	43
2.3	Summary	45
III.	GENERAL APPROACH	46
3.1	The First Pilot Study	49
3.2	Experiments with a Flow-Model Implementation	54
3.3	Our Second Pilot and Flow-Model Experiment	55
3.4	Validating our Flow Model Experiment: The Test	58
3.5	The Important and Difficult Introductory Computer Science Concepts	63
3.6	The Script	65
3.6.1	Introduction and Agenda	66
3.6.2	A Computer Program	66
3.6.3	Sequence	67
3.7	Summary	70
IV.	RAW RESULTS OF OUR EXPERIMENTS	71
4.1	The First Pilot Study	72
4.1.1	Frequency Distribution	73
4.2	The Second Pilot Study	84
4.2.1	Frequency Distribution	85
4.2.2	Group Statistics and Independent Samples Test	95
4.2.3	Internal Reliability using Cronbach's Alpha	95
4.3	The Experiment	96
4.3.1	The Frequency Distribution	96
4.3.2	Overall Group Statistics and Independent Samples Test	108

4.3.3	Internal Reliability using Cronbach's Alpha.....	110
4.4	Summary	110
V.	CORRELATIONS AND COMPARISONS FOUND IN THE FINAL EXPERIMENT	111
5.1	Comparisons to Measure our Hypothesis	112
5.2	Correlations and Comparisons Involving Test Grade	114
5.2.1	Correlations and Comparisons Involving the Visual/Verbal Style	115
5.2.2	Correlations Involving the Visual/Verbal Style Considering Gender	116
5.2.3	Correlations Involving the Visual/Verbal Style Considering Time of Day	116
5.2.4	Correlations and Comparisons Involving the Sequential/Global Style	116
5.2.5	Correlations Involving the Sequential/Global Style Considering Gender	117
5.2.6	Correlations Involving the Sequential/Global Style Considering Time of Day	117
5.2.7	Correlations and Comparisons Involving the Sensing/Intuitive Style	117
5.2.8	Correlations Involving the Sensing/Intuitive Style Considering Gender	119
5.2.9	Correlations Involving the Sensing/Intuitive Style Considering Time of Day	120
5.2.10	Correlations and Comparisons Involving the Active/Reflective Style.....	121
5.2.11	Correlations Involving the Active/Reflective Style Considering Gender	121
5.2.12	Correlations Involving the Active/Reflective Style Considering Time of Day	121
5.2.13	Correlations Involving Test Grade with General Measures	122
5.2.14	Correlations Involving Test Grade with General Measures Considering Gender	123
5.2.15	Correlations Involving Test Grade with General Measures Considering Time of Day	123
5.2.16	Comparisons of Test Grade Considering Gender	124
5.2.17	Comparisons of Test Grade Considering Time of Day	124
5.3	Correlations and Comparisons Involving the Index of Learning Styles	124
5.3.1	Correlations Involving the Visual/Verbal Style	125
5.3.2	Correlations Involving the Visual/Verbal Style Considering Gender	126
5.3.3	Correlations Involving the Visual/Verbal Style Considering Time of Day	127
5.3.4	Correlations Involving the Sequential/Global Style	128
5.3.5	Correlations Involving the Sequential/Global Style Considering Gender	129

5.3.6	Correlations Involving the Sequential/Global Style Considering Time of Day	131
5.3.7	Correlations Involving the Sensing/Intuitive Style	132
5.3.8	Correlations Involving the Sensing/Intuitive Style Considering Gender	133
5.3.9	Correlations Involving the Sensing/Intuitive Style Considering Time of Day	134
5.3.10	Correlations Involving the Active/Reflective Style	135
5.3.11	Correlations Involving the Active/Reflective Style Considering Gender	137
5.3.12	Correlations Involving the Active/Reflective Style Considering Time of Day	138
5.3.13	Comparisons of General Demographics Considering Gender	139
5.3.14	Comparisons of General Demographics Considering Time of Day	139
5.4	Summary	140
VI.	DISCUSSION	141
6.1	Test Grade and Visual/Verbal Correlation Explanation	141
6.2	Test Grade and Flow Teaching Style and Text Teaching Style Explanation	142
6.3	Sensing/Intuitive and Test Grade Correlation Explanation	143
6.4	Sensing/Intuitive Experiment	144
6.5	Correlations between Sensing/Intuitive and Sequential/Global	144
6.6	Sequential/Global and Visual/Verbal	145
6.7	Sensing/Intuitive and Visual/Verbal Correlation Explanation	145
VII.	FUTURE WORK	147
7.1	Improvements to RAPTOR	147
7.2	Suggestions to Improve Computer Science Curriculum	148
7.2.1	Flow and Text Separately	148
7.2.2	Flow and Text Together	149
7.3	What's Next?	150
VIII.	CONCLUSION	152
	APPENDIX A: THE QUESTIONNAIRE	155
	APPENDIX B: THE SCRIPT FOLLOWED DURING THE EXPERIMENT	160
B.1	Introduction and Agenda	160

B.2	A Computer Program	161
B.3	Sequence	161
B.4	Variable.....	165
B.5	User input.....	169
B.6	Alternative	173
B.7	Iteration.....	181
APPENDIX C: The Concepts Presented To Participants in the Second Pilot and Experiment.....		194
APPENDIX D: The Text, Second Pilot.....		287
APPENDIX E: The Test, Experiment		296
REFERENCES		312

LIST OF FIGURES

Figure	Page
1.1 Acceptable Evidence 1	7
1.2 Acceptable Evidence 2.....	8
1.3 Acceptable Evidence 3.....	9
1.4 Percentage of engineering students whose learning style corresponds to each dimension. Data represent over 4,499 engineering students from over 10 countries	16
3.1 A sample slide that was presented to the test group as part of their lecture.	51
3.2 The Test Group's Learning Style distribution.	52
3.3 The Control Group's Learning Style distribution.	53
3.4 Sample RAPTOR Slide.....	56
3.5 Test Question 1, Flow Style.....	61
3.6 Slide 5F, presenting “Hello World!” using RAPTOR.	67
3.7 Slide 8, displaying Hello World in a command prompt.	68
3.8 RAPTOR's Start.	68
3.9 Slide 10F, RAPTOR’s PUT statement, PUT “Hello World!”	69
3.10 Slide 11F, RAPTOR’s End.	70
4.1 First Pilot Age Frequency Distribution.	74
4.2 First Pilot Gender Frequency Distribution.....	75
4.3 First Pilot Class Level Frequency Distribution.....	75
4.4 First Pilot GPA Range Frequency Distribution.	76

4.5	First Pilot Active or Reflective Frequency Distribution.	77
4.6	First Pilot Percent of Active or Reflective Novices.	77
4.7	First Pilot Weighted Percent Active or Reflective Scores.	78
4.8	First Pilot Sensing or Intuitive Frequency Distribution.	79
4.9	First Pilot Percent of Sensing or Intuitive Novices.	79
4.10	First Pilot Weighted Percent Sensing or Intuitive Scores.	80
4.11	First Pilot Visual or Verbal Frequency Distribution.	81
4.12	First Pilot Percent of Visual or Verbal Novices.	81
4.13	First Pilot Weighted Percent of Visual or Verbal Scores.	82
4.14	First Pilot Sequential or Global Frequency Distribution.	83
4.15	First Pilot Percent of Sequential or Global Students.	83
4.16	First Pilot Weighted Percent of Sequential or Global Scores.	84
4.17	Second Pilot Age Distribution.	85
4.18	Second Pilot Gender Distribution.	86
4.19	Second Pilot Class Level Distribution.	86
4.20	Second Pilot GPA Distribution.	87
4.21	Second Pilot Active or Reflective Frequency Distribution.	88
4.22	Second Pilot Percent of Active or Reflective Students.	88
4.23	Second Pilot Weighted Percent of Active or Reflective Scores.	89
4.24	Second Pilot Sensing or Intuitive Frequency Distribution.	90
4.25	Second Pilot Percent of Sensing or Intuitive Students.	90
4.26	Second Pilot Weighted Percent of Sensing or Intuitive Scores.	91
4.27	Second Pilot Visual or Verbal Frequency Distribution.	92

4.28	Second Pilot Percent of Visual or Verbal Students.	92
4.29	Second Pilot Weighted Percent of Visual or Verbal Scores.	93
4.30	Second Pilot Sequential or Global Frequency Distribution.	94
4.31	Second Pilot Percent of Global or Sequential Students.	94
4.32	Second Pilot Weighted Percent of Global or Sequential Scores.	95
4.33	The Experiment's Age Distribution.	97
4.34	The Experiment's Gender Distribution.	98
4.35	The Experiment's Class Level Distribution.	98
4.36	The Experiment's GPA Distribution.	99
4.37	The Experiment's Right and Left Handedness.	99
4.38	The Experiment's Test Time Distribution.	100
4.39	The Experiment's Active or Reflective Frequency Distribution.	101
4.40	The Experiment's Percent of Active or Reflective Students.	101
4.41	The Experiment's Weighted Percent of Active or Reflective Scores.	102
4.42	The Experiment's Sensing or Intuitive Frequency Distribution.	103
4.43	The Experiment's Percent of Sensing or Intuitive Students.	103
4.44	The Experiment's Weighted Percent of Sensing or Intuitive Scores.	104
4.45	The Experiment's Visual or Verbal Frequency Distribution.	105
4.46	The Experiment's Percent of Visual or Verbal Students.	105
4.47	The Experiment's Weighted Percent of Visual or Verbal Scores.	106
4.48	The Experiment's Sequential or Global Frequency Distribution.	107
4.49	The Experiment's Percent of Global or Sequential Students.	107
4.50	The Experiment's Weighted Percent of Global or Sequential Scores.	108

7.1	An example of what Flow and Text would look like together.	149
7.2	An example of inserting Flow control into our Text slide.....	150
B.1	Slide 5F, presenting “Hello World!” using RAPTOR.	162
B.2	Slide 8, displaying Hello World in a command prompt.	162
B.3	Slide 9F, RAPTOR’s Start.....	163
B.4	Slide 10F, RAPTOR’s PUT statement, PUT “Hello World!”	164
B.5	Slide 11F, RAPTOR’s End.	164
B.6	Slide 14F, program to calculate Age, hardcoded values.....	166
B.7	Slide 16F, BirthYear = 1981 highlighted.....	167
B.8	Slide 18F, calculating Age.....	167
B.9	Slide 19F, setting value of Age.....	168
B.10	Slide 20F, printing out Age.....	168
B.11	Slide 24F, introducing the concept of User Input.	170
B.12	Slide 25F, displaying a sample program that uses User Input.	171
B.13	Slide 30F, displaying the results of our program that demonstrates user input..	172
B.14	Slide 31F, PUT statement highlighted.	172
B.15	Slide 33F, displaying when the user was born.....	173
B.16	Slide 36F, displaying the alternative structure.....	174
B.17	Slide 37F, a sample program using Alternative.....	175
B.18	Slide 44F, highlighting the alternative structure in use.	176
B.19	Slide 45F, highlighting the PUT statement.....	177
B.20	Slide 48F, displaying the alternative structure.....	178
B.21	Slide 49F, displaying the a more advanced alternative structure in a program. .	179

B.22	Slide 61F, displaying the looping structure.	181
B.23	Slide 62F, displaying the looping structure in a program.	182
B.24	Slide 68F, highlighting the Loop shape.	183
B.25	Slide 69F, highlighting the conditional.	184
B.26	Slide 70F, highlighting the first shape in the Yes branch.	185
B.27	Slide 72F, highlighting the second shape in the Yes branch..	186
B.28	Slide 75F, highlighting the third shape in the Yes branch.	187
B.29	Slide 76F, highlighting the Loop shape again.	188
B.30	Slide 85F, highlighting the conditional again.	189
B.31	Slide 86F, highlighting the conditional.	191
B.32	Slide 87F, highlighting the No branch in the conditional.	192
C.1	Slide 1 – Flow.	195
C.2	Slide 1 - Text.	195
C.3	Slide 2 – Flow and Text.	196
C.4	Slide 3 – Flow and Text.	196
C.5	Slide 4 – Flow and Text.	197
C.6	Slide 5 – Flow.	198
C.7	Slide 5 - Text.	198
C.8	Slide 6 – Flow.	199
C.9	Slide 6 - Text.	199
C.10	Slide 7 – Flow.	200
C.11	Slide 7 - Text.	200
C.12	Slide 8 – Flow and Text.	201

C.13	Slide 9 – Flow.	201
C.14	Slide 9 - Text.....	201
C.15	Slide 10 - Text.....	202
C.16	Slide 10 – Flow.	204
C.17	Slide 11 - Text.....	204
C.18	Slide 11 – Flow.	205
C.19	Slide 12 - Text.....	205
C.20	Slide 12 – Flow.	206
C.21	Slide 13 - Text.....	206
C.22	Slide 13 – Flow.	207
C.23	Slide 14 - Text	207
C.24	Slide 14 – Flow.	208
C.25	Slide 15 - Text.....	208
C.26	Slide 15 – Flow.	209
C.27	Slide 16 - Text.....	209
C.28	Slide 17 - Text.....	210
C.29	Slide 16 – Flow.	211
C.30	Slide 18 - Text.....	211
C.31	Slide 17 – Flow.	212
C.32	Slide 19 - Text.....	212
C.33	Slide 18 – Flow.	213
C.34	Slide 20 - Text.....	213
C.35	Slide 19 – Flow.	214

C.36	Slide 21 - Text.....	214
C.37	Slide 20 – Flow.	215
C.38	Slide 22 - Text.....	215
C.39	Slide 21 – Flow and Slide 23 - Text.	216
C.40	Slide 22 – Flow.	217
C.41	Slide 24 - Text.....	217
C.42	Slide 23 – Flow.	218
C.43	Slide 25 - Text.....	218
C.44	Slide 24 – Flow.	219
C.45	Slide 26 - Text.....	219
C.46	Slide 24 – Flow.	220
C.47	Slide 27 - Text.....	220
C.48	Slide 25 – Flow.	221
C.49	Slide 28 - Text.....	221
C.50	Slide 29 - Text.....	222
C.51	Slide 27 – Flow.	223
C.52	Slide 30 - Text.....	223
C.53	Slide 28 – Flow.	224
C.54	Slide 31 - Text.....	224
C.55	Slide 29 – Flow and Slide 32 – Text.....	225
C.56	Slide 30 – Flow and Slide 33 – Text.....	225
C.57	Slide 31 – Flow.	226
C.58	Slide 34 - Text.....	227

C.59	Slide 32 – Flow.	227
C.60	Slide 35 - Text.....	228
C.61	Slide 33 – Flow and Slide 36 - Text.	228
C.62	Slide 34 – Flow.	229
C.63	Slide 37 - Text.....	229
C.64	Slide 35 – Flow.	230
C.65	Slide 38 - Text.....	230
C.66	Slide 36 – Flow.	231
C.67	Slide 39 - Text.....	231
C.68	Slide 37 – Flow.	232
C.69	Slide 40 - Text.....	232
C.70	Slide 38 – Flow.	233
C.71	Slide 41 - Text.....	233
C.72	Slide 42 - Text.....	234
C.73	Slide 39 – Flow.	235
C.74	Slide 43 - Text.....	235
C.75	Slide 40 – Flow.	236
C.76	Slide 44 - Text.....	236
C.77	Slide 41 – Flow and Slide 45 - Text.	237
C.78	Slide 42 – Flow and Slide 46 - Text.	237
C.79	Slide 43 – Flow.	238
C.80	Slide 47 - Text.....	238
C.81	Slide 44 – Flow.	239

C.82	Slide 48 - Text.....	239
C.83	Slide 49 - Text.....	240
C.84	Slide 45 – Flow.	241
C.85	Slide 50 - Text.....	241
C.86	Slide 46 – Flow and Slide 51 - Text.	242
C.87	Slide 52 - Text.....	242
C.88	Slide 47 – Flow.	243
C.89	Slide 53 - Text.....	243
C.90	Slide 48 – Flow.	244
C.91	Slide 54 - Text.....	244
C.92	Slide 49 – Flow.	245
C.93	Slide 55 - Text.....	245
C.94	Slide 50 – Flow.	246
C.95	Slide 56 - Text.....	246
C.96	Slide 57 - Text.....	247
C.97	Slide 51 – Flow.	248
C.98	Slide 58 - Text.....	248
C.99	Slide 52 – Flow.	249
C.100	Slide 59 - Text.....	249
C.101	Slide 53 – Flow and Slide 60 – Text.....	250
C.102	Slide 54 – Flow and Slide 61 – Text.....	250
C.103	Slide 55 – Flow.	251
C.104	Slide 62 - Text.....	251

C.105	Slide 56 – Flow.	252
C.106	Slide 63 - Text.....	252
C.107	Slide 64 - Text.....	253
C.108	Slide 65 - Text.....	253
C.109	Slide 57 – Flow.	254
C.110	Slide 66 - Text	254
C.111	Slide 58 – Flow and Slide 67 - Text.	255
C.112	Slide 68 - Text.....	255
C.113	Slide 59 – Flow.	256
C.114	Slide 69 - Text.....	256
C.115	Slide 60 – Flow.	257
C.116	Slide 70 - Text.....	257
C.117	Slide 61 – Flow.	258
C.118	Slide 71 - Text.....	258
C.119	Slide 62 – Flow.	259
C.120	Slide 72 - Text.....	259
C.121	Slide 63 – Flow.	260
C.122	Slide 73 - Text.....	260
C.123	Slide 64 – Flow.	261
C.124	Slide 74 - Text.....	261
C.125	Slide 65 – Flow and Slide 75 – Text.....	262
C.126	Slide 66 – Flow and Slide 76 – Text.....	262
C.127	Slide 67 – Flow.	263

C.128	Slide 77 - Text.....	263
C.129	Slide 68 – Flow.	264
C.130	Slide 69 – Flow.	265
C.131	Slide 78 - Text.....	265
C.132	Slide 79 - Text.....	266
C.133	Slide 70 – Flow.	267
C.134	Slide 80 - Text.....	267
C.135	Slide 71 – Flow and Slide 81 – Text.....	268
C.136	Slide 72 – Flow.	269
C.137	Slide 82 - Text.....	269
C.138	Slide 73 – Flow and Slide 83 – Text.....	270
C.139	Slide 74 – Flow and Slide 84 – Text.....	270
C.140	Slide 75 – Flow.	271
C.141	Slide 85 - Text.....	271
C.142	Slide 76 – Flow.	272
C.143	Slide 86 - Text.....	272
C.144	Slide 77 – Flow.	273
C.145	Slide 87 - Text.....	273
C.146	Slide 88 - Text.....	274
C.147	Slide 78 – Flow.	275
C.148	Slide 89 - Text.....	275
C.149	Slide 79 – Flow and slide 90 – Text.	276
C.150	Slide 80 – Flow.	277

C.151	Slide 91 – Text.....	277
C.152	Slide 81 – Flow and Slide 92 – Text.....	278
C.153	Slide 82 – Flow and slide 93 – Text.	278
C.154	Slide 83 – Flow.....	279
C.155	Slide 94 - Text.....	279
C.156	Slide 84 – Flow.....	280
C.157	Slide 95 - Text.....	280
C.158	Slide 85 – Flow.....	281
C.159	Slide 96 - Text.....	281
C.160	Slide 86 – Flow.....	282
C.161	Slide 97 - Text.....	282
C.162	Slide 98 - Text.....	283
C.163	Slide 99 - Text.....	283
C.164	Slide 87 – Flow.....	284
C.165	Slide 100 - Text.....	284
C.166	Slide 88 – Flow and Slide 101 – Text.....	285
C.167	Slide 102 - Text.....	285
C.168	Slide 89 – Flow.....	286
C.169	Slide 103 - Text.....	286
D.1	Test Question 1 – Flow.....	288
D.2	Test Question 1 - Text.....	288
D.3	Test Question 2 – Flow.....	289
D.4	Test Question 2 - Text.....	289

D.5	Test Question 3 – Flow.....	290
D.6	Test Question 3 - Text.....	290
D.7	Test Question 4 - Flow.....	291
D.8	Test Question 4 - Text.....	291
D.9	Test Question 5 - Flow.....	292
D.10	Test Question 5 - Text.....	292
D.11	Test Questions 6 and 7 - Flow.	293
D.12	Test Questions 6 and 7 - Text.	293
D.13	Test Question 8 - Flow.....	294
D.14	Test Question 8 - Text.....	294
D.15	Test Questions 9 and 10 - Flow.	295
D.16	Test Questions 9 and 10 - Text.	295
E.1	Test Question 1 - Flow.....	297
E.2	Test Question 1 - Text.....	297
E.3	Test Question 2 - Flow.....	298
E.4	Test Question 2 - Text.....	298
E.5	Test Question 3 - Flow.....	299
E.6	Test Question 3 - Text.....	299
E.7	Test Questions 4, 5, and 6 - Flow.	300
E.8	Test Questions 4, 5, and 6 - Text.	300
E.9	Test Question 7 - Flow.....	301
E.10	Test Question 7 - Text.....	301
E.11	Test Questions 8 and 9 - Flow.	302

E.12	Test Questions 8 and 9 - Text.	302
E.13	Test Question 10 - Flow.....	303
E.14	Test Question 10 - Text.....	303
E.15	Test Question 11 - Flow.....	304
E.16	Test Question 11 - Text.....	304
E.17	Test Questions 12, 13, and 14 - Flow.	305
E.18	Test Questions 12, 13, and 14 - Text.	305
E.19	Test Question 15 - Flow.....	306
E.20	Test Question 15 - Text.....	306
E.21	Test Question 16 - Flow.....	307
E.22	Test Question 16 - Text.....	307
E.23	Test Question 17 - Flow.....	308
E.24	Test Question 17 - Text.....	308
E.25	Test Question 18 - Flow.....	309
E.26	Test Question 18 - Text.....	309
E.27	Test Question 19 - Flow.....	310
E.28	Test Question 19 - Text.....	310
E.29	Test Question 20 - Flow.....	311
E.30	Test Question 20 - Text.....	311

LIST OF TABLES

Table	Page
1.1 Results of “Which programming concepts have been difficult for you to learn?”. 4	
1.2 List of important and difficult introductory computer science concepts. 5	
1.3 Classification of the MBTI's four dichotomies. 11	
1.4 The Four Dimensions of the ILS. 14	
1.5 Pearson's Correlation of test-retest scores. 15	
1.6 Internal Consistency Reliability measured with Cronbach's alpha. 16	
1.7 The relationships between dynamic visualization obstacles and this work. 21	
2.1 From Schulte and Bennedsen [24], the six most important topics that teachers teach in an introductory programming course. 35	
2.2 A survey of distinguishing characteristics of flow-model tools. 35	
2.3 Grades of students involved in the using Alice experiment. 42	
2.4 Percent retention of students involved in the Alice experiment. 42	
2.5 RAPTOR and other tools Final Exam Comparison Statistics. 44	
3.1 The Test and Control Groups Learning Style Distribution. 52	
3.2 Final Programming Grade and Final Overall Grade results. 53	
3.3 Bloom's Taxonomy of Learning. 58	
3.4 Bloom's Taxonomy's application to Computer Science. 59	
3.5 Key words associated with Bloom’s Taxonomy. 59	
3.6 Further examples of Computer Science concepts applied to Bloom's Taxonomy 60	

3.7	Normalized scale of importance and difficulty.....	64
3.8	Slide 5 Flow and Text scripts.....	67
3.9	Slide 6 Flow and Text scripts.....	67
3.10	Slide 9 Flow and Text scripts.....	68
3.11	Slide 10 Flow and Slide 11 Text scripts.	69
3.12	Slide 11 Flow and Slide 12 Text scripts.	69
4.1	First Pilot Program and Final Score Statistics and Independent Samples Test. ...	84
4.2	Second Pilot Test Scores Statistics and Independent Samples Test.	95
4.3	The Second Pilot's Entire Test's Reliability Statistics.	96
4.4	The Second Pilot's Test's Reliability Statistics, Removed Questions 2 and 8.	96
4.5	The Experiment's Statistics and the Experiment's Independent Samples Test, grouped by Flow and Text.	109
4.6	The Experiment's Independent Samples Test, comparing the Computer Science concepts of Sequence, Alternative, and Iteration, grouped by Flow and Text.	110
4.7	The Experiment's Test's Reliability Statistics.....	110
5.1	Comparing General Demographics considering Teaching Style.....	113
5.2	The ANCOVA to determine the effects of Class Level and Flow or Text exposure on Test Grade.	113
5.3	A Sample Format for our Statistical Tests.....	115
5.4	Test Grade Correlations Involving Visual/Verbal scores considering Gender...	116
5.5	Test Grade Correlations Involving Sensing/Intuitive scores.	118
5.6	Test Grade Correlations Involving Sensing/Intuitive scores Considering Gender	118

5.7	Test Grade Correlations Involving Sensing/Intuitive scores considering Time of Day	109
5.8	Test Grade Correlations Involving General Measures.....	122
5.9	Test Grade Correlations Involving General Measures considering Gender.	123
5.10	Test Grade Correlations Involving General Measures considering Time of Day.....	124
5.11	Comparing General Demographics considering Gender.	124
5.12	Comparing General Demographics considering Time of Day.	124
5.13	Correlating Visual/Verbal scores with other ILS scores and other General demographics.	126
5.14	Correlating Visual/Verbal scores with other ILS scores and other General demographics.	127
5.15	Correlating Visual/Verbal scores with other ILS scores and other General demographics considering Time of Day.	128
5.16	Correlating Sequential/Global scores with other ILS scores and other General demographics.	129
5.17	Correlating Sequential/Global scores with other ILS scores and other General demographics considering Gender.	130
5.18	Correlating Sequential/Global scores with other ILS scores and other General demographics considering Time of Day.	131
5.19	Correlating Sensing/Intuitive scores with other ILS scores and other General demographics.	133

5.20	Correlating Sensing/Intuitive scores with other ILS scores and other General demographics considering Gender.	134
5.21	Correlating Sensing/Intuitive scores with other ILS scores and other General demographics considering Time of Day.	135
5.22	Correlating Active/Reflective scores with other ILS scores and other General demographics.	136
5.23	Correlating Active/Reflective scores with other ILS scores and other General demographics.	137
5.24	Correlating Active/Reflective scores with other ILS scores and other General demographics considering Time of Day.	138
5.25	Comparing General Demographics considering Gender.	139
5.26	Comparing General Demographics considering Time of Day.	139
B.1	Slide 5 Flow and Text scripts.....	161
B.2	Slide 6 Flow and Text scripts.....	162
B.3	Slide 9 Flow and Text scripts.....	163
B.4	Slide 10 Flow and Slide 11 Text scripts.	163
B.5	Slide 11 Flow and Slide 12 Text scripts.	164
B.6	Slide 13-14 Flow and Slide 14-15 Text scripts.....	165
B.7	Slide 15 Flow and Slide 16-17 Text scripts.....	166
B.8	Slide 22 Flow and Slide 24 Text scripts.	169
B.9	Slide 24 Flow and Slide 26 Text scripts.	169
B.10	Slide 25 Flow and Slide 27 Text scripts.	170
B.11	Slide 26 Flow and Slide 28-29 Text scripts.....	170

B.12	Slide 34 Flow and Slide 37 Text scripts.	173
B.13	Slide 36 Flow and Slide 39 Text scripts.	173
B.14	Slide 38 Flow and Slide 41-42 Text scripts.	175
B.15	Slide 43-44 Flow and Slide 47-49 Text scripts.....	176
B.16	Slide 47-48 Flow and Slide 53-54 Text scripts.....	177
B.17	Slide 50 Flow and Slide 56-57 Text scripts.	179
B.18	Slide 57 Flow and Slide 64-66 Text scripts.	180
B.19	Slide 59-60 Flow and Slide 69-70 Text scripts.....	180
B.20	Slide 61 Flow and Slide 71 Text scripts.	181
B.21	Slide 69 Flow and Slide 78-79 Text scripts.	183
B.22	Slide 77 Flow and Slide 87-88 Text scripts.	188
B.23	Slide 84-85 Flow and Slide 95-96 Text scripts.....	189
B.24	Slide 86 Flow and Slide 97-99 Text scripts.	190
B.25	Slide 89 Flow and Slide 102-103 Text scripts.	193

LIST OF ACRONYMS

ANCOVA – Analysis of Covariates

CG – Conceptual Graph

COTS – commercial off the shelf

IDE – Integrated Development Environment

ILS – Index of Learning Styles

ITiCSE – Innovation and Technology in Computer Science Education

MBTI – Myers-Briggs Type Indicator

NS – Not Significant/No Significance

RAPTOR – Rapid Application ProtoTyping of Objects and their Relations

SARS – Severe acute respiratory syndrome

VP – Visual Programming

CHAPTER I

INTRODUCTION

The field of computer science is losing valuable minds to the field *just* because novices cannot program, get frustrated, and do not pursue the science further, and this fact can account for a portion of the reason that computer science has a 30-50% attrition rate [1]. Two studies [2] performed at the University of Strathclyde in the United Kingdom found that students could not program, trace programs, nor design programs at acceptable levels at the conclusion of their introductory computer science course. The reason suggested is because students often held non-viable mental models of key programming concepts which may cause misconceptions and difficulties in solving programming problems. Felder laments that “Unfortunately, a single approach has dominated engineering education since its inception: the professor lectures and the students attempt to absorb the lecture content and reproduce it in examinations. That particular size fits almost nobody: it violates virtually every principle of effective instruction established by modern cognitive science and educational psychology [3-6]” [7, p. 57]. This approach finds its roots in Germany in the 1440s. With the arrival of the printing press, all data flows had to “pass through the defile of the signifier”[8, p. 104], and here begins the constraining text-based understanding system that we still use today.

It is interesting that we still use lectures, when language is inherently one-dimensional and sequential [9], and is limited to one-dimensional understanding since sound waves arrive at the ear of the listener over time. The cognitive processing which interprets audio sensory information is such that even if multiple sources of sound are picked up by the ear, we are generally unable to comprehend multiple threads of conversation simultaneously. Written language suffers the same limitation, since the symbols of a written language, usually associated with sounds or sequences of sound, form strings of symbols which have meaning and which can easily be converted back into meaningful audio sequences (read aloud) [9].

Crapo, in contrast, describes the benefits of visual understanding [9]. Vision is not one-dimensional, since the retina of the eye is a two-dimensional surface with topographical mapping to visual portions of the brain [10-12]. The results are that we usually interpret the stimulus reaching our eyes in a three dimensional manner.

Furthermore, visual features such as motion, color, intensity, size, intersection, closure, orientation, lighting direction, and distance from the observer may be extracted “preattentively,” without conscious effort and within 100-200 milliseconds. These features “pop out” of the visual scene [13], which Card et al. [14] refer to as “automatic” processing. Parallelism also exists in higher levels of the brain where, for example, processing of spatial information appears to occur separately from processing of texture or motion [11].

In this work we develop a method that encourages communication of programming solutions by primarily appealing to programming novices who prefer to understand visually (per the Index of Learning Styles (ILS) Questionnaire [15]), an

understanding method not currently accommodated through the standard lecture style used in most classes [16]. We use an existing tool to test this method, and share the program understanding results using this theory.

One way to define *programming* is the process of transforming a mental plan of desired actions for a computer into a representation that can be understood by the computer [17]. With that in mind, meaningful understanding is strived for, in which the learner connects new material with knowledge that already exists in memory [18]. The existing knowledge in memory has been called a *schema*, and the process of connecting new information to it has been called *assimilation* [19].

1.1 Who is a Novice?

There are several stages from which a novice programmer turns into an expert programmer [20]. The most commonly cited are the five stages proposed by [21]: novice, advanced beginner, competence, proficiency, and expert.

A survey of the literature found that common mistakes that novices make, and difficulties that instructors feel that novices have include

- “students find it hard to create a general picture of the execution of a program that solves a problem” [22, p. 214]
- “there are often misconceptions related to variable initialization, loops, conditions, pointers and recursion. Students also have problems with understanding that each instruction is executed in the state that has been created by the previous instructions” [23, p. 15].

Lahtinen et al. [23] surveyed 559 students and 34 teachers (from six partner universities, including the survey's authors' institution, Tampere University of Technology in Tampere, Finland) which included the question, "Which programming concepts have been difficult for you to learn?" on a scale from 1 to 5, where 5 means very important or very difficult, and 1 means no important or not difficult. These results are tabulated in Table 1.1. Note that *n* indicates how many people were in the sample and *Std Dev* indicates the standard deviation.

Table 1.1 Results of "Which programming concepts have been difficult for you to learn?"

Question	Students			Teachers		
	n	Average	Std Dev	n	Average	Std Dev
Variables (lifetime, scope)	541	2.10	.97	34	2.41	.70
Selection structures	552	1.98	.90	34	2.38	.70
Loop structures	551	2.09	.97	34	2.79	.91
Recursion	512	3.22	1.03	31	4.06	.96
Arrays	526	2.79	1.15	33	3.24	.71
Pointers, references	518	3.59	1.04	32	4.44	.56
Parameters	513	2.60	1.09	32	3.47	.76
Structured data types	496	2.90	1.03	31	3.45	.81
Abstract data types	499	3.02	1.10	31	4.06	.81
Input/Output handling	519	2.96	1.04	32	3.75	.88
Error handling	481	3.33	1.01	32	4.13	.79
Using Language Libraries	465	3.04	1.09	32	3.88	.71

Schulte and Bennedsen [24] asked over 700 computer science instructors (drawn from teachers who have attended conferences and/or workshops in computer science education as well as mailing lists) to rank 28 introductory computer science concepts in order of importance and in order of difficulty on a scale of 1 to 5, where 5 means very important or very difficult, and 1 means not important or not difficult. 191 responded to the on-line questionnaire completely, and the results were tabulated.

Table 1.2 List of important and difficult introductory computer science concepts.

Importance Scale	Computer Science Concept	Difficulty Scale	Computer Science Concept
4.7	Selection and Iteration	3.8	Recursion
4.6	Simple Data Structures	3.7	Algorithm Efficiency
4.3	Parameters	3.7	Polymorphism and Inheritance
4.2	Scope	3.7	Generics
4.2	Objects and Classes	3.7	Advanced Data Structures
4.1	Syntax	3.6	Pointers and References
4.0	Design Single Class	3.5	Design Multiple Classes
3.9	Instance and other types of variables	3.4	Divide and conquer (decomposition of a problem), stepwise refinement or other problem solving strategies
3.9	Integrated Development Environment (IDE)	3.4	Algorithm Design
3.9	Method Design	3.4	CRC-cards
3.8	Debugging	3.3	Static vs. Non-Static
3.8	Algorithm Design	3.2	Debugging
3.7	Object Communication	3.1	Object Communications
3.7	Encapsulation	3.0	Method Design
3.6	Divide and conquer (decomposition of a problem), stepwise refinement or other problem solving strategies	2.8	Scope
3.6	Design Classes	2.8	Parameters
3.4	Mental Model	2.8	Encapsulation
3.4	Libraries	2.8	Design Single Class
3.2	Pointer and References	2.7	Object and Class
3.2	Static vs. Non-Static	2.7	Instance and other types of variables
3.1	Polymorphism and Inheritance	2.6	Libraries
3.0	UML Class Diagrams	2.6	Mental Model
2.8	Recursion	2.6	Simple Data Structures
2.7	Ethics	2.5	Syntax
2.5	CRC-cards	2.4	Selection and Iteration
2.4	Advanced Data Structures	2.3	UML Class Diagrams
2.2	Algorithm Efficiency	2.3	IDE
2.1	Generics	2.2	Ethics

1.2 Types of Novices

Learning styles refer to the idea that different people learn information in different ways. A determination of sufficient evidence for the learning-styles concept was sought in [25], and we will introduce some of their findings here (the authors are cognitive psychologists with an interest in both basic science of learning and memory and in the way that this science can be harnessed to be more helpful to teachers and students and were commissioned to assess the scientific evidence underlying practical application of learning style assessment in school contexts).

On the academic side, at least 71 [26] different learning style models have been developed, and several thousand articles and dozens of books have been written about learning styles. There is also a commercial side to learning styles, consisting around publishing and selling measurement devices to help teachers assess individual learning styles, and many organizations at all levels of education recommend testing. The *meshing hypothesis* is the claim that presentation should mesh with the learner's preferred method of learning, while Richard Felder notes that

If professors teach exclusively in a manner that favors their students' less preferred learning style modes, the students' discomfort level may be great enough to interfere with their learning. On the other hand, if professors teach exclusively in their students' preferred modes, the students may not develop the mental dexterity they need to reach their potential for achievement in school and as professionals.
[27, p. 18]

A supporting view of the popularity of learning styles is that it is a product of its success in fostering learning and instruction. A more critical view of learning styles' popularity is that "people are concerned that they, and their children, be seen and treated

by educators as unique individuals” [25, p. 107]. A second critical view of learning styles’ popularity is that “if a person or a person’s child is not succeeding or excelling in school, it may be more comfortable for the person to think that the educational system, not the person or the child himself or herself, is responsible” [28, p. 108].

Acceptable evidence of Learning Styles effectiveness is described best by Figure 1.1, Figure 1.2, and Figure 1.3 and is reproduced from [25, p. 110]. These figures will be used to graphically represent the results of validating if learning style preferences match their perceived abilities.

In Figure 1.1:

1. A Style Learner using Method 1 performs better than B Style Learner using Method 1 and A Style Learner using Method 1 performs better than A Style Learner using Method 2.
2. B Style Learner using Method 2 performs better than A Style Learner using Method 2 and B Style Learner using Method 2 performs better than A Style Learner using Method 1.

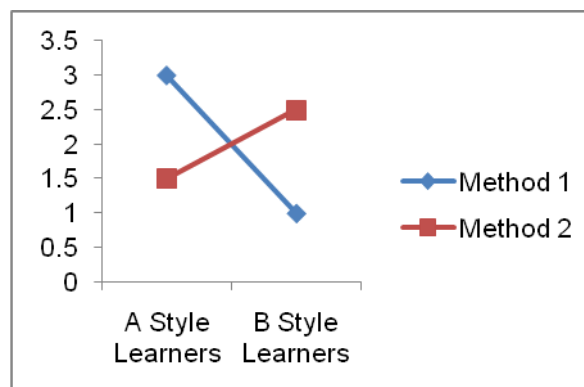


Figure 1.1 Acceptable Evidence 1.

In Figure 1.2:

1. A Style Learner using Method 1 performs worse than B Style Learner using Method 1 and A Style Learner using Method 1 performs better than A Style Learner using Method 2.
2. B Style Learner using Method 2 performs better than A Style Learner using Method 2 and B Style Learner using Method 2 performs better than A Style Learner using Method 1.

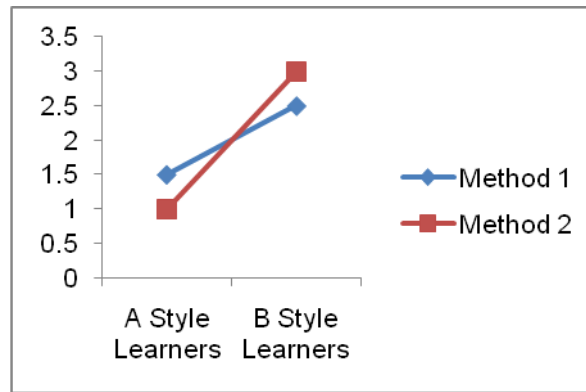


Figure 1.2 Acceptable Evidence 2.

In Figure 1.3

1. A Style Learner using Method 1 performs the same as B Style Learner using Method 1 and A Style Learner using Method 1 performs better than A Style Learner using Method 2.
2. B Style Learner using Method 2 performs better than A Style Learner using Method 2 and B Style Learner using Method 2 performs better than A Style Learner using Method 1.

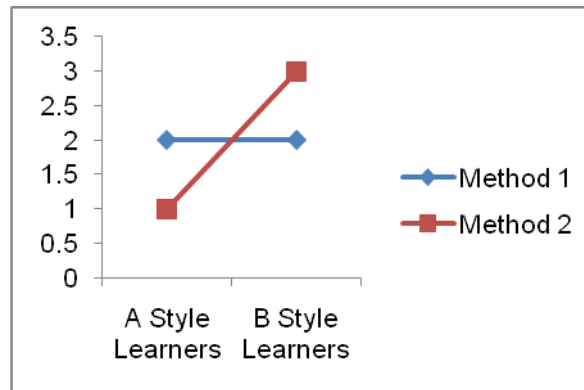


Figure 1.3 Acceptable Evidence 3.

At least five learning styles have been the subject of studies in the engineering education literature. The oldest and perhaps best known of these models is Jung's Theory of Psychological Type [29] as operationalized by the Myers-Briggs Type Indicator (MBTI) [30]. Other models that have been applied extensively to engineering are those of the Kolb Cycle by Kolb [31] and the Index of Learning Styles (ILS) by Felder and Silverman [32].

1.2.1 The Kolb Cycle

The Kolb Cycle addresses that there are four types of understanding [16]. The *why* student wants to know why the current subject is important. The *what* student wants to know the facts about the subject. The *how* student wants to know how to apply the subject to real problems. Finally, the *what if* student wants to experiment with different possibilities. This theory of understanding states that if an instructor *teaches around the Circle* by starting in the *why* quadrant and ending in the *what if* quadrant, then the needs of all the learners are met [33]. We take from the Kolb Cycle the idea of *teaching around the circle*, by addressing each of the types of understanding to ease understanding that

corresponds to an individual learner and to strengthen those types of understanding that are not as well developed within that individual learner.

1.2.2 The Myers-Brigg Type Indicator (MBTI)

The MBTI measures people's four dichotomies, and their relationships are described in Table 1.3.

Table 1.3 Classification of the MBTI's four dichotomies.

Attitudes		(E)xtravert – action oriented, seek breadth of knowledge and influence, prefer more frequent interaction, recharge and get their energy from spending time with people		(I)ntrovert – thought oriented, seek depth of knowledge and influence, prefer more substantial interaction, recharge and get their energy from spending time alone	
Functions	perceiving	(S)ensing – more likely to trust information that is in the present, tangible and concrete; information that can be understood by the five senses, distrust hunches, prefer to look for details and facts		I(n)tuition – trust information that is more abstract or theoretical, that can be associated with other information (either remembered or discovered by seeking a wider context or pattern), tend to trust hunches	
	judging	(T)hinking – tend to decide things from a more detached standpoint, measuring the decision by what seems reasonable, logical, causal, consistent and matching a given set of rules		(F)eeling – tend to come to decisions by associating or empathizing with the situation, looking at it ‘from the inside’ and weighing the situation to achieve, on balance, the greatest harmony, consensus and fit, considering the needs of the people involved	
Lifestyle		(J)udgment – likes to have matters settled		(P)erception – likes to keep decisions open	
		If the judgment is thinking , the person tends to appear to the world as logical	If the judgment is feeling , the person tends to appear empathetic	If the perception is sensing , then the person tends to appear to the world as concrete	If the perception is intuitive , then the person tends to appear to the world as abstract

These traits are similar to left or right handedness: individuals are either born with, or develop, certain preferred ways of thinking and acting. These four dimensions can be arranged in 16 different unique permutations, such as INFP (introversion, intuition,

feeling, and perception) who would typically be “idealistic, loyal to their values and to people who are important to them” [28].

1.2.3 Lateralization of Brain Function

In the human and some non-human brains, a longitudinal fissure separates the brain into two distinct cerebral hemispheres, connected by the corpus callosum. In research that began by testing patients who had their corpus callosum severed, researchers learned about the inter-hemispheric transfer of perceptual, sensory, motor, and other forms of information [34]. Research performed in [35] concluded that some humans’ brains’ functions are relegated to a particular hemisphere, while some functions are more bilaterally controlled. The right hemisphere generally performs functions such as the processing of visual and audiological stimuli and the left hemisphere generally performs functions such as grammar and vocabulary. Brain function lateralization is evident in the phenomena of right or left handedness, but a person’s preferred hand is not a clear indication of the location of brain function. Although 95% of right-handed people have left-hemisphere dominance for language, only 18.8% of left-handed people have right-hemisphere dominance for language function. Additionally, 19.8% of the left-handed have bilateral language functions [36].

1.2.4 The Index of Learning Styles

Even though the concept of learning styles is not universally accepted [7], we have chosen to use Felder’s Index of Learning Styles (ILS) to incorporate into our

research. This was based on the analysis on reliability and validity done by Olds et al. [37]. Such factors include

- High Test-retest reliability – the extent to which test results for an individual are stable over time
- High internal consistency reliability- the homogeneity of items intended to measure the same quantity-that is, the extent to which responses to the items are correlated
- Scale orthogonality-the extent to which the different scales of the instrument (if there are two or more scales) are independent
- Construct validity- the extent to which an instrument actually measures the attribute it purports to measure. The instrument scores are said to have convergent validity if they correlate with quantities with which they should correlate, and divergent or discriminant validity if they fail to correlate with quantities with which there is no reason to expect correlation.

We will capitalize on the Kolb Cycle's concept of *teaching around the circle* by satisfying both sides of each ILS dimension, which would reach the greatest number of novices. A further discussion of the ILS [38] [32] is in order. Students have different strengths and preferences in the ways they take in and process information: different learning styles that are continuous, not discrete. The ILS classifies students as having preferences for one category or the other in each of the following four dimensions (this project will focus on one dimension, Visual/Verbal, and discuss two other dimensions,

Active/Reflective and Sequential/Global, but will not discuss Sensing/Intuitive due to space constraints):

Table 1.4 The Four Dimensions of the ILS.

Sensing - concrete thinker, practical, oriented toward facts and procedures	Intuitive - abstract thinker, innovative, oriented toward theories and underlying meanings
Active – learn by trying things out, enjoy working in groups	Reflective – learn by thinking things through, prefer working alone or with a single familiar partner
Visual – prefer visual representations of presented material, such as pictures, diagrams and flow charts	Verbal – prefer written and spoken explanations
Sequential – linear thinking process, learn in small incremental steps	Global – holistic thinking process, learn in large leaps

We chose the Index of Learning Styles for its satisfactory test-retest reliability, satisfactory internal consistency reliability and satisfactory construct validity.

Test-retest reliability should be measured on a time interval that is long enough so that subjects cannot remember their responses from one examination to the next, but not so large that the item being measured might evolve significantly. Four weeks is suggested to be ideal for this purpose [39]. The following table shows the results of three studies using Pearson's Correlation of test-retest scores.

Table 1.5 Pearson's Correlation of test-retest scores.

Time	Active/ Reflective	Sensing/ Intuitive	Visual/ Verbal	Sequential/ Global	n	Source
4 weeks	.804	.787	.870	.725	46	[40]
7 months	.73	.75	.68	.60	24	[41]
8 months	.683	.678	.511	.505	124	[39]

Internal consistency reliability is measured using Cronbach's alpha, a statistic commonly used to measure the internal consistency or reliability of a psychometric test score for a sample of examinees, and we will use that coefficient to measure internal consistency reliability. When these statistics were first calculated in [42], the authors concluded that the ILS scales had low internal reliability, with Cronbach's alpha between 0.41 and 0.65 and were concerned of the ILS's robustness and construct validity. The authors of [43] suggests that there is a difference between instruments that measure a univariate quantity, such as a test of knowledge of a subject area, and instruments that measure preferences or attitudes, such as the ILS. In tests of a univariate quantity, a high level of proficiency in the subject area assessed should lead to correct answers to most items and a low level of proficiency in the subject area assessed should lead to mostly incorrect responses, therefore a high Cronbach alpha would be expected. However, with an instrument that measures learning styles, a very high correlation would suggest that the items are not assessing independent aspects of the preference, but are simply reworded variants of the same question. Considering this, Tuckman suggests [43] that a Cronbach alpha of 0.50 is acceptable for attitude and preference assessment, and the overall Cronbach alpha values for all four scales of the ILS meet this.

Table 1.6 Internal Consistency Reliability measured with Cronbach's alpha.

Active/ Reflective	Sensing/ Intuitive	Visual/ Verbal	Sequential/ Global	n	Source
.60	.77	.74	.56	572	[44]
.56	.72	.60	.54	242	[41]
.62	.76	.69	.55	584	[45]
.51	.65	.56	.41	284	[42]
.60	.70	.63	.53	557	[39]

Research [45] has reported results from over 4,499 engineering student questionnaires from 10 different English-speaking universities were collected between 1995 and 2002, and the following results emerged.

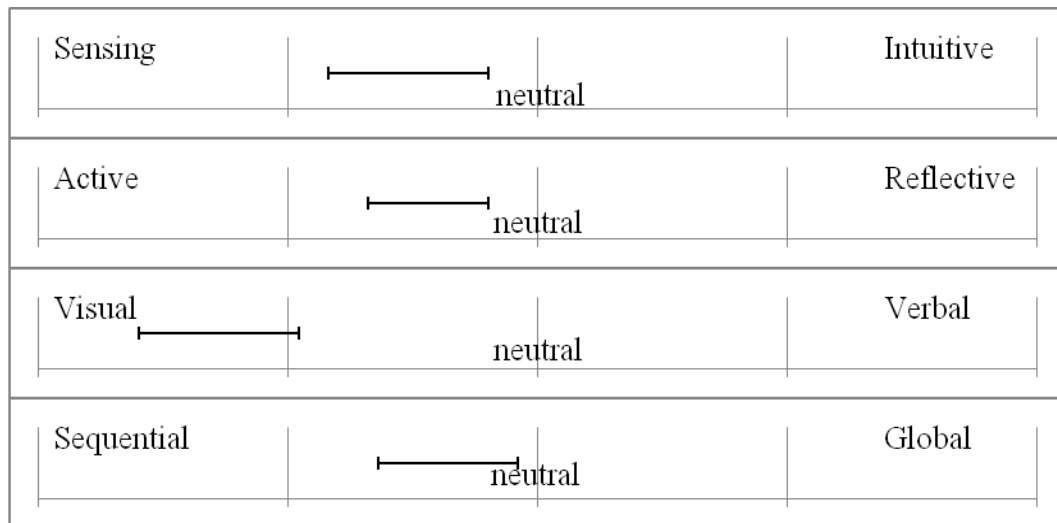


Figure 1.4 Percentage of engineering students whose learning style corresponds to each dimension. Data represent over 4,499 engineering students from over 10 countries.

According to learning style theory, conventional instruction in engineering courses favors reflective learners (since students in traditional lecture course are largely passive), intuitive learners (since the emphasis in most engineering courses is on theory and mathematical models), verbal learners (since most lectures and textbooks are

predominantly verbal), and sequential learners (since most courses and textbooks follow fairly rigid sequences in their presentation of information and little is generally done to provide ‘big picture’ contextualization of course material) [32, 46]. We will now suggest a method of understanding for novices that accommodates the visual/verbal, sequential/global and active/reflective learning styles.

1.3 The Most Underserved Novices: Addressing the Visual/Verbal Dimension

Our current method of understanding already addresses the verbal dimension of understanding, but engineering students predominately prefer to understand visually [45]. We want to preserve the existing method of teaching, while extending it to reach visual learners.

Visual learners learn through charts, diagrams, rich multimedia and simulations. Verbal learners, on the other hand, prefer lectures. Since verbal learners are easily accommodated through the standard lecture style used in most classes, and visual learners are harder to reach [16], we believe that utilizing Visual Programming (VP) is a way to appeal to the currently underserved visual learners.

We expect that students who are more visual learners will learn better using visual programming than students who are more verbal learners [47]. However, we expect all understanding to improve by handling both types of learning styles.

1.3.1 Visual Programming (VP)

The impetus for visualization in computing comes from the inherent abstractness of the basic building blocks of the field [of computer science]. Intuition suggests that, by

making these building blocks more concrete, graphical representations will help one to better understand how they work [48], by shaping how programmers visualize the operation of a program [49] and representing an existing mental model better than text can [50]. Vision is by far the predominant mode by which humans perceive the world in which we live [10, 51].

Many publications claim that VP is naturally superior to other forms of programming. When a 29-person working group on Improving the Educational Impact of Algorithm Visualization was asked by Naps et al. [48], *Using visualization¹ can help students learn computer concepts*, all respondents either strongly agreed (59%) or agreed (41%). In the same survey, when 67 Innovation and Technology in Computer Science Education (ITiCSE) 2002 conference attendees were asked the same question and 92% of the respondents either strongly agreed (43%) or agreed (49%), while the remainder were neutral or had no opinion. Many open responses were included, and we chose the following that were the most applicable to this project:

- Dynamic visualization is useful in explaining dynamic behavior or state changes.
- Dynamic visualization helps improve comprehension of both theoretical and practical issues because it makes the ideas more tangible.
- Dynamic visualization gives students immediate feedback and also helps students understand patterns that abound in the structures.
- One respondent has observed in twenty years of teaching that everyone learns better through visualization.¹ This respondent believes that visualization and kinetic experience better instill learning in long-term memory than do other

¹ In this section, visualization is defined as technology that can be used to graphically illustrate various concepts in computer science.

sensory and cognitive processes and concluded with the observation that learners better manipulate symbols when they have visual representations of process.

- Visualizations to convey more difficult or visual oriented material, explaining that dynamic visualizations express such ideas more clearly and succinctly than words.
- Another respondent observed that when students exercise on a conceptual level how algorithms work, they understand the algorithms better. This respondent went on to say that visual representation of the data structure supports this conceptual understanding, because implementation details no longer blur the students' view.
- Another respondent has used visualizations to help students achieve a deeper understanding of class concepts. This respondent has found that dynamic visualizations reach a broader cross-section of students than do some other pedagogical techniques.

In the same survey [48], the respondents were asked to indicate benefits that they had experienced from using visualization. The answers are as follows:

- 90% said the teaching experience is more enjoyable
- 86% saw improved level of student participation
- 83% gave anecdotal evidence that the class was more fun for students
- 76% had anecdotal evidence of improved student motivation
- 76% said visualization provides a powerful basis for discussing conceptual foundations of algorithms

- 76% said visualization allows meaningful use of the available technology
- 72% had anecdotal evidence of improved student learning
- 62% said (mis)understandings become apparent when using visualizations
- 52% had objective evidence of improved student learning
- 48% cited interaction with colleagues as a benefit.

While these responses indicate benefits of using visualization, we realize that anecdotal evidence is merely suggestive and that this survey was not designed to address it.

1.3.2 Limits of Visual Programming

When the 2002 ITiCSE attendees were asked [48] about factors that make the respondent or the respondent's colleagues reluctant or unable to use dynamic visualizations, many reasons emerged. The most relevant results follow, and we add how this project will attempt to address those results.

Table 1.7 The relationships between dynamic visualization obstacles and this work.

Why the respondent is reluctant to use dynamic visualizations.	How this project will address their reluctance.
93% said that time required to search for good examples	We will analyze what is fundamental to understanding computer science, and then provide helpful visual examples that represent this.
90% said time it takes to develop visualizations	
79% said time it takes to adapt visualizations to teaching approach and/or course content	
69% said time it takes to transition them into the classroom	
66% said unsure of how to integrate the technology successfully into a course	
90% said time it takes to learn the new tools	We will analyze the tools that are available, and find those which are both user friendly and powerful.
83% said lack of effective development tools	If the research supports that visualization is helpful for understanding programming, then the market of development tools will naturally grow, increasing the quality of available tools.
69% said lack of effective and reliable software	
59% said lack of evidence of effectiveness	We will test the effectiveness ourselves by collecting data and performing a statistical test on the results.
48% said unsure of how algorithm visualization technology will benefit students	
31% said AV may hide important details and concepts.	

We note specifically the concerns of lack of effective development tools and the lack of effective and reliable software. There are two reasons that there is a lack of effective development tools and a lack of effective and reliable software. Either we are wrong, or there hasn't been enough research done in this area. We believe that as research supports learning with visual programming tools, more developers will be drawn to develop more visual programming tools, and the market of visual programming tools will thrive.

Visual programming languages are notoriously difficult to scale to an enterprise level. At this time, a visual programming language should not seek to replace text languages, such as C, but instead strive to bridge the semantic gap between the novice's mental model and a textual program [52]. Although outside the scope of this dissertation,

Code Bubbles [53] are a new way to view existing code outside the traditional file paradigm. More limitedly, Cells in Forms [54] could display the equations *behind* the front cell, and the front cell would only display the result. Finally, as part of a more complete solution, the Map metaphor [55] should be explored. Each of these methods could address part of the problem with scalability.

An advanced organizer is useful; therefore we suggest using a diagram before introducing textual computing concepts. Work by Bransford and Johnson has been performed that suggests that a diagram, a component of Visual Programming, should be presented *before* technical text to increase meaningfulness, and that a diagram presented after technical text was no more useful than no diagram at all. Bransford and Johnson's experiment began [56], where a detailed technical passage was presented to subjects, then those were tested afterwards on their ability to recall details of the passage.

Three experiments were performed; one where the passage was presented without a title, one where the passage was presented with a title, and one where the passage was presented with a title only after the passage was read. The results of post-tests of those subjects who ranked the passage's comprehensibility and the subject's recall of the passage revealed that recall and comprehension was low if there was no title and if there was a title *after* the passage, and higher if there was a title before the passage. The work was followed by [56-58] which also found that students' recall of ambiguous and technical passages was enhanced when an organizing diagram was given prior to reading. We suggest that a plan of understanding must incorporate diagrams early.

Furthermore, experiments suggest that it is difficult to construct and maintain detailed images in memory [11, 59], and as the complexity of the image increases, our

ability to examine or manipulate the image to solve problems becomes increasingly inferior to our ability to use an external visualization to solve the same problem [60].

1.3.3 Empirical Evidence

An overarching theme of empirical evidence for and against visual programming is that there is little empirical evidence about visual programming, first broadly documented in [61]. A reason as to why this is includes the resistance of computer researchers to adopt techniques outside of the traditional set of computer science techniques.

1.4 Secondarily Addressing the Sequential/Global Dimension

In a survey of program understanding, [62] summarizes studies (in particular [63] and [20]) that explain some differences between experts and novices. They point out that experts generally understand concepts at higher levels than novices, and tend to conceptualize problems from the top down, which they can do because they have experience. Experts

have efficiently organized and specialized knowledge schemas; organize their knowledge according to functional characteristics such as the nature of the underlying algorithm (rather than superficial details such as language syntax); use both general program solving strategies (such as divide-and-conquer) and specialized strategies; use specialized schemas and a top-down, breadth-first approach to efficiently decompose and understand programs; and are flexible in their approach to program comprehension and their willingness to abandon questionable hypotheses. [64, p. 139]

By definition, novices do not have many of the strengths of experts. Continuing

Studies reviewed by [20], for example, have concluded that novices are limited to surface and superficially organized knowledge, lack detailed mental models, fail to apply relevant knowledge, and approach programming “line by line” rather than using meaningful program “chunks” or structures. [64, p. 140]

Program understanding should account for novices, who learn from the bottom up (sequentially), and should handle expert programming practices, where programs are understood from the top down (globally). This also can be addressed by using the Map metaphor [55].

1.5 Secondarily Addressing the Active/Reflective Dimension

Another dimension of the ILS is the active-reflective one. This classifies learners based on their preference for learning through first-hand experimentation and social interaction (i.e., action) as opposed to learning by thinking through the process and examining ideas mentally (i.e., reflection) [65].

Again, intuition and Lister suggest that novices “should first be taught to read programs before they write programs. As children in our early school years, the ability to read and understand our native language outstripped our capacity to write fragments of that language. Why should it not be the same when learning to program?” [66, p. 159]

However, in [20, p. 21], Winslow argues

Studies have shown that there is very little correspondence between the ability to write a program and the ability to read one. Both need to be taught along with some basic test and debugging strategies.

We argue that Lister’s case describes reflective learners, while Winslow’s case describes active learners, and that both learning styles should be accommodated.

Learners who are actively engaged with visualization technology have consistently outperformed learners who passively view visualizations [67], which is consistent with the Sociocultural Constructivist Learning Theory [68] that postulates that people learn better through active participation. There are six levels of learner engagement (which apply to all learners, regardless of learning style):

1. No viewing
2. Viewing
3. Responding
4. Changing
5. Constructing
6. Presenting

We aim to engage learners in all of these levels, as “more is better” [48, p. 148].

More specifically, creating solutions themselves allows the learners to talk about their creations more so than creating a solution that is taught to them from an instructor. We note that this is not specifically visual-oriented, but by enabling novices to create their solutions, they are engaged at level 5, and by discussing their creations with their peers; they are engaged at level 6. We will show in a further section how visualization technology can be the medium to actively engage learners at all of these levels.

1.6 Considering the Sensing/Intuitive Dimension

The final dimension of the ILS that we will discuss is sensing-intuitive. This classifies learners as concrete thinkers, practical, and oriented towards facts and procedures (sensing) or as abstract thinkers, innovative, oriented toward theories and underlying meanings. We believe that software programming may appeal to sensing learners because novices who have never programmed before may resonate with the facts and procedures associated with a software program. However, intuitive learners may find

a software program very abstract because they cannot actually see what software is doing; only what it represents on the screen.

1.7 Reducing Experimental Bias

In order to quantify “does a visual programming language increase a novice’s program comprehension,” at least one formal experiment must be set up. Many variables and biases, both from the student’s point of view and the instructor’s point of view, could exist in an experiment designed to test the effectiveness of visualization, and we will identify some of these now:

From the student’s point of view:

The driven student. The nature of volunteers for a study naturally appeals to a more driven person than someone who will not want to volunteer for a study.

Furthermore, this means that our population will probably be more ambitious than those who will rather not volunteer. This self-selection was present in our pilot study. We aim to address this by finding our volunteers from our universities’ Psychology Department, which requires that all students who enroll in introductory Psychology classes participate in experiments. Therefore, in our experiment (not our first pilot study), the students will be equally driven, non-Computer Science majors.

Student’s engagement: Not all students participate in class equally. A properly engaged student can ask questions and clarify when the student doesn’t understand, or when the instructor misspeaks or makes a mistake. We can limit

the instructor's chance of making mistakes by using the same, well-trained instructor.

Student's background: Students have many varied backgrounds. Students come from many different geographical areas, different nations, and different educations, and have different preferred learning styles, which a method of understanding should understand and accommodate. By collecting data concerning the student's backgrounds and preferred learning styles, we can measure the differences, but we may miss a key component, such as previous education in logic, or having a sibling who is a computer scientist.

From the instructor's point of view:

A biased instructor. As the authors support this method, we would constitute a biased instructor, who might spend more time answering questions of the test group and less time instructing the control group. By faithfully keeping to an instructional script and recording our experiments for review, we minimize this.

Different instructors. Different instructors have different teaching styles, and different levels of knowledge and experiences. By using the same instructor to teach both a control and test group, we will minimize the effects of having different instructors teach each group.

Too much overhead. The visualization technology may simply incur too much overhead to make it worthwhile [48]. By publishing our results and our methods, we reduce the overhead for other projects looking to share our technique.

1.8 Summary

We conclude that programming is difficult for many to learn, and a reason that it is so difficult could be because it is uncomfortable for novices to learn if they prefer to learn visually (which most engineering students do prefer), yet are taught verbally, which is a current method of teaching that dates back to the “alphabetic monopoly” forced on society with the printing press [8]. We believe that more novices could understand computer science concepts such as programming if they were engaged in a more visual way. We will begin our work by exploring the following hypothesis: *computer science novices understand computer science concepts (here sequence, iteration, alternative) better, and test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*, and our null hypothesis is *computer science novices will understand computer science concepts (here sequence, iteration, alternative) the same, and will not test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*.

CHAPTER II

RELATED WORK

In Section 2.1 we outline five different categories of technologies that support learning programming. Our research focused on methods and tools that fit into Sections 2.1.3 and 2.1.5. Section 2.1.3 is of particular importance because it introduces the flow-model that we believe will be best to use for our experiment. Section 2.1.5 is of importance because it describes a tiered-language category of visualization that grows with the knowledge and experience of the user. In Section 2.2 we discuss studies that have been published concerning technologies that support learning and we summarize the related work in Section 2.3.

2.1 Technologies that Support Learning Programming

Powers et al. [69] suggest that software developed to support learning programming can be divided into the following parts, presented as an outline to the reader, and discussed in more detail in Sections 2.1.1 through 2.1.5:

1. *narrative tools*

- a. The distinguishing characteristic of the program is to tell a story, often in 3 dimensions.

- b. This category provides motivation by attracting students to be interested in the programs they construct, which can lead students to spend time on the tasks.

This category's appeal is similar to the *specialized output realizations* appeal.

2. *visual programming tools*

- a. The distinguishing characteristic of the category is that the user interface enforces program structure, often by avoiding letting the program enter an illegal state (by utilizing drag-and-drop and other gestures), and therefore avoiding syntax errors.
- b. This category removes the distraction of learning programming language syntax and how to interpret compiler errors messages.

3. *flow-model tools*

- a. The distinguishing characteristic of the category is that the program is represented by a flowchart, and is a more specific type of *visual programming tool*.
- b. This category allows students to visualize how programs work and develop algorithms in a more intuitive fashion.

4. *specialized output realizations*

- a. The distinguishing characteristic of the category is that the program directly causes a concrete object to behave.
- b. This category provides motivation by attracting students to be interested in the programs they construct, which can lead students to spend time on the tasks.

This category's appeal is similar to the *narrative tool's* appeal.

5. *tiered language tools*

- a. The distinguishing characteristic of the category is that the program “grows” with what the subject understands.
- b. This category eases the student into learning complicated concepts by first beginning with simple concepts and incrementally adjusting the complexity.

We initially proposed that Conceptual Graphs (CGs), a knowledge modeling language [70], could support learning programming, particularly by being a flow-model visual programming language. Even though work has been done to show them execute [71-73], CGs have many abilities that we do not *need* to model program code, and will only work to overwhelm the novice. We established [74] that a CG can indeed completely model a programming language, but we found that there are other tools that have been developed that will work just as well as a newly created CG editor [75]. Current CG tools will require much rework to achieve the same functionality as current VP tools, and removing extra CG artifacts (to reduce a novice’s cognitive load) will remove much of what makes a CG a CG. We do like that the free form nature of CGs allows more of the screen to be utilized (horizontally instead of the predominately vertical nature of other visual languages such as 2.1.3), which addresses the scalability of visual programming languages, but in front of a novice that free form graph translates to a larger cognitive gap.

2.1.1 Narrative Tools

Cooper [69, p. 560] suggests that “the main appeal of storytelling is in attracting students to be interested in the programs they construct. Motivation is important, and it leads to students spending significant time on task.” An example of a widely used narrative tool is Alice [76].

Alice is a family of programs that appeals to specific subpopulations not normally exposed to computer programming, such as female students of middle school age, by encouraging storytelling as opposed to most other programming languages which are designed around computation [76]. Alice is a 3D programming environment where students create animations by placing objects in a 3D virtual world, and then program their behavior.

We claim that these tools are geared towards specific subpopulations, and fill the niche of convincing those specific subpopulations (such as female students of middle school age) to become programmers. Our method will be used to help the rest of the population that do not need convincing (although we admit that in our experiment we had to convince our participants), but instead need an abstract representation of the instructions necessary to program software that closely reflects an accurate mental representation of software in order to learn programming which is not currently being addressed, because computer science has a 30-50% attrition rate [1].

2.1.2 Visual Programming Tools

Since introductory students can be distracted when learning programming language syntax and how to interpret compiler error messages, visual programming tools

address this problem by providing a visual interface in which programs are constructed using gestures, such as drag and drop. Generally, these environments contain a user interface that enforces program structure, preventing syntax errors. With each gesture (or drag and drop), the user takes the program from one legal state to another. Furthermore, this approach elevates the unit of discourse from the character to the semantic unit by allowing users to directly manipulate graphical representations of programming concepts rather than type individual characters [69]. Furthermore, Ken Goldman [69] suggests that with the immediate feedback of JPie, he has seen students understand concepts better and the course can cover more concepts without losing valuable hands-on experience. Other examples of tools in this category are Alice, Karel Universe, and ALOHA [77].

ALOHA (Translated from Finnish: Practicing Algorithmic Programming) is a tool for learning structured programming and is based on understanding the program structures instead of the detailed syntax of the programming language. ALOHA also supports both visual and active learning styles [77]. A student is presented a set of blocks that will be used to make a program to match a given exercise. ALOHA supports the user dragging blocks from the set of available components and drops them into a separate window frame showing the current solution.

2.1.3 Flow-Model Tools

Flow-model tools, which are visual programs based on flowcharts, allow students to visualize how programs work and to develop algorithms in a more intuitive fashion. From [69, p. 561], Martin Carlisle “learned that, given a choice, over 95% of our students selected to use a flowchart to represent an algorithm, even when they had seen flowcharts

in only a single lecture and had been taught in 3rd generation programming language.”

Martin Carlisle believes that this is because the flow model greatly reduces syntactic complexity, allowing students to focus on solving the program instead of finding missing semicolons”[69, p. 561]. Examples of tools in this category includes RAPTOR [78], Iconic Programmer [79], and VisualLogic [80].

2.1.3.1 RAPTOR (Rapid Application ProtoTyping of Objects and their Relations)

RAPTOR is an iconic programming environment, designed specifically to help students visualize classes and methods and limit syntactic complexity[78]. It is a free, open-source tool with breakpoints that is written in A# and C# at the United States Air Force Academy.

2.1.3.2 Iconic Programmer

Iconic Programmer [79] allow students to create programs using flowcharts and supports input/output, selection, looping, and code generation, but does not support subprograms. Iconic Programmer does support multi-dimensional flowcharts (in an *if* statement, subsequent commands extend horizontally across the screen, as opposed to vertically on the screen), which help to use more available space on the screen.

2.1.3.3 VisualLogic

VisualLogic [80] is a commercial tool that supports creation of programs with multiple procedures, each of which is represented as a flowchart. The look and feel of VisualLogic is very similar to that of RAPTOR. The tool has a feature that allows the

tool to run for free in Demo Mode, where your content cannot be saved. VisualLogic supports breakpoints.

2.1.3.4 Summary of Flow-Model Tools

The following table summarizes the previously mentioned flow-model tools regarding topics that teachers find most important to include in an introductory programming course.

Table 2.1 From Schulte and Bennedsen [24], the six most important topics that teachers teach in an introductory programming course.

	RAPTOR	Iconic Programmer	Visual Logic
Selection and Iteration	yes	yes	yes
Simple Data Structures (arrays, strings)	yes	no	strings
Parameters	yes	no	yes
Scope	yes	yes	yes
Objects and Classes	yes	yes	no
Syntax	yes	yes	yes

The following table summarizes the previously mentioned flow-model tools regarding other distinguishing characteristics.

Table 2.2 A survey of distinguishing characteristics of flow-model tools.

	RAPTOR	Iconic Programmer	Visual Logic
Free	yes	yes	demo is free
Open Source	yes	no	no
Breakpoints	yes	no	yes
Zoom	yes	no	yes
Cut/Copy/Paste	yes	no	yes
Multidimensional	no	some multidimensions	no
Functions	yes	no	yes

RAPTOR is therefore the closest match of a visual programming language to our ideal method. However, this flow-model tool could be improved if it supported:

- Round-tripping. By making modifications to the text code and then have those imported back into the flow-model design, the novice will be able to make changes in either text or the flowchart, and will only need to know one in order to be able to see the other.
- Concurrent displays of the flowchart and a particular language's text, which will be necessary to teach syntax to novices.
- Clear expansion and contraction of user-defined functions. To begin, novices will need the details of a function, but to save space, and also to take out unnecessary complexities after they understand the function, they will want to simplify the visualization of the function.
- Multidimensional flowchart. If the flowchart were to be able to grow two dimensionally, then screen space will be more effectively utilized.

In electronic discussions with Martin Carlisle, the developer of RAPTOR at the U.S. Air Force Academy, he addressed our concerns. He stated, to each point [81]

- Round-tripping – Dr. Carlisle said implementing round-tripping was too time consuming, especially for the free-form nature of RAPTOR.
- Concurrent displays of the flowchart and text – Dr. Carlisle had not thought of that, but said it will require a textual representation of the RAPTOR programming language.

- Clear expansion and contraction of user-defined functions – Dr. Carlisle found that his students didn't use the existing expansion and contraction capabilities, so he has not furthered this area (besides supporting expanding and contracting selection and loops).
- Multidimensional flowchart – Dr. Carlisle agrees that scalability is a problem for visual programming languages, but he is not trying to address it with RAPTOR, and we only allude to its need for a visual programming scheme for intermediate and more advanced coding uses.

2.1.4 Specialized Output Realizations

Specialize Output Realizations are a category of tools where a user programs a robot to perform an activity or a series of activities. From [69], Myles McNally suggests that “for many educators the first appeal of using MindStorms is the motivation factor.” FLOGO I and FLOGO II are robotics programming languages developed with the goal of empowering children to construct more sophisticated behaviors for their robots, and to learn more by doing so [82]. Examples include Lego Mindstorms, JES, and FLOGO. As this category describes a more concrete approach to programming, it does not really apply to our general purpose program understanding, and therefore specialized output realizations will not be discussed further.

2.1.5 Tiered Language Tools

Tiered Language Tools start with the minimal syntax and complexity, with error messages designed to provide feedback appropriate for the students' understanding of the

language. As understanding progresses, new features are added and new error messages are modified to reflect students' current understanding of the language [69]. Viera Proulx suggests in [69, p. 561] that

Starting with a simple syntax of beginner level language students can focus on the design of the program and understand clearly the most salient points of program design. Additional language features are added when programming in the simple language becomes tedious and repetitious. Students recognize the need for a more complex feature – indeed in our experience they outright beg for it.

Examples include ProfessorJ [83] and RoboLab [84].

Utilizing a tiered approach like this supports chunking [85] and will help to lessen the student's feelings of being overwhelmed by too many new ideas, too soon [86]. By following a concept-first approach, and drawing on students' everyday experiences when introducing computer science concepts, we do not double the novice's cognitive load [87]. We will draw on students' everyday experiences:

Most learners already know something about any new topic they are asked to study, or they can make meaningful connections between what they know and what they are being asked to learn. However, possessing relevant prior knowledge is no guarantee that learners will activate and use it appropriately. In an instructional situation, then, the activation of prior knowledge should not be left to chance. To assure that meaningful learning takes place, instructors and designers can employ a variety of strategies to help learners relate their prior knowledge to new information they are to acquire. [88]

In future iterations of our work, we hope to implement our ideal method using a tiered language approach; however, we will not make this implementation change at this time.

2.2 Studies using Technologies that Support Learning

After researching the theories and tools that are available, we wanted to determine what effects they have on students. We discuss in the following sections more recent publications of the effects of using technologies that support learning, paying particular attention to RAPTOR in Section 2.2.3.

2.2.1 Using Multimedia Recordings

From 2001-2003 a suite of software visualization tools was developed for use for the CS1 course at the University of Minnesota, Duluth. Even though students consistently ranked the visualization software as more important to their learning than any other element of the course, these rankings were not highly correlated with actual outcomes [89]. This study of learning style determined that reflective and verbal learners outperformed active and visual ones. Student opinions of the value of programming projects and lectures rank highest and seem to cut across learning style preference. This study of learning style determined that reflective and verbal learners outperformed active and visual ones [90].

We argue that the multimedia recordings presented at the University of Minnesota, Duluth, are very good tutorials for how to use C++ in an IDE, but it doesn't visualize the abstract ideas of what happens behind the scenes of a textual programming language. A different method that appeals to active and visual novices is needed.

We want to incorporate from this study how problems are suggested for the novice to engage in learning what was just demonstrated.

2.2.2 Using Alice

In a study of the effectiveness of the Alice [91] spanning two universities, a curriculum, including a textbook, were designed for a new course that would immediately precede the traditional first computer science course or occur concurrently with the traditional first computer science course (dependent on the university). The reasons given that Alice was selected for [91] are

- Working with an easy-to-use 3D graphics environment is attractive and highly motivating to today's generation of media-conscious students
- The visual nature and immediate feedback of program visualizations makes it easy for students to see the impact of a statement or group of statements. Further, it makes debugging easier
- The drag-and-drop editor prevents students from making syntax errors that are prevalent for beginners
- The 3D modeled classes and instantiated objects in Alice provide a very concrete notion of the concept of an "object-first" approach [92-93].

Further reasons for the choice of Alice are [92] that the students develop

- A strong sense of design
- A contextualization for objects, classes, and object-oriented programming
- An appreciation of trial and error
- An incremental construction approach
- A firm sense of objects
- Good intuitions concerning encapsulation

- The concept of methods as a means of requesting an object to do something
- A strong sense of inheritance
- An ability to collaborate
- An understanding of Boolean types
- A sense of the program state
- An intuitive sense of behaviors and event-driven programming.

The authors compiled a list of fundamental concepts commonly taught in current first computer science course. These were decisions, repetition (definite and indefinite, as well as recursion), functions/methods, collections (typically arrays, though sometimes lists), objects (including state and behavior), inheritance, encapsulation, polymorphism and interactivity.

The students for this study were deemed at-risk computer science students, those who had minimal previous programming experience and perhaps a weak mathematical background (defined by using a calculus readiness exam). The groups were delineated:

- Treatment group –12 students – students at risk and who enrolled in Alice course
- Control Group 1 – 6 students – students at risk and who did not enroll in the Alice course
- Control Group 2 – 19 students – students not at risk or low risk and who did not enroll in the Alice course

Table 2.3 Grades of students involved in the using Alice experiment.

Group	2001-2002		2002-2003		2-Year Total	
Treatment	n	GPA	n	GPA	n	GPA
High Risk	7	2.86	12	3.05	19	2.98
Medium Risk	2	3.65	4	2.93	6	3.17
Total	9	3.04	16	3.02	25	3.03
Control Group 1	n	GPA	n	GPA	n	GPA
High Risk	10	1.32	2	.50	12	1.18
Medium Risk	14	2.36	4	2.75	18	2.45
Total	24	1.93	6	2.00	30	1.94
Control Group 2	n	GPA	n	GPA	n	GPA
Low Risk	19	2.28	3	3.33	22	2.43
No Risk	7	3.34	23	3.51	30	3.47
Total	26	2.57	26	3.49	53	3.03
Total of all students	n	GPA	n	GPA	n	GPA
	59	2.38	48	3.15	107	2.73

The results of the collected data suggest that students who were at risk and learned computer concepts using Alice before undertaking the full first computer science course scored considerably better than students who were at risk and did not use Alice before undertaking the full first computer science course (2.98 GPA to 1.18 GPA). Retention among these groups was also tracked, and the data is shown in Table 2.4.

Table 2.4 Percent retention of students involved in the Alice experiment.

Group	% retained 2001-2002	% retained 2002-2003	2-Year Total
Treatment			
High Risk	86 (6/7)	83 (10/12)	84 (16/19)
Medium Risk	100 (2/2)	100 (4/4)	100 (6/6)
Total	89 (8/9)	87 (14/16)	88 (22/25)
Control Group 1			
High Risk	10 (1/10)	50 (1/2)	15 (2/12)
Medium Risk	57 (8/14)	100 (4/4)	67 (12/18)
Total	37 (9/24)	83 (5/6)	47 (14/30)
Control Group 2			
Low Risk	63 (12/19)	100 (3/3)	68 (15/22)
No Risk	86 (6/7)	78 (18/23)	80 (24/30)
Total	69 (18/26)	81 (21/26)	75 (39/52)
Total of all students	59 (35/59)	83 (40/48)	70 (75/107)

The results of this data suggest that students who were at risk and learned computer science concepts using Alice before undertaking the full first computer science course were more likely to stay in computer science than the students who were at risk and did not use Alice before undertaking the full first computer science course (88% to 47%).

2.2.3 Using RAPTOR at the U.S. Air Force Academy

From [94], in an introductory IT course at the U.S. Air Force Academy, a customized version of RAPTOR was compared to the existing commercial off the shelf (COTS) software visualization package, Microsoft Visio. The experiment [38] focused on three questions from the final exam. The first question asked the students to get three numbers from the user and print the numbers starting with the first number through the second number but excluding the third number. The second question had a bowling theme testing loops and accumulators. The user would enter the score for a team of four bowlers playing three games. The program would validate each score, re-prompt on invalid scores, and then calculate a total score for the team. The third question dealt with selection and had a Severe Acute Respiratory Syndrome (SARS) theme. The program asked an airline passenger four health-related questions; one was their temperature and the other three were yes/no questions. If the answers to two or more of the questions indicated the possibility of SARS, the program would direct the passenger for further examination; otherwise it would release the passenger to board the aircraft. The results of the experiment showed that the class that used RAPTOR answered the same three questions (that were chosen out of the entire exam) in their Final Exam more accurately

than those classes that used Ada or MATLAB. This is even more remarkable given that the RAPTOR class had average GPAs at the start of this class lower than the average GPAs at the start of the other classes (2.47 compared to 2.84). Since the RAPTOR class only had 16 students in it, the sample size was too small to achieve statistical significance, even though the other classes had 365 students.

With these encouraging results, the department decided to replace the COTS tool with RAPTOR in all sections, beginning in Fall of 2003. The U.S. Air Force Academy continued collecting data measuring their classes' data [95]. Their data compared the class that learned with Ada or MATLAB in Spring 2003 (class size of 365) to the class the learned with RAPTOR in Spring 2004 (class size of 530). The results are summarized below in Table 2.5 (the Summer 2003 data is from the RAPTOR class, and the Spring 2003 data is from the classes that teach using Ada or MATLAB). A comparison was also made between the RAPTOR classes taught during Spring 2003 (class size 365) and Spring 2004 (class size 429).

Table 2.5 RAPTOR and other tools Final Exam Comparison Statistics.

		Question1	Question 2	Question 3
Spring 03 (other tools)	Average	72.00%	78.40%	88.60%
	Std Dev	23.40%	24.20%	18.10%
	n	365	365	365
Summer 03 (RAPTOR)	Average	77.50%	88.80%	94.20%
	Std Dev	22.30%	17.50%	9.70%
	n	16	16	16
Fall 03 (RAPTOR)	Average	76.30%	74.70%	92.60%
	n	530	530	530
Spring 04 (RAPTOR)	Average	76.0%	79.0%	92.7%
	n	429	429	429

The overall results of the U.S. Air Force Academy's experiment with RAPTOR showed statistically significant increases in performance on both the Question 1 and Question 3, but results "were less clear cut" [95, p. 178] for Question 2. The authors of data [95] suggest that the statistically significant decrease in Question 2 could be attributed to the fact that "arrays in RAPTOR are implicitly declared and hence less obvious to the students. In addition, the programming assignment for arrays during the Fall 2003 was far more challenging than the other semesters," so it could be said that too much was changed to really compare them.

2.3 Summary

We have introduced a broad categorization of technologies that support program understanding, and focus on one category in particular, flow-model tools, whose relevance to our project will be explored in the next section. We also discuss several recent experiments involving integrating visualization into current program understanding environments. Next, we will present our general approach to testing if a Visual Teaching Style is better for Visual Learners than a Verbal Teaching Style is for Visual Learners and if a Verbal Teaching Style is better for Verbal learners than a Visual Teaching Style is for Verbal Learners.

CHAPTER III

GENERAL APPROACH

Some variables concerning novices who want to understand computer science and variables concerning those experts who want to share their understanding of computer science we can measure, some we can control, and some we can do nothing about. We suggest the use of a visual programming language, designed with the following features in mind, to test the hypothesis that *computer science novices understand computer science concepts (here sequence, iteration, alternative) better, and test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. From [85, p. 189] “Pedagogical support, such as providing examples by analogy, will likely be an integral part of future software tools. Results from the empirical works also suggest that there is a need for tools to help programmers learn a new language.” More specifically, such a tool must provide many services, such as

1. Where is a particular subroutine/procedure invoked?
2. What are the arguments and results of a function?
3. How does control flow reach a particular location?
4. Where is a particular variable set, used or queried?

5. Where is a particular variable declared?
6. Where is a particular data object accessed?
7. What are the inputs and outputs of a module?

Furthermore, [96] offered two important properties for making hidden operations of a language clearer to the novice, which we will append to the previous list:

8. Simplicity. There should be a “small number of parts that interact in ways that can be easily understood”
9. Visibility. Novices should be able to view “selected parts and processes” of the model “in action.”

In general, a good tool should be essentially transparent to the user. Otherwise valuable working memory and cognitive resources will be spent on the tool instead of on the modeling task itself. If one must think about a pencil to write with one, the quality of the writing will suffer [9], but at one point however, you need to learn to use a pencil. We will focus on the narrow cognitive gap that a flow-model (see 2.1.3) programming environment provides. We chose this environment out of the other categories of visual programming environments because by taking only those ideas that are needed to model program code, we provide a simple method for showing the flow of data and control to novice programmers. Whereas other very high level programming languages, such as Alice, use a storytelling objective to convince young, under-represented students that programming is satisfying, our approach instead focuses on the cognitive gap between programming using a text-based language and the representation of code visually, using a flow-model technique. Other research using graph-based teaching of textual programming has been performed (see 2.2.3) and we seek to build on their success.

A flow-model tool contains the basic building blocks of all programming languages, and allows the novice to see the text-based representation at will, using an interface that is intuitive to them. Furthermore, novices first develop a control flow abstraction of a program which captures the sequence of operations in the program [97], and since people frequently do not translate a given problem into a normalized internal (mental) representation that is independent of the original external representation [61], it seems natural to represent programs as flowcharts. All notations have syntax, so the real advance in VP is achieved by assisting the programmer to translate a set of design semantics from one syntax to another [98]. By using this method to create solutions, the novice understands the different core concepts of computing, and can implement their solution in any supported language, similarly to how TXL [99] can play a role in helping users see examples of how code constructs in one language will appear in a new language.

Although further iterations of our experimental method could support structures and classes, in a *tiered language* (see Section 2.1.5) fashion, by not making those capabilities available at the start, and by focusing on the basics of programming, we aim not to overwhelm: “Software tools typically have many features which may be overwhelming not only for novice users, but also for expert users. This information overload could be reduced through the use of adaptive interfaces” [85, p. 188].

This type of visual environment removes traditional syntax errors, and instead allows the novice to focus on creating a working program. Semantic errors still will exist, and we would like to measure if novices can find these more quickly in a graph-based visualization by tracing the history of the error visually (because a flow-model

technique utilizes the modification-use relation [100] where lines of control and the history of the data values are obvious).

By starting with an experiment to test if adding visual programming to traditional text-based solutions had any effect on novices' learning, we performed a pilot study, which we discuss in Section 3.1. We discuss the need for a second pilot study with the results found from our first pilot study in Section 3.2. In Section 3.3, we discuss our second pilot study. In Section 3.4 we discuss the test that we give to our participants to measure their understanding. We discuss in Section 3.5 some important and difficult computer science topics and introduce representations of those concepts visually and introduce our script that we will follow in Section 3.6 (the entire script is found in Appendix B).

3.1 The First Pilot Study

We performed our first pilot study with two classes of Introduction to Programming Language students in the fall of 2009 at the University of Alabama in Huntsville. Our hypothesis was that *computer science novices understand computer science concepts (here sequence, iteration, alternative) better, and test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. This pilot study lacked participation from the students of the test group, who merely watched an example that the instructor created as the instructor stepped through the example's execution. We wanted to measure the two groups' final grades, and learn if the test group was affected by the addition of a flow-model representation to their lecture. We also wanted to measure if the visual

learners (as defined by Felder's ILS) would score better than verbal learners (as defined by Felder's ILS) in the test group, and the verbal learners would score better than the visual learners in the control group.

The 11 participants in the test group and the 11 participants in the control group were recruited voluntarily from two classes of the Introduction to Programming Language class. It is important to note that students who take this class cannot apply their credit to a major in Computer Science, so students who take this course do not typically go on to majors in Computer Science. Each participant began by filling out an ILS Questionnaire and returning it to the instructor with a unique identifying number that was associated by their instructor with their grades. By reviewing the test group's homework, test, and in-class assignments, the test group was exposed to seven PowerPoint presentations that contained dataflow and control flow diagrams representing the test group's homework, test, and in-class assignments. An example of a slide of the class presentation is presented in Figure 3.1. Note that these slides were CGs based on our earlier work [74].

Ch 4, page 146, 4.8

```

int main(void)
{
    int x, y, i, j;
    printf("Enter two integers between 1 and 20: ");
    scanf("%d%d", &x, &y);
    for(i = 1; i <= y; i++){
        for(j = 1; j <= x; j++){
            printf("@");
        }
        printf("\n");
    }
    return 0;
}

```

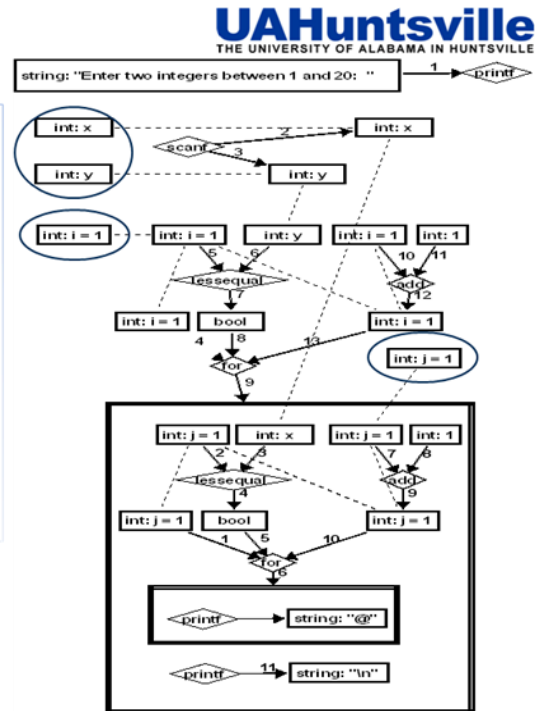


Figure 3.1 A sample slide that was presented to the test group as part of their lecture.

These presentations were created using CharGer [101] to create images that represent snapshots of time while a program was running. The method of creating PowerPoint presentations of flow-model diagrams within CharGer was tedious, time consuming, and error prone, and because the activity was so time consuming for the instructors, it would not benefit the students to have them create their own. At the end of the semester, the instructors gave us the participants' final grades, which were identified by the students' unique identifying number that originated with the ILS Questionnaire.

It should be noted that since our sample size was so small, we had wide error bars representing learning styles, therefore the graph spans both sides of the Sensing/Intuitive, Visual/Verbal, Active/Referential and Sequential/Global axis, as opposed to the much larger sample shown in Figure 1.4, where the error bars are much smaller. The Learning

Styles distribution from the test group (Figure 3.2) and control group (Figure 3.3) follow and the mean and standard deviation are presented in Table 3.1.

Table 3.1 The Test and Control Groups Learning Style Distribution.

	Test Group	Control Group
Sensing/Intuitive	43%±30% Sensing	21%±55% Sensing
Visual/Verbal	26%±46% Visual	65%±18% Visual
Active/Reflective	24%±43% Active	14%±51% Active
Sequential/Global	17%±38% Sequential	1%±52% Global

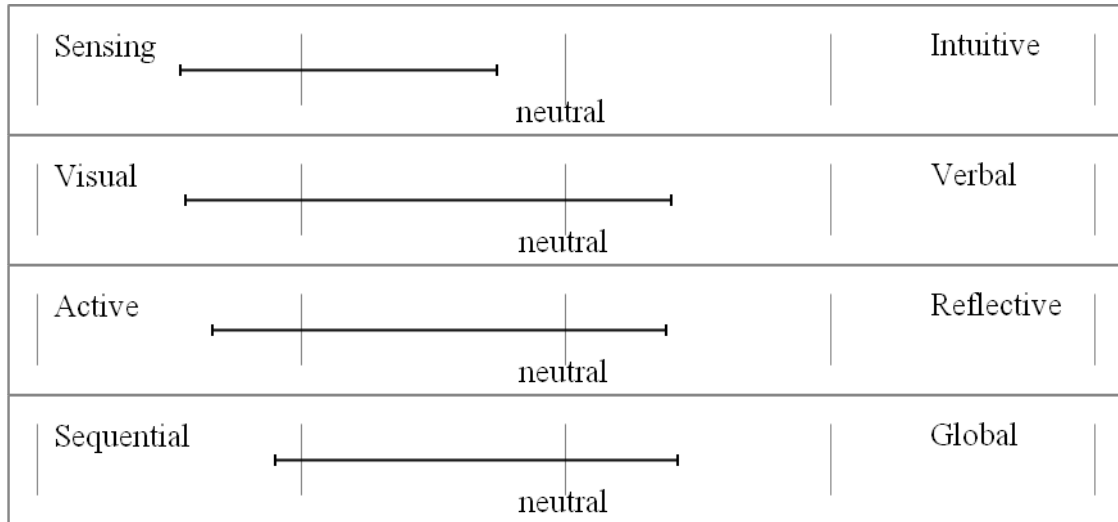


Figure 3.2 The Test Group's Learning Style distribution.

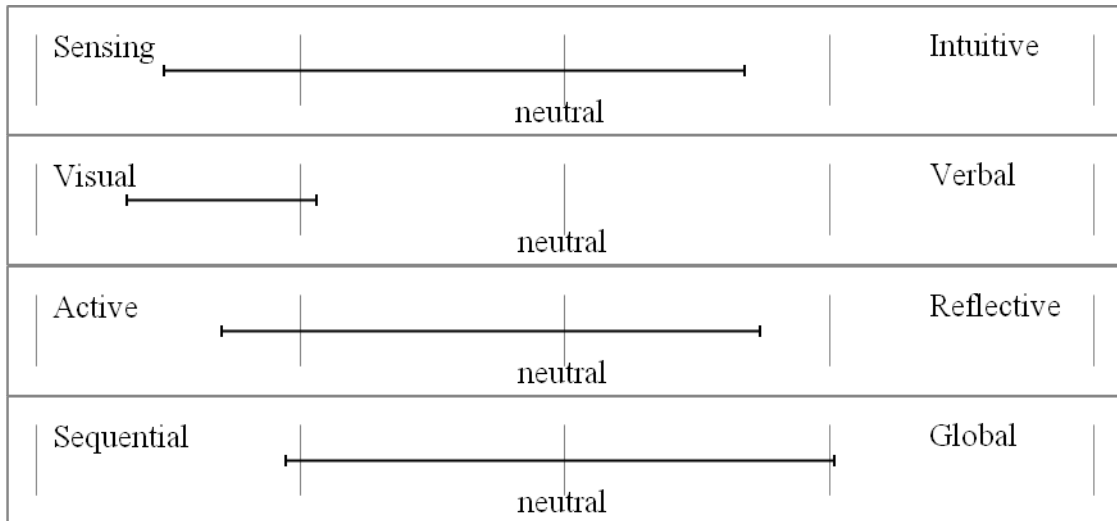


Figure 3.3 The Control Group's Learning Style distribution.

The test group's programming averages and final grades were slightly higher than the control group's programming averages and final grades, as displayed in Table 3.2.

Table 3.2 Final Programming Grade and Final Overall Grade results.

	Test Group	Control Group
Final Programming Grade	96 \pm 4	94 \pm 8
Final Overall Grade	94 \pm 8	92 \pm 8

These results are encouraging, but since they were not statistically significant, and the sample size was too small, no conclusions can be drawn. The instructor of the control group suggested that the students who participated in the study were overachievers, while those students who did not participate were not as driven, and since we did not measure this, we have no measurable response. A possible result of this experiment is that students are overloaded by learning a flow-model language and a text-based language, and do not perform as well, but this is purely speculation.

3.2 Experiments with a Flow-Model Implementation

RAPTOR, as it is, like most visual programming languages, does not scale to enterprise level programs, so it must instead be used as a program understanding tool to enable users to learn another language that will indeed scale.

It is important to keep cognitive load low enough that it won't interfere with learning and with the formation of hierarchical information networks, or *schemas* [102]. Some cognitive load cannot be reduced, since that load is a core characteristic of the material being taught. Furthering this is a generally accepted axiom to “represent as little as possible explicitly” [103, p. 17] while still supporting the solution of the problem at hand. Unnecessary representation in a model simply makes it more difficult to focus on the important aspects of the problem. This is consistent with the view of Alabastro et al. [104] that visual models “significantly increase the speed and quality of model development” by focusing attention on what is absent and important (and should therefore be added and what is present and unimportant (and should therefore be eliminated)” [9, p. 21].

We found that a flow-model technique capitalizes on the narrow cognitive gap between visual programming languages and cognitive understanding. We believe that the cognitive load that cannot be reduced are the same elements that make a program Turing-complete, namely, sequence, alternative, and iteration. Total load levels can be adjusted by adjusting *germane* and *extraneous* cognitive load [105]. Since RAPTOR is Turing-complete, and since a Turing Machine is the basic set of services that must be provided in order to model a fully-functioning computer, RAPTOR is a good platform to use to model a fully-functional computer, and is therefore a good cognitive fit [106-108].

Critics could argue that Turing Machines are hard to create practical applications with due to the extreme detail that is needed to perform actions, and that is why we do not program in Turing machines or assembly language very often. We argue that by thinking in terms of a Turing Machine as the very basic building blocks of computer programs, novices can use these building block pieces to build their programs, from the bottom, up [85].

Many experiments need to be performed in order to completely determine the effect that using a visual programming language to help understand programming concepts has on understanding programming concepts. The experiments would likely involve updating RAPTOR to include suggestions made by Martin Carlisle (see 2.1.3.4) and others, and would likely involve many novice computer scientists, and would take many years. We would want to test many different permutations of sequences. Should a visual student learn visually first, then textually? Should that student learn both methods at the same time? Should classes be separated based on learning style preferences? Although we do not support the last suggestion, we do agree that there are many questions to ask, measure, and analyze. We will start with one set of experiments, which we discuss in the next section.

3.3 Our Second Pilot and Flow-Model Experiment

We used RAPTOR, as it is, as a program understanding augmentation for novice computer scientists. In this way, novices are immersed in programming, without the need to deal with traditional syntax errors, which is only one of the high-risk topics that are difficult to understand for novice programmers and that instructors of introductory

programming think are important to understand [24]. A seamless transition needs to exist so that the novices will learn how to correlate the two (graph-based versus text-based). Our null hypothesis is that *computer science novices will understand computer science concepts (here sequence, iteration, alternative) the same, and will not test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. In our experiment, the above Figure 3.1 now looks like Figure 3.4.

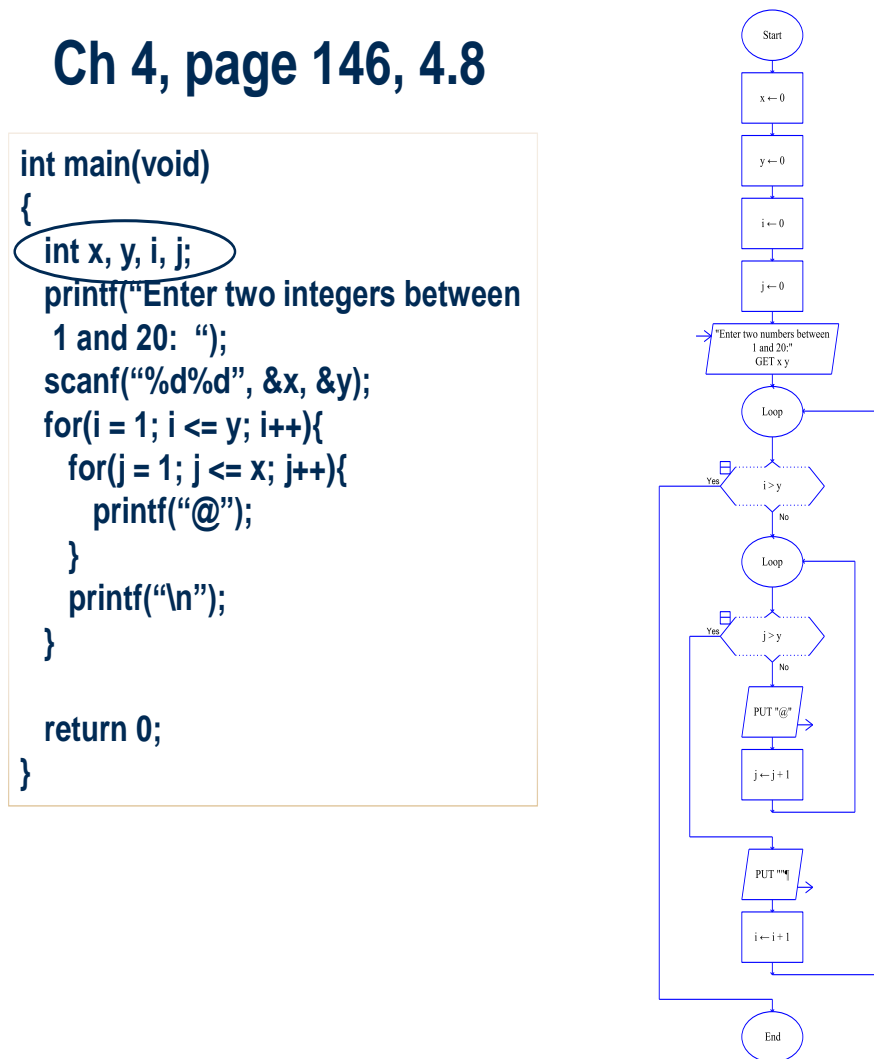


Figure 3.4 Sample RAPTOR Slide.

The total group of novice programmers was drawn from University of Alabama in Huntsville's Psychology department. When a student enrolls in introductory Psychology classes, they are required to participate in a certain number of experiments during the semester (typically about 3 hours worth). All participating students took the ILS Questionnaire. Pretest questions consisted of current GPA, age, sex, left or right handedness, previous math classes and previous math grades, and experience programming (which we expected to be nonexistent). The proctor also collected what time of day the class was held, and how long it took each participant to complete the test. The entire experiment lasted approximately one hour, with the pretest survey taking 15 minutes, the lab taking 20 minutes, and their test taking 15 minutes. Multiple experiments were run during the first two weeks of the semester to account for the various students' schedules. Each class contained no more than 20 students, as this is the capacity of the classroom being used.

Our test group, or the Flow Teaching Style group, was introduced to programming using screenshots of the RAPTOR program, and slides consisted of RAPTOR, while our control group, or Text Teaching Style group, was introduced to programming using C. Sequence, variables, user input, alternative, and iteration were covered, and the entire presentation of concepts took 20 minutes. Each topic consisted of a definition of a new concept, and then its incorporation in instructor-led examples. Both groups took the same test, but in their respective programming language. Tests consisted of presenting code samples and asking the student what they think the code sample will do. We utilized both multiple choice questions and free text questions. We ended with

comparing the final grades from both classes, taking into account their learning styles and whether they were exposed to the Flow (RAPTOR) or Text (C) programming language.

3.4 Validating our Flow Model Experiment: The Test

In order to validate our Flow Model Experiment, we tested our novices. We used the Bloom Taxonomy of Learning as a measure of our test, since it is considered to be a valid benchmark that measures a student's level of understanding in a particular subject [109]. The six levels of Bloom's Taxonomy are described in Table 3.3.

Table 3.3 Bloom's Taxonomy of Learning.

Level 1 – Knowledge	Fact recall with no real understanding behind the meaning of the fact
Level 2 – Comprehension	The ability to grasp the meaning of the material
Level 3 – Application	The ability to use learned material in new and concrete situations
Level 4 – Analysis	The ability to break a complex problem into parts
Level 5 – Synthesis	The ability to put parts together to create a unique new entity
Level 6 - Evaluation	The ability to judge the value of the material for a given purpose

A summary of what can be associated with each level can be found in [48] and we display a summary of this in Table 3.4.

Table 3.4 Bloom's Taxonomy's application to Computer Science.

Level	What Can the Learner Do	Sample Tasks and Assignment
1	Recognize and informally define specific concepts in algorithmics, like stacks, trees, graphs, Quicksort, AVL-tree, linear probing, or basic analysis concepts like Big-O notation and worst-case complexity.	Define the following concepts: directed graph, binary tree, array. List three different sorting algorithms
2	Understand the general principle behind an algorithm and explain how it works using words and figures. Define concepts formally, that is, recognize their essential properties and present them in an exact way.	Write a program that sorts an array of 100 integers using shell sort. Explain why Quicksort is, in its worst case, a quadratic time algorithm.
3	Adapt a previously studied algorithm for some specific application, environment or specific representation of data. Construct the best-case and worst-case analysis of basic algorithms.	Implement a program that sorts a linked list of strings using insertion sort and demonstrate that it works.
4	Understands the relation of the algorithm with other algorithms solving the same or related problems. Be able to analyze a complicated problem, identify essential objects in it and split the problem into manageable smaller problems.	Compare the performance of Quicksort and Heapsort. Argue why Dijkstra's algorithm works. Explain why Prim's algorithm works for graphs containing edges with a negative weight but Dijkstra's algorithm does not.
5	Design solutions to complex problems where several different data structures, algorithms and techniques are needed.	Design a search engine and analyze its efficiency of space and time. Design the data structures and algorithms need by a car navigation system.
6	Discuss the pros and cons of different algorithms that solve the same or similar problems.	Discuss the design of a solution, and argue why it is better or worse than a different solution.

Further suggestions of key words that are associated with each of Bloom's Taxonomy are located in [110] and are presented in Table 3.5 and applications of Bloom's Taxonomy to Computer Science is represented in Table 3.6.

Table 3.5 Key words associated with Bloom's Taxonomy.

Level	Keywords
1	know, name, recall, state, list
2	comprehend, explain, generalize, interpret, predict, summarize, translate
3	applies, computes, demonstrates, manipulates, modifies, produces, and solves.
4	analyzes, compares, contrasts, differentiates, discriminates, selects, and separates
5	combines, compiles revises, summarizes, and organizes
6	criticizes, critiques, evaluates, explains, and summarizes

Table 3.6 Further examples of Computer Science concepts applied to Bloom's Taxonomy.

Level	Examples
1	Name the three kinds of looping structures in C++.
	List three methods of performing I/O in a computer.
	State five things that are true of a RISC architecture
2	Explain in words what happens in the following C++ code. <pre>for(int i = 0; i < 10; i++) { if (i mod 2 == 0) cout << i << endl; }</pre>
	If x is 25 and y is 15, give the values of x, y and z each time through the loop. <pre>z = x % y while z != 0: x = y y = z z = x % y print y</pre>
	Explain why more registers inside the CPU can make the processor faster
3	Students having learned about arrays and records/structures, give them a problem to combine these two concepts together. An appropriate question might be: Create a data structure for a company of 50 workers where each worker has information stored about them that includes their name, job, salary, and number of dependents.
	Modify this for loop so it becomes a while loop
	Give the students a programming problem similar to something done in class or for a programming assignment. The students have done a previous program that converted from miles to kilometers. Then an appropriate problem is: Write a program that reads a temperature in Fahrenheit, converts the temperature to Celsius using $F_c = (F_f - 32) * 9.0/5.0$, and then outputs the result.
4	Compare relative advantages and disadvantages of a RISC architecture with a CISC architecture.
	A program is to read some integer data and then print out the average, median and mode. List all the subprograms needed for this program and describe what each of them does.
5	Ask them to follow up the previous question under analysis by writing a complete program or subroutines for reading data, computing average, median, and mode and printing out the results.
	If the class is studying an object-oriented language ask the students to layout the code for the object. For example: "Give the interface for a class that can represent fractions, including adding, subtracting, multiplying, and dividing fractions.
	Have them illustrate inheritance or polymorphism for a class that has been presented on the test.
6	Give code to the students that has a logic error in the code and then ask: "Find the logic error in this code."
	Give the students code then ask: "Give test data to completely check this code. Explain what is being checked with each set of test data."
	Critique the code given below listing positive and negative points.

We also tested our novices based on common bugs described in the literature. These include the anthropomorphism of computer programs from [64]. Students also have problems with understanding that each instruction is executed in the state that has been created by the previous instructions [23].

Our questions that we used are summarized in Appendix D. We summarize what each question tests here and each question's Bloom Level:

1. Question 1: Bloom Level 2: Alternative - We test if the user will chose the correct path ($\text{Age} < 30$), even though we have asked the user for extraneous information.

The question will be presented as Figure 3.5.

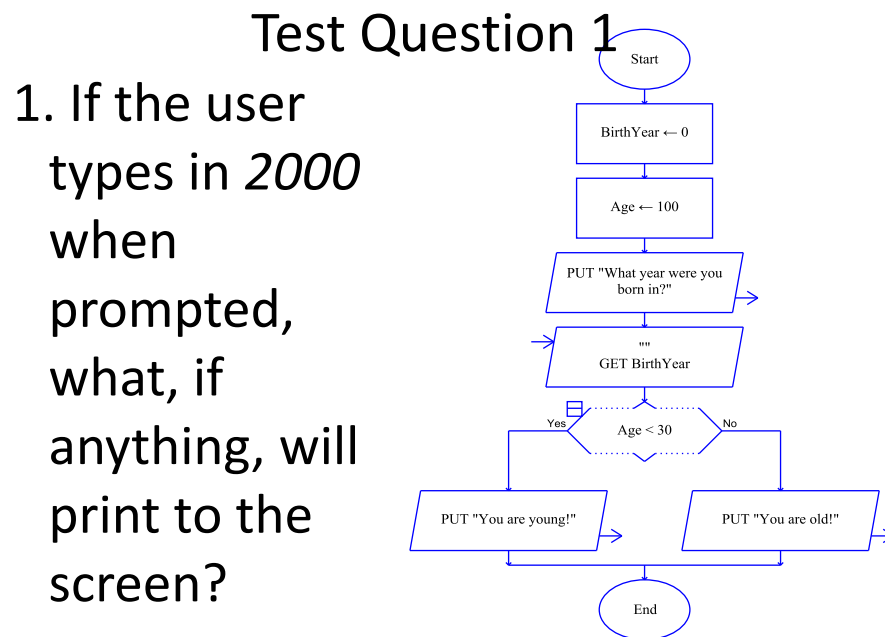


Figure 3.5 Test Question 1, Flow Style.

2. Question 2: Bloom Level 6: Anthropomorphism of programs – We test the user's knowledge that the program only knows the commands that it has read, it does not know what commands exist until they are read.
3. Question 3: Bloom Level 4: Iteration – We test the user's understanding of how iteration must update the variable that the iteration is using to exit the iteration.
4. Question 4, 5: Bloom Level 1: Variable identification – We test the user's ability to identify what a variable is.
5. Question 6: Bloom Level 2: Alternative – We test the user's understanding of math (that 2010 minus 1975 is 35) and conditionals (that 35 is not greater than 35).
6. Question 7: Bloom Level 3: Comparing iteration to alternative – We test the user's ability to compare an iterative statement to a comparative statement.
7. Question 8, 9: Bloom Level 2: Alternative and Sequence – We test the user's comprehension of an alternative statement in probably the most straightforward way possible.
8. Question 10: Bloom Level 6: Software Errors – We test the user's ability to identify extraneous information and also to identify a logic error in the software by reading what the software is supposed to do, and then try to identify what the software is actually doing.
9. Question 11: Bloom Level 4: Iteration – We test the user's understanding of how iteration does not have to start at 0.

10. Question 12, 13, 14: Bloom Level 6: Math, Iteration, Alternative – We test the user’s understanding of variable initialization, iteration, and alternative together based on different hypothetical input.
11. Question 15: Bloom Level 2: Math – We test the user’s understanding of how to calculate the area of a circle given certain input.
12. Question 16: Bloom Level 2: Math, Sequence – We test the user’s understanding of how to calculate the area of a square given a side. This question was chosen to help the test-taker see what Question 1’s code should look like if Question 1’s code were set up correctly.
13. Question 17: Bloom Level 2: Math, Alternative – We test the user’s understanding of conditionals (is $1000 > 500$?) and also of dividing 1000 by 2.
14. Question 18: Bloom Level 2: Alternative – We test the user’s understanding of *if-else if* statements.
15. Question 19: Bloom Level 2: Alternative – We test the user’s understanding of successive *if* statements.
16. Question 20: Bloom Level 2: Alternative – We test the user’s understanding of an *if-else* statement followed by an *if* statement.

3.5 The Important and Difficult Introductory Computer Science Concepts

If we begin with Table 1.7 and normalize the Important and Difficult values, we get Table 3.7, which we use as an outline to develop visual computer science concepts. The Aggregate Scale is the Important Scale of a concept from Table 1.7 multiplied by the Difficulty Scale from Table 1.7 of the same concept.

Table 3.7 Normalized scale of importance and difficulty.

Aggregate Scale	Computer Science Concept	Section
14.06	Algorithm Design	
12.6	Design Multiple Classes	
12.24	Divide and conquer (decomposition of a problem), stepwise refinement or other problem solving strategies	
12.16	Debugging	
12.04	Parameters	
11.96	Simple Data Structures	
11.76	Scope	
11.7	Method Design	
11.52	Pointer and References	
11.47	Object Communication	
11.47	Polymorphism and Inheritance	
11.34	Objects and Classes	
11.28	Selection and Iteration	Appendix B
11.2	Design Single Class	
10.64	Recursion	
10.56	Static vs. Non-Static	
10.53	Instance and other types of variables	
10.36	Encapsulation	
10.25	Syntax	
8.97	IDE	2.2.1
8.88	Advanced Data Structures	
8.84	Libraries	
8.84	Mental Model	
8.5	CRC-cards	
8.14	Algorithm Efficiency	
7.77	Generics	
6.9	UML Class Diagrams	
5.94	Ethics	

From this list, we identify several factors that are both important and difficult to understand, and show examples of visualizations of them. Although not strictly in order of our aggregate Difficult and Important scale, we will first introduce programming in general, and then cover Selection, Iteration, and Functions. We note that the study of visualization covered in 2.2.1 can address understanding the IDE. We introduce the following section, one part of our script, not as a tutorial of programming tasks to the reader, but instead as reproducible script to follow for this experiment. The entire script can be found in Appendix B.

3.6 The Script

In an effort to reduce the author's bias when performing the experiments, we present the script that was followed during the experiments for analysis that the two groups were treated equally when learning computer science concepts. The following sections correspond to the slides for both the Text and Flow groups. A "T" will indicate that the slide was presented to the Text group, and an "F" will indicate that the slide was presented to the Flow group. In the event that both groups were presented the same slide, then a "S" will indicate the slide (which typically occurred at the beginning of the experiment). For example, 5F indicates that the script corresponds to the fifth slide that the Flow group was exposed to and 6F, 10T would indicate that the slide corresponds to the sixth slide that the Flow group was exposed to and the tenth slide that the Text group was exposed to (which typically occurs when displaying the output of both programs). If the differences in the script between both the Flow and Text groups are more than a few sentences, or if the same concept spans multiple slides, then we have

displayed the scripts in a table. When not necessary, intermediate slides that consist of steps of execution are omitted and can be found in Appendix B. In this section, the italicized text indicates that the words are read to the participants.

3.6.1 Introduction and Agenda

S1. Hello and thanks for volunteering for this experiment. I'm B.J. Smith and I'm working on computer science understanding. S2. I'm going to ask you to fill out a survey, and then we are going to learn about some computer science concepts for about 25 minutes, and then test these concepts. You may now fill out your 5 page survey. I've allocated 15 minutes for you to do this.

3.6.2 A Computer Program

S3. Generally, a computer program is a series of commands that are understood by a computer. It's written in an EXACT language that is unambiguous; a computer can understand a program in only one way. English wouldn't work as a programming language because of such ambiguities as the line "Herb roasted chicken," which could mean that a guy named Herb is roasting a chicken, or it could describe a chicken that has been roasted with herbs and spices. Software runs commands in order, and will do whatever you ask it to do, as long as you ask in a language that it understands. Some popular computer languages are C, C++, C#, Java, and Python.

3.6.3 Sequence

S4. *So this is our first concept that we will learn. Sequence. Programs are read and understood from top to bottom, in order, following the arrows, and until a line has been reached, the computer doesn't know about it. Let's look at an example.*

Table 3.8 Slide 5 Flow and Text scripts.

Slide 5 Flow Script	Slide 5 Text Script
<i>So, this is probably the first program that you guys have ever seen. Most computer languages have some sort of “start” and an “end” as shown in Figure 3.6.</i>	<i>So, this is probably the first program that you guys have ever seen. Every C program has a “main” function. The words “void” here are more advanced for this class, but you have to have something close to that there.</i>

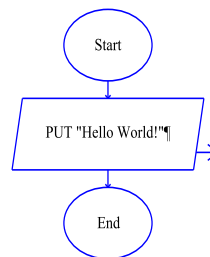


Figure 3.6 Slide 5F, presenting “Hello World!” using RAPTOR.

Table 3.9 Slide 6 Flow and Text scripts.

Slide 6 Flow Script	Slide 6 Text Script
<i>Note that there are arrows. There are arrows between the shapes, vertically, which indicate to you, and to the computer, the direction that the code travels. There is also an arrow that points to the right, which indicates that data is going out to the screen.</i>	<i>Note that for every opening bracket, this is a corresponding closing bracket, and that each command and most lines, here, “printf”, ends with a semicolon</i>

S7. So this particular program would print “Hello World” to the screen as shown in Figure 3.7.

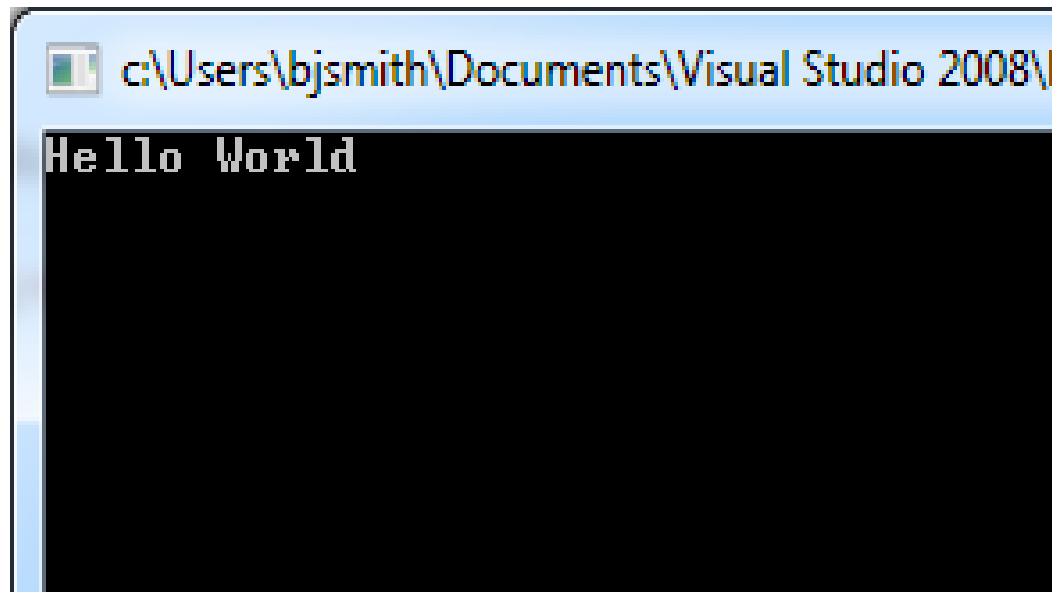


Figure 3.7 Slide 8, displaying Hello World in a command prompt.

S8. Now this is a real simple program, but let's walk through it.

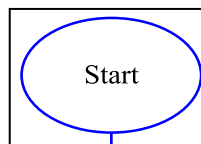


Figure 3.8 RAPTOR's Start.

Table 3.10 Slide 9 Flow and Text scripts.

Slide 9 Flow Script	Slide 9 Text Script
<i>This is where you start. Remember, the computer doesn't know what all is in the program at the beginning, even though we do, it only sees the Start.</i>	<i>You start at the “main.” Remember, the computer doesn't know what all is in the program at the beginning, even though we do, it only sees the beginning line.</i>

10T. The program then reads the next line, finds an opening bracket, which indicates to the computer that there is something inside that the computer can understand. It also means that there must be a corresponding closing bracket, and we'll see that later.

Table 3.11 Slide 10 Flow and Slide 11 Text scripts.

Slide 10 Flow Script	Slide 11 Text Script
<i>We only have one direction to go from Start, which is to the PUT command. This command, PUT, means print to the screen, and whatever comes after PUT, in quotation marks mean "put what is in the quotation marks on the screen."</i>	<i>The next line is a command. This command, "printf," means print to the screen, and whatever comes after printf, surrounded by quotation marks, and these quotation marks enclosed by parenthesis, will be printed to the screen.</i>

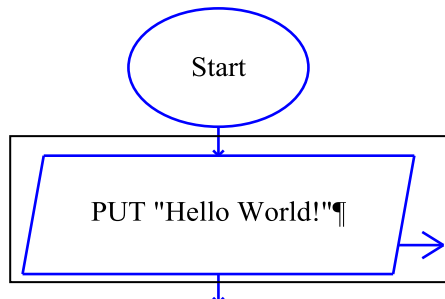


Figure 3.9 Slide 10F, RAPTOR's PUT statement, PUT "Hello World!"

Table 3.12 Slide 11 Flow and Slide 12 Text scripts.

Slide 11 Flow Script	Slide 12 Text Script
<i>There is only one way to go from here, and that is to the End.</i>	<i>The next line is the end of the program.</i>

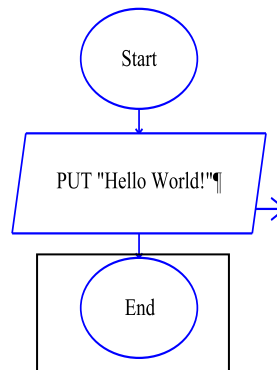


Figure 3.10 Slide 11F, RAPTOR's End.

12F, 13T. *Now, this program was not very interesting, because we knew already, just by looking at it, what the program was going to do. What we need is a variable to hold some value that could change.*

3.7 Summary

We would like to measure the impact of augmenting the traditional way of understanding programming with a visual representation of textual programming to address the difficulty many novices have with learning programming. We believe that using this script as a foundation, we can measure what components of computer science novices understand better when the same material is presented using either a Flow Teaching Style or the Text Teaching Style.

CHAPTER IV

RAW RESULTS OF OUR EXPERIMENTS

We will now present the data from the First Pilot Study (with 22 introductory Computer Science students), the Second Pilot Study (with 12 students from many different majors who enrolled in a psychology class), and the entire Experiment Population (with 172 students from many different majors who enrolled in a psychology class). Within our Experiment Population, we have three subpopulations that we will consider. The first population is the Experiment, and consists of everyone who participated (172 students). We divide the Experiment population based on Teaching Style, which are Flow and Text. The Flow group contains 82 students and the Text group contains 90 students. First, let us introduce the statistics used in the study.

In Sections 4.1.1, 4.2.1 and 4.3.1, we will introduce the Frequency Distribution each population (First Pilot, Second Pilot, and Entire Experiment/Flow group/Text group) and then present the T-Test using Independent Samples to compare the two different groups of students (Flow and Text). We will at least articulate the number of students, the number of males and females, the distribution of the class levels (High School, Freshman, Sophomore, Junior, Senior), the distribution of the students' Grade Point Average, and the distribution of the ILS scores.

We will display the number of people who scored each value on the ILS questionnaire (11, 9, 7, 5, 3, 1 for Active or Reflective, Sensing or Intuitive, Visual or Verbal, Global or Sequential). We then calculate the percentage of respondents who are Active or Reflective, Sensing or Intuitive, Visual or Verbal, and Global or Sequential. Last, we use the respondents' scores to measure a weighted average of *how* Active or Reflective, Sensing or Intuitive, Visual or Verbal, and Global or Sequential they are. For example, if we have 2 respondents, and one scores an 11 Active and the other scores a 3 Reflective, then we would consider the group evenly split, with one Active and one Reflective respondent, so our population is 50% Active and 50% Reflective. Using this same sample, however, we would say that the weighted average would be 78.6% Active ($11/(11+3)$) and 21.4% ($3/(11+3)$) Reflective.

Finally in Sections 4.2.3 and 0 we will present Cronbach's Alpha describing the reliability of the tests used in the Second Pilot and the Experiment. Cronbach's Alpha is a coefficient of reliability that is commonly used as a measure of the internal consistency of a psychometric test score for a sample of examinees. While higher alpha values are more desirable, a reliability of .70 is sufficient for our work.

We will present the Correlations and Linear Regressions in Chapter 5.

4.1 The First Pilot Study

This section describes the results of the first pilot study. The environment was a computer lab and each student present sat in front of a computer with access to the internet. A projector with a computer attached to it was used to present the slides to the

students. Eleven students were in the control group, and eleven students were in the test group. The experiment took place during the entire fall 2009 semester.

4.1.1 Frequency Distribution

The students' age distribution is presented in Figure 4.1.

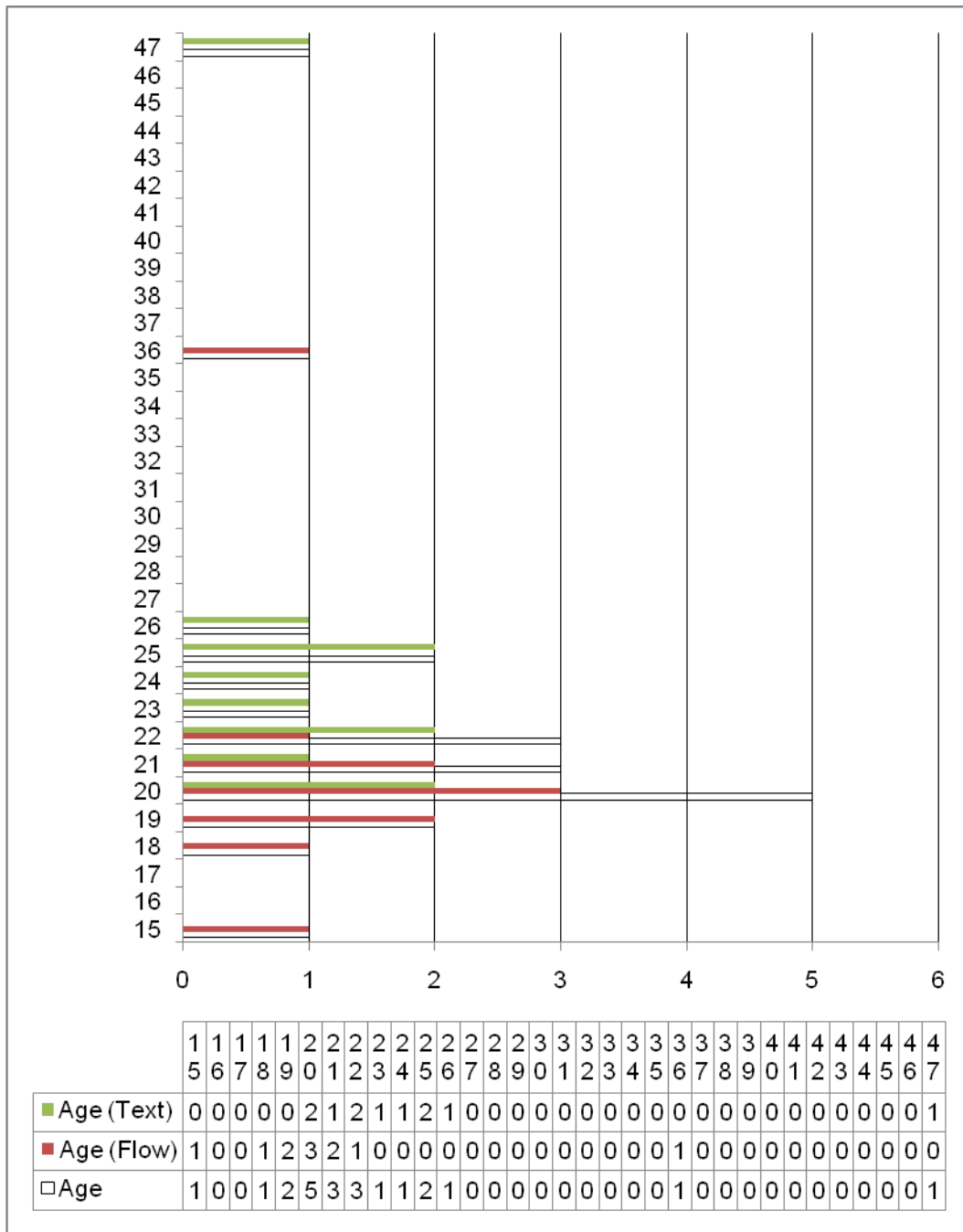


Figure 4.1 First Pilot Age Frequency Distribution.

The students' gender distribution is presented in Figure 4.2.

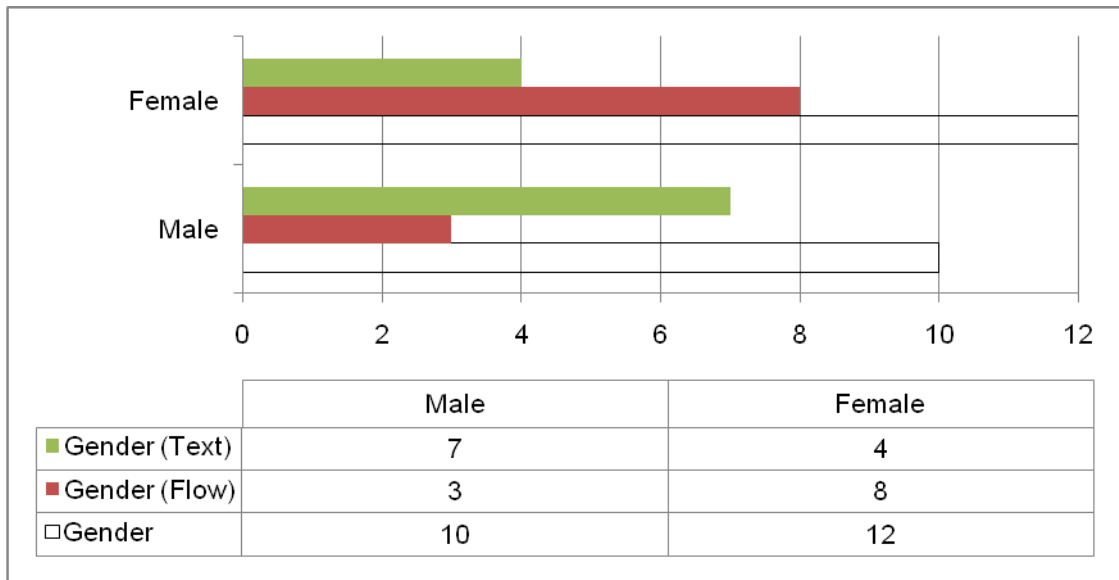


Figure 4.2 First Pilot Gender Frequency Distribution.

The students' class level distribution is presented in Figure 4.3.

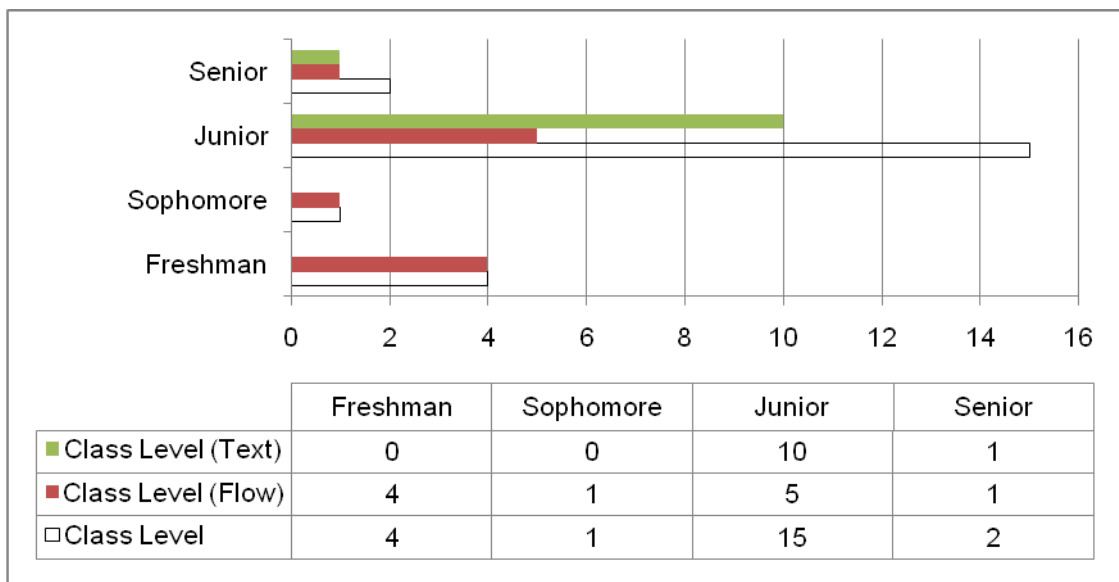


Figure 4.3 First Pilot Class Level Frequency Distribution.

The students' GPA Range distribution is presented in Figure 4.4.

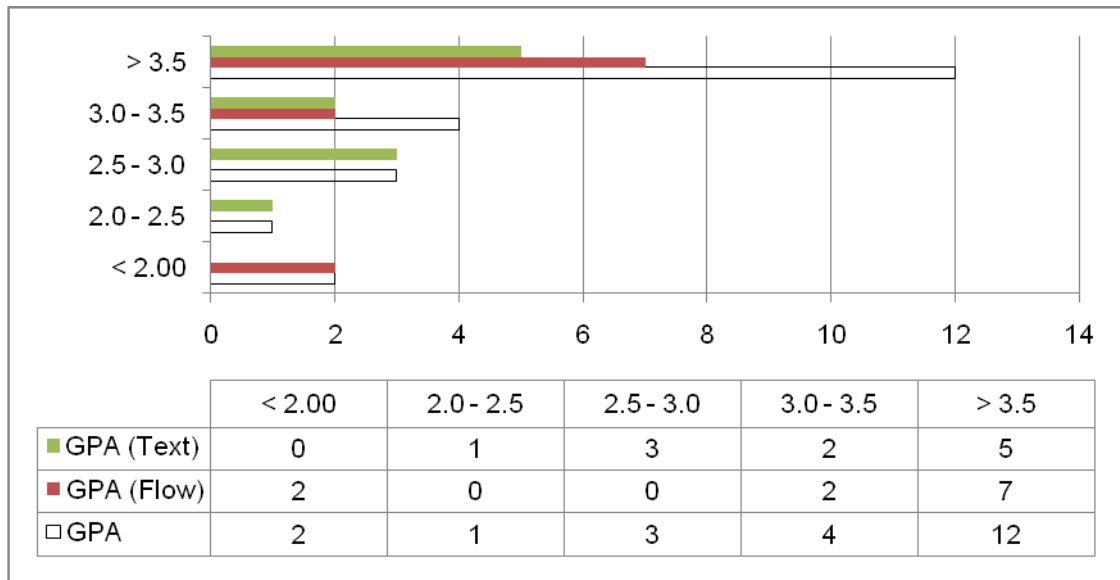


Figure 4.4 First Pilot GPA Range Frequency Distribution.

The students' Active or Reflective distribution is presented in Figure 4.5, Figure 4.6, and Figure 4.7.

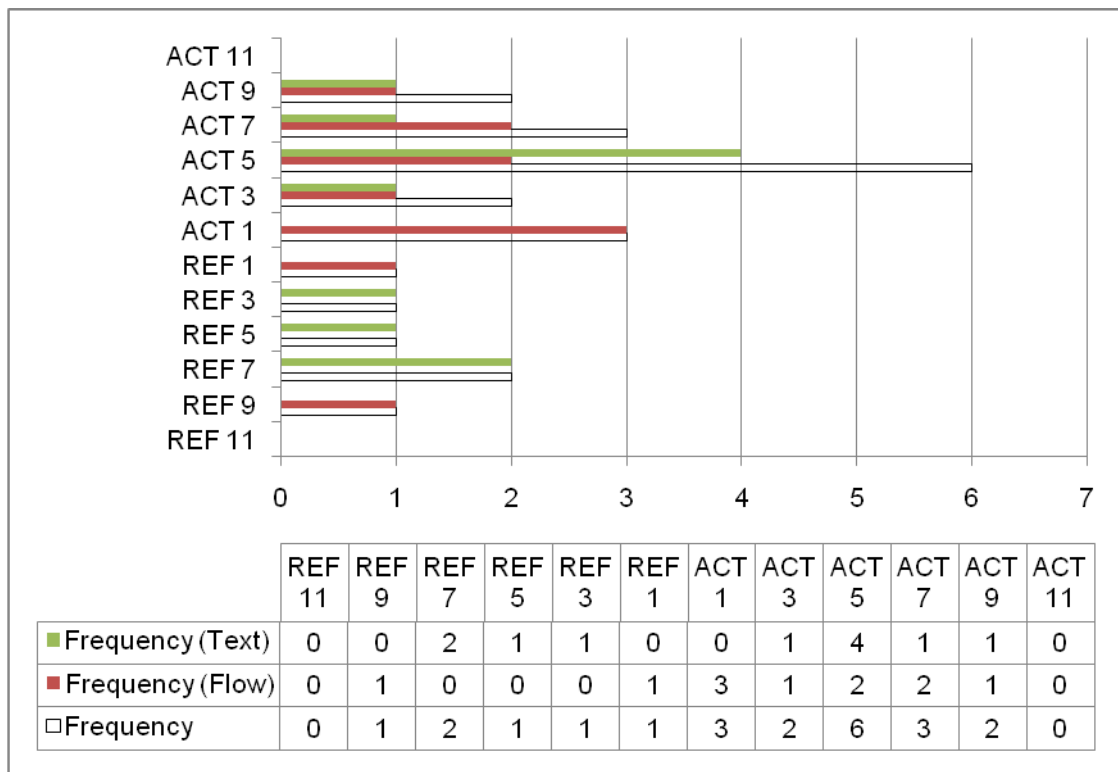


Figure 4.5 First Pilot Active or Reflective Frequency Distribution.

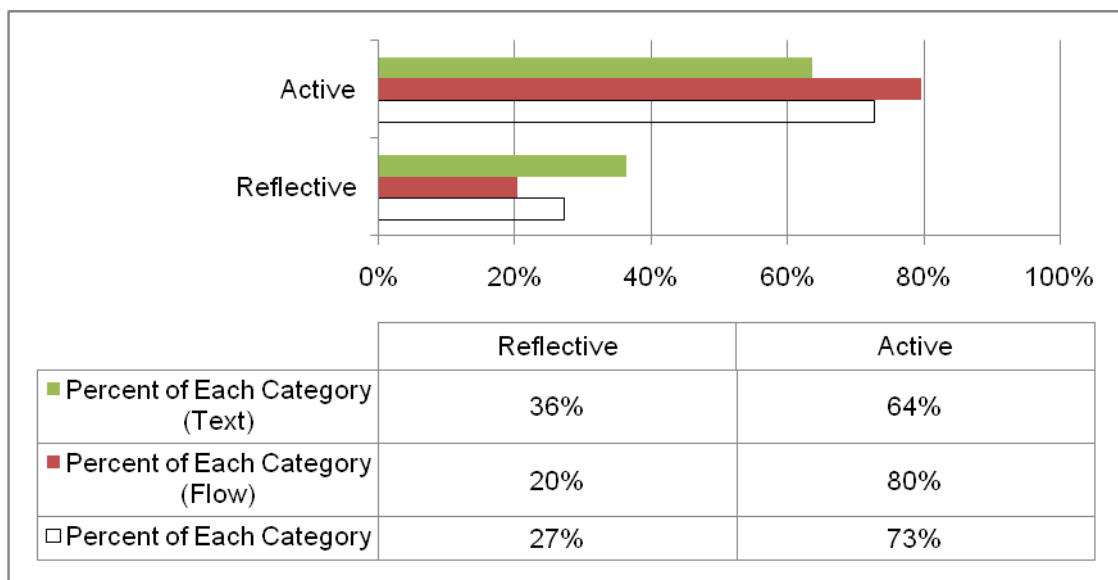


Figure 4.6 First Pilot Percent of Active or Reflective Novices.

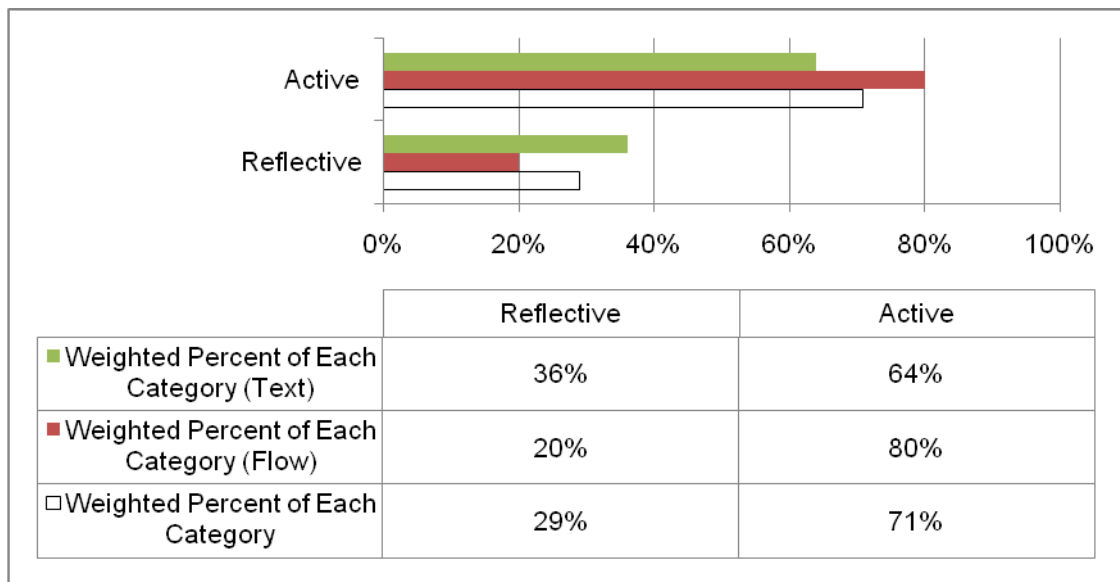


Figure 4.7 First Pilot Weighted Percent Active or Reflective Scores.

The students' Sensing or Intuitive distribution is presented in Figure 4.8, Figure 4.9, and Figure 4.10.

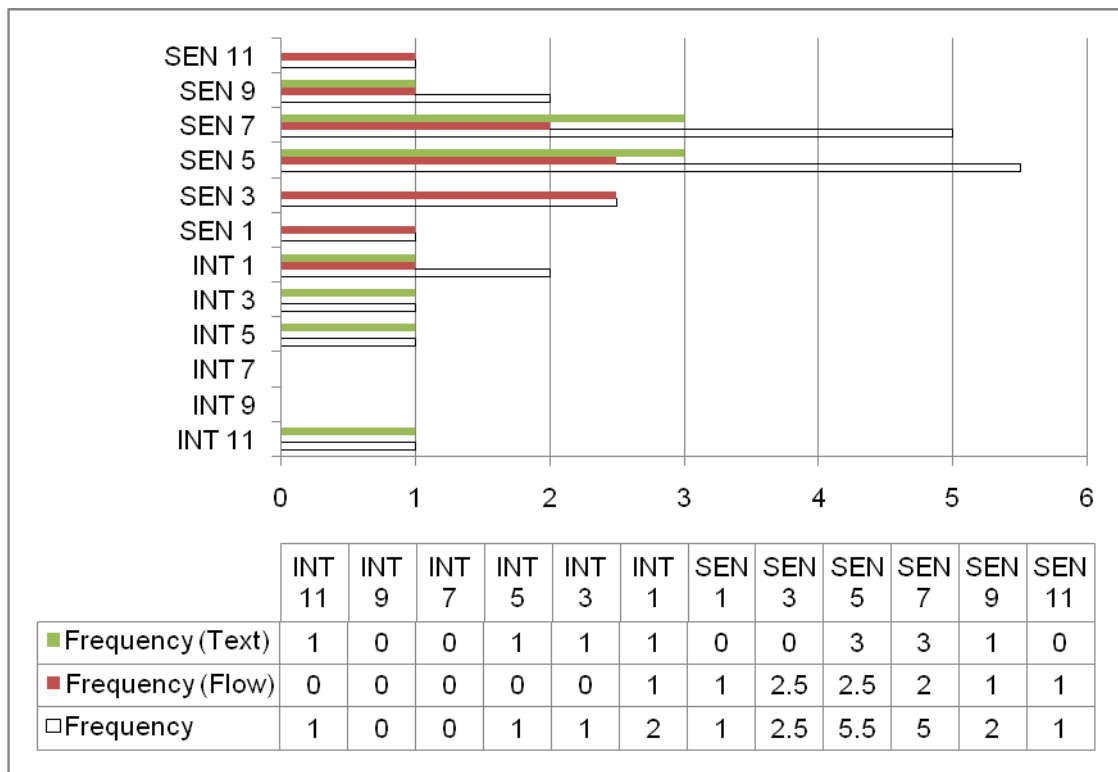


Figure 4.8 First Pilot Sensing or Intuitive Frequency Distribution.

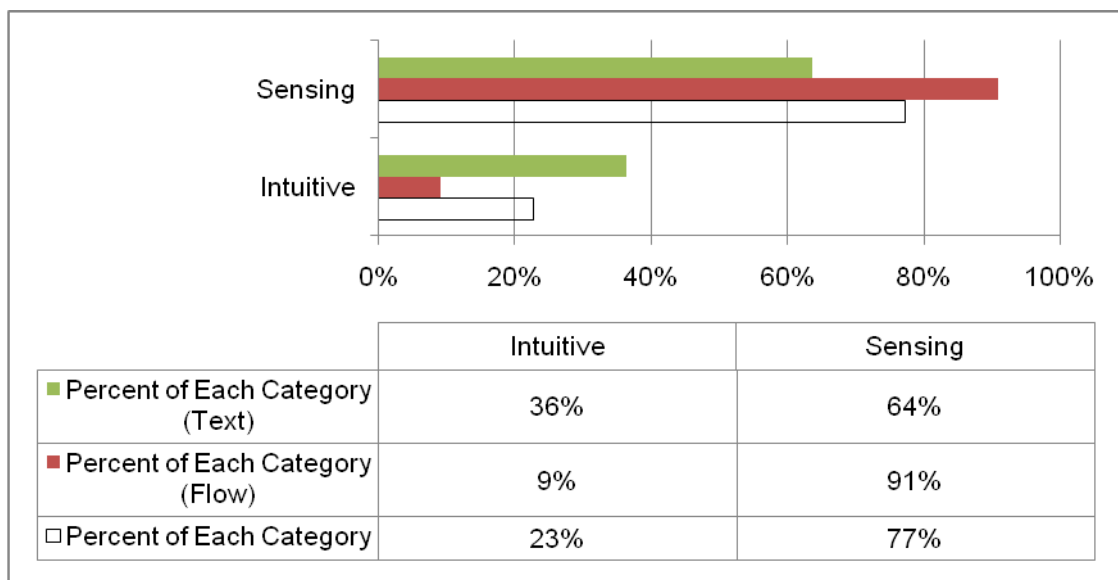


Figure 4.9 First Pilot Percent of Sensing or Intuitive Novices.

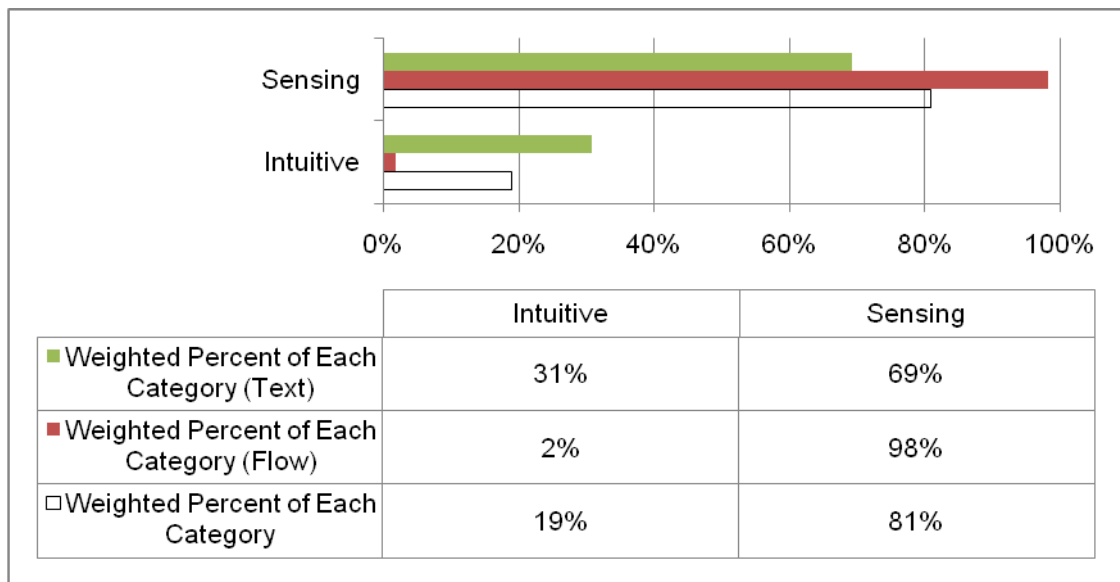


Figure 4.10 First Pilot Weighted Percent Sensing or Intuitive Scores.

The students' Visual or Verbal distribution is presented in Figure 4.11, Figure 4.12, and Figure 4.13.

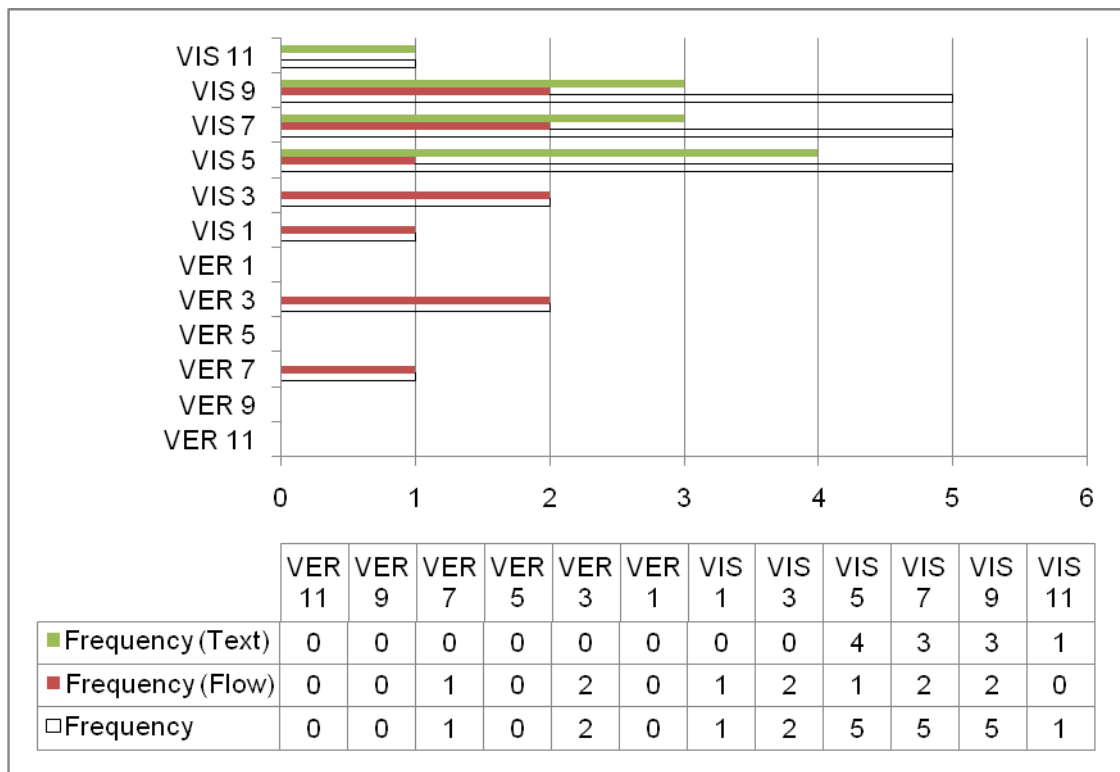


Figure 4.11 First Pilot Visual or Verbal Frequency Distribution.

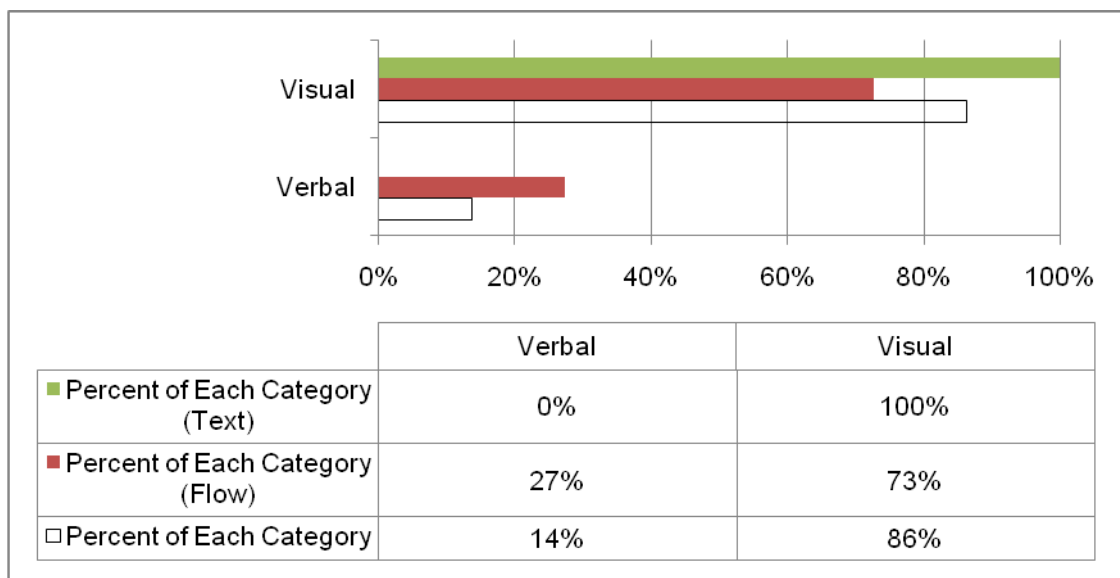


Figure 4.12 First Pilot Percent of Visual or Verbal Novices.

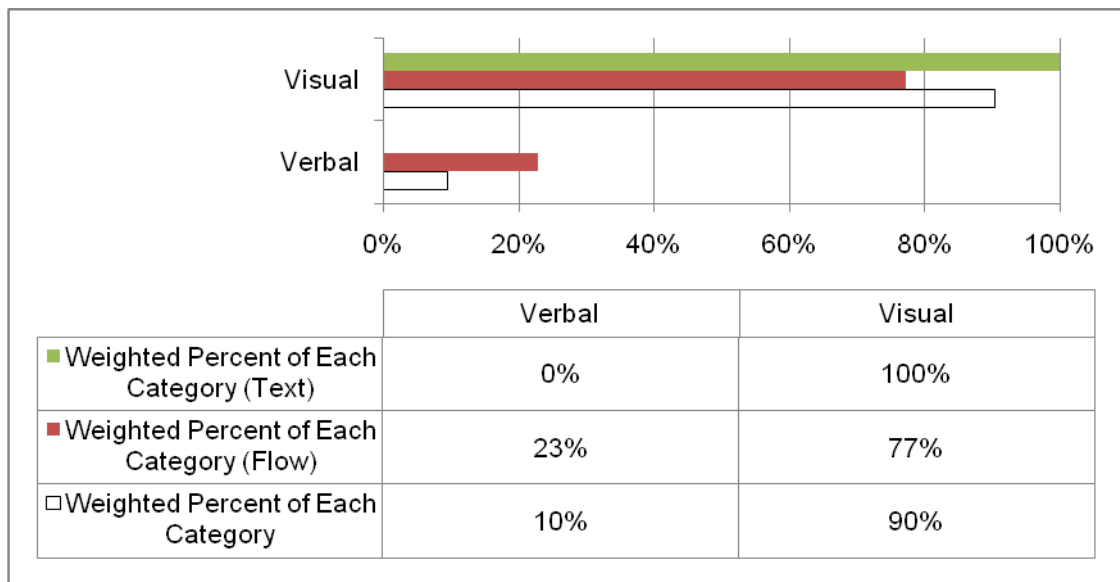


Figure 4.13 First Pilot Weighted Percent of Visual or Verbal Scores.

The students' Sequential or Global distribution is presented in Figure 4.14, Figure 4.15, and Figure 4.16.

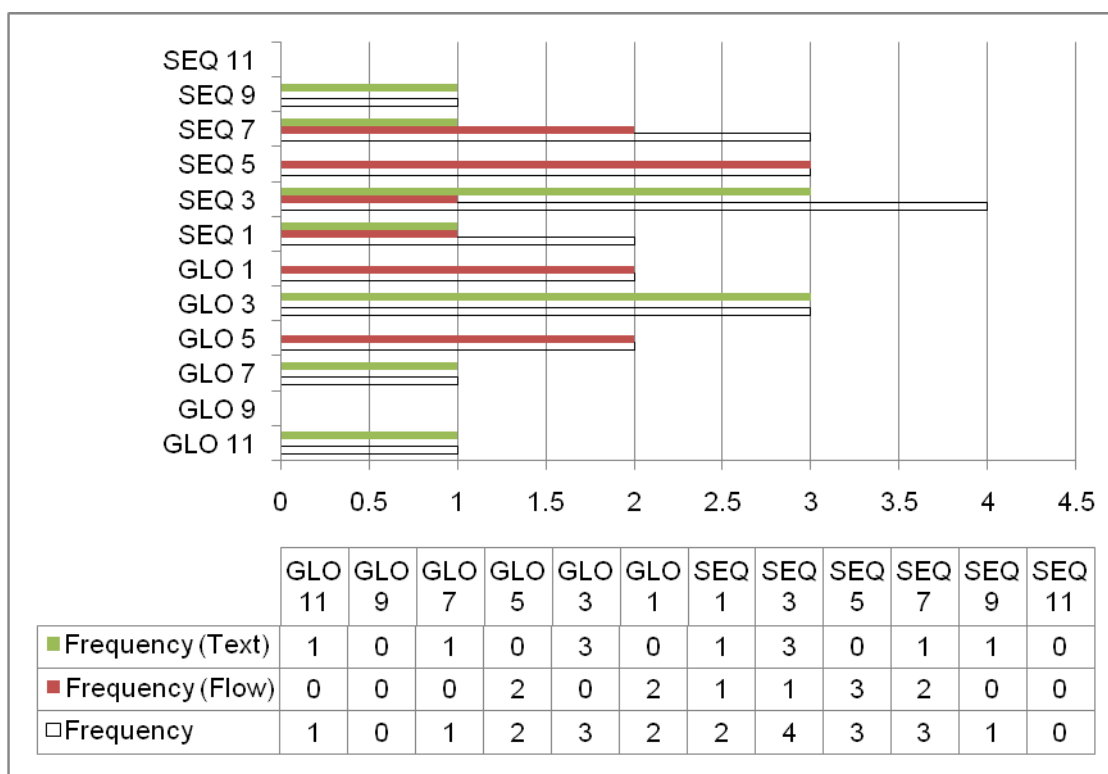


Figure 4.14 First Pilot Sequential or Global Frequency Distribution.

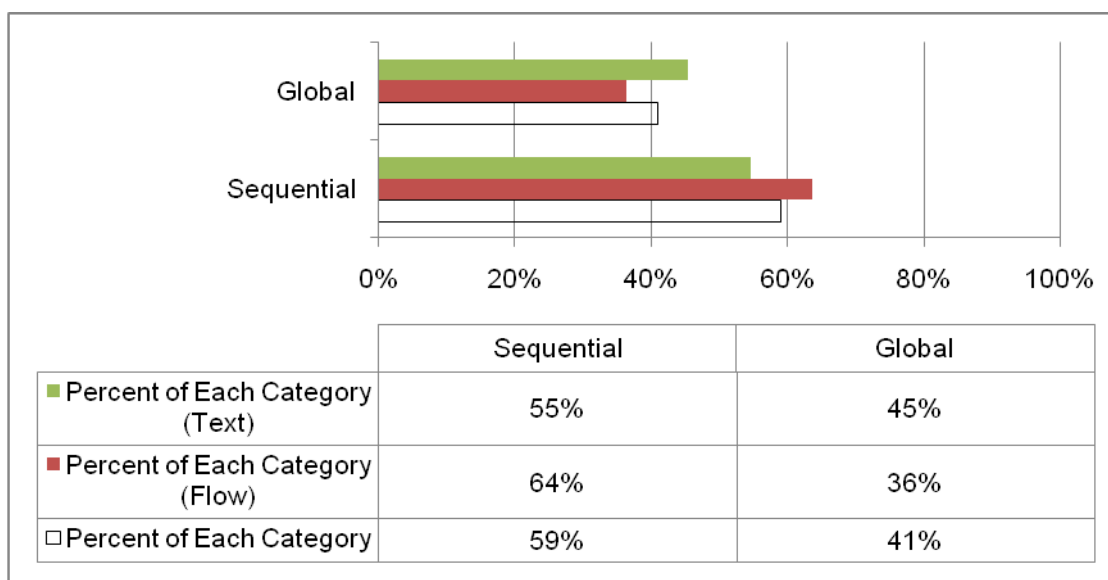


Figure 4.15 First Pilot Percent of Sequential or Global Students.

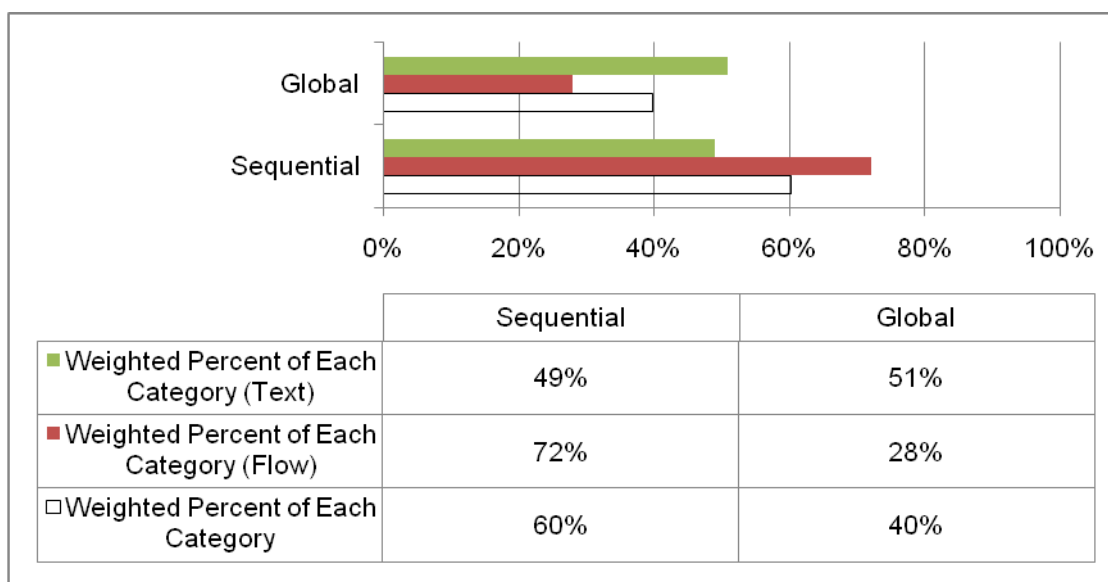


Figure 4.16 First Pilot Weighted Percent of Sequential or Global Scores.

The First Pilot's Program and Final Score statistics and Independent Samples Test are presented in Table 4.1.

Table 4.1 First Pilot Program and Final Score Statistics and Independent Samples Test.

Flow or Text Treatment	n	Mean	Std. Deviation	Std. Error Mean	t	df	Sig. (2-tailed)
Program Grade Flow	11	95.91	4.28	1.29	.685	20	.501
Program Grade Text	11	94.00	8.20	2.47			
Final Grade Flow	11	93.79	8.58	2.59	.613	20	.547
Final Grade Text	11	91.55	8.62	2.60			

4.2 The Second Pilot Study

This section describes the results of the second pilot study. The environment was a class room with a projector with a computer attached to it that was used to present the slides to the students. Six students were in the control group, and six students were in the

test group. The experiment took place during six one-hour time periods during the summer 2010 semester.

4.2.1 Frequency Distribution

The students' age distribution is presented in Figure 4.17.

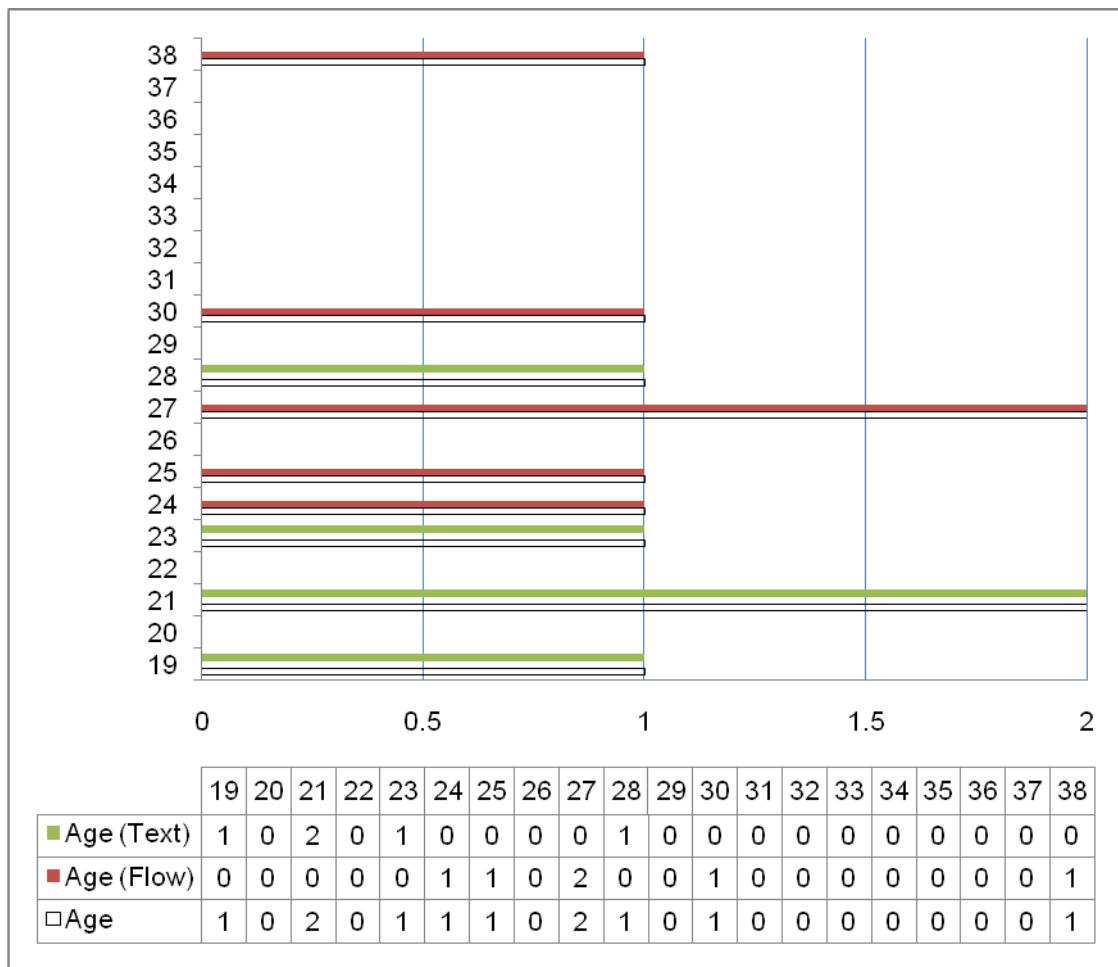


Figure 4.17 Second Pilot Age Distribution.

The students' gender distribution is presented in Figure 4.18.

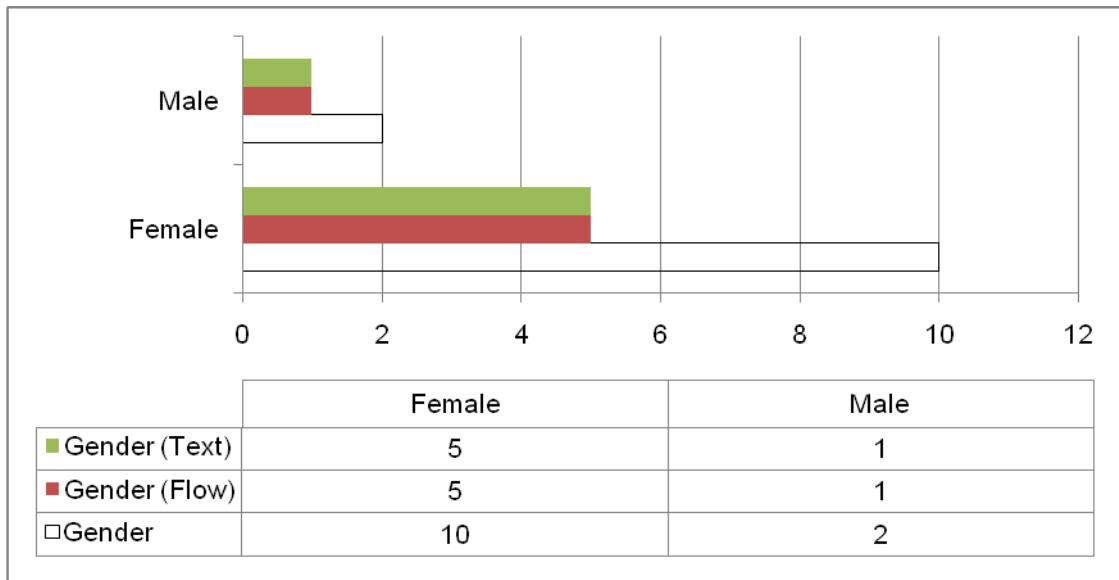


Figure 4.18 Second Pilot Gender Distribution.

The students' class level distribution is presented in Figure 4.19.

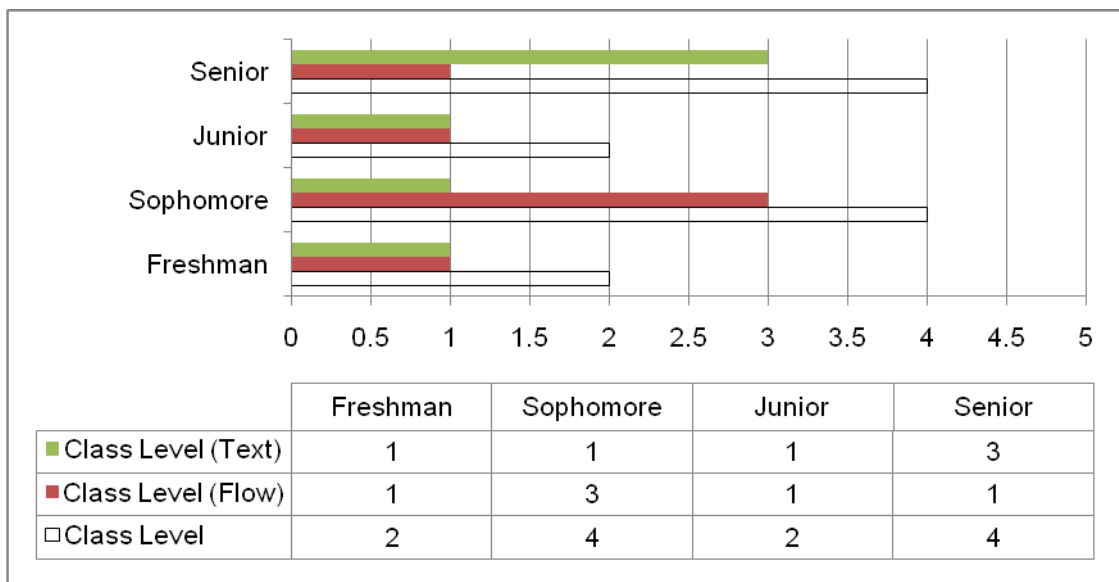


Figure 4.19 Second Pilot Class Level Distribution.

The students' GPA Range distribution is presented in is presented in Figure 4.20.

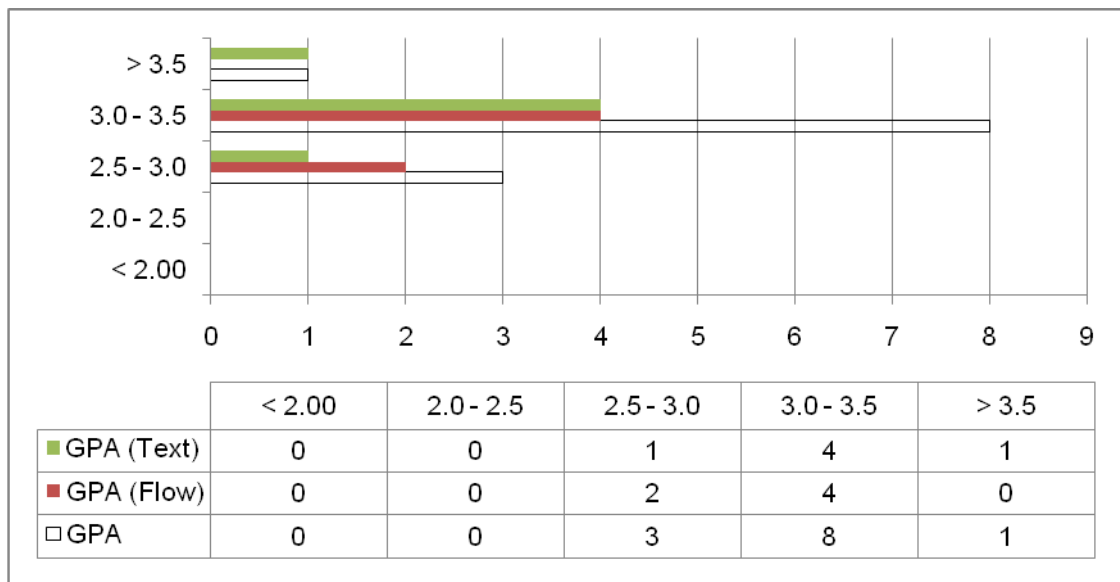


Figure 4.20 Second Pilot GPA Distribution.

The students' Active or Reflective distribution is presented in Figure 4.21, Figure 4.22, and Figure 4.23.

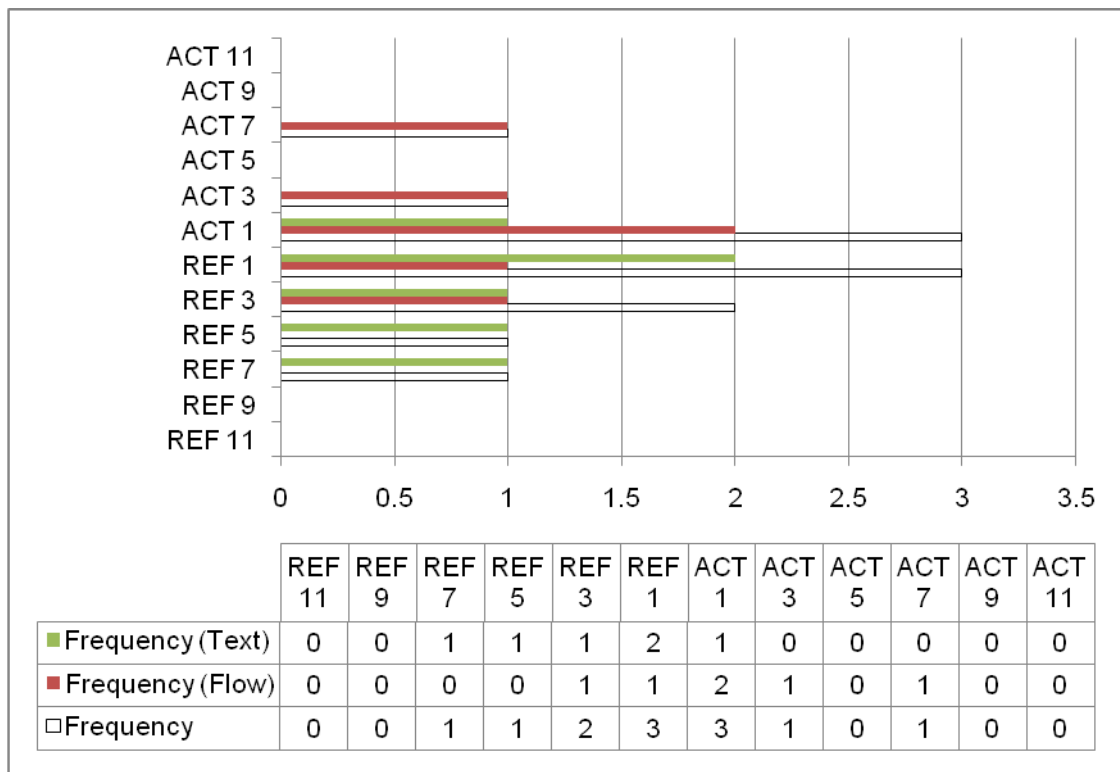


Figure 4.21 Second Pilot Active or Reflective Frequency Distribution.

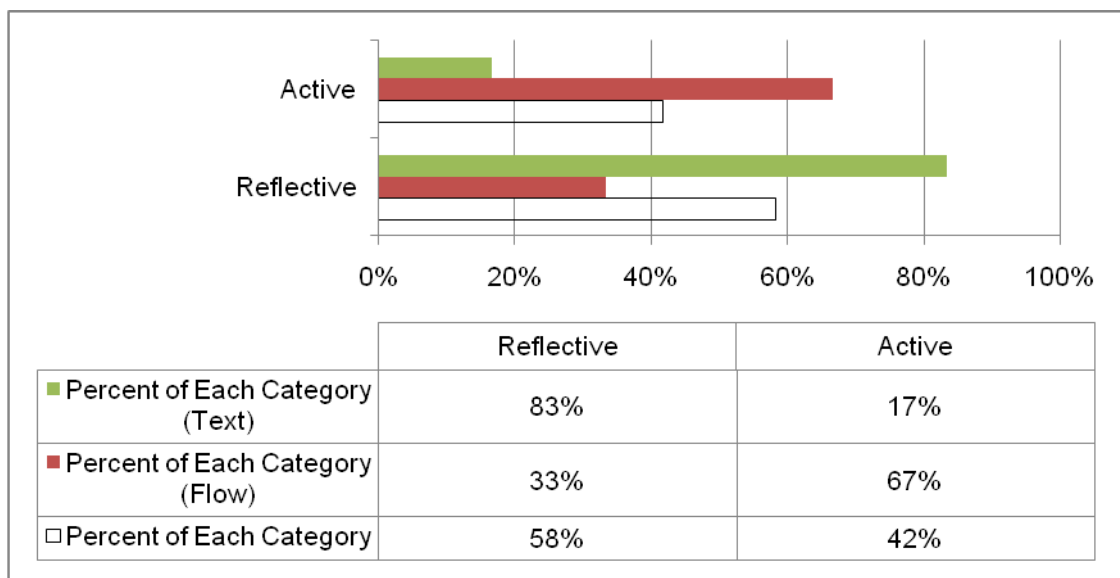


Figure 4.22 Second Pilot Percent of Active or Reflective Students.

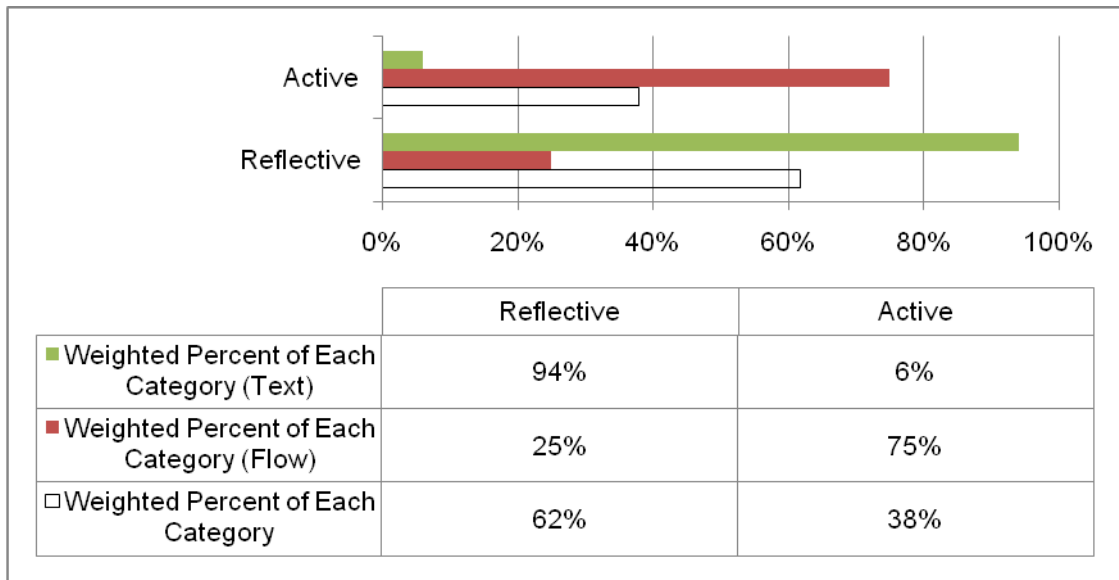


Figure 4.23 Second Pilot Weighted Percent of Active or Reflective Scores.

The students' Sensing or Intuitive distribution is presented in Figure 4.24, Figure 4.25, and Figure 4.26.

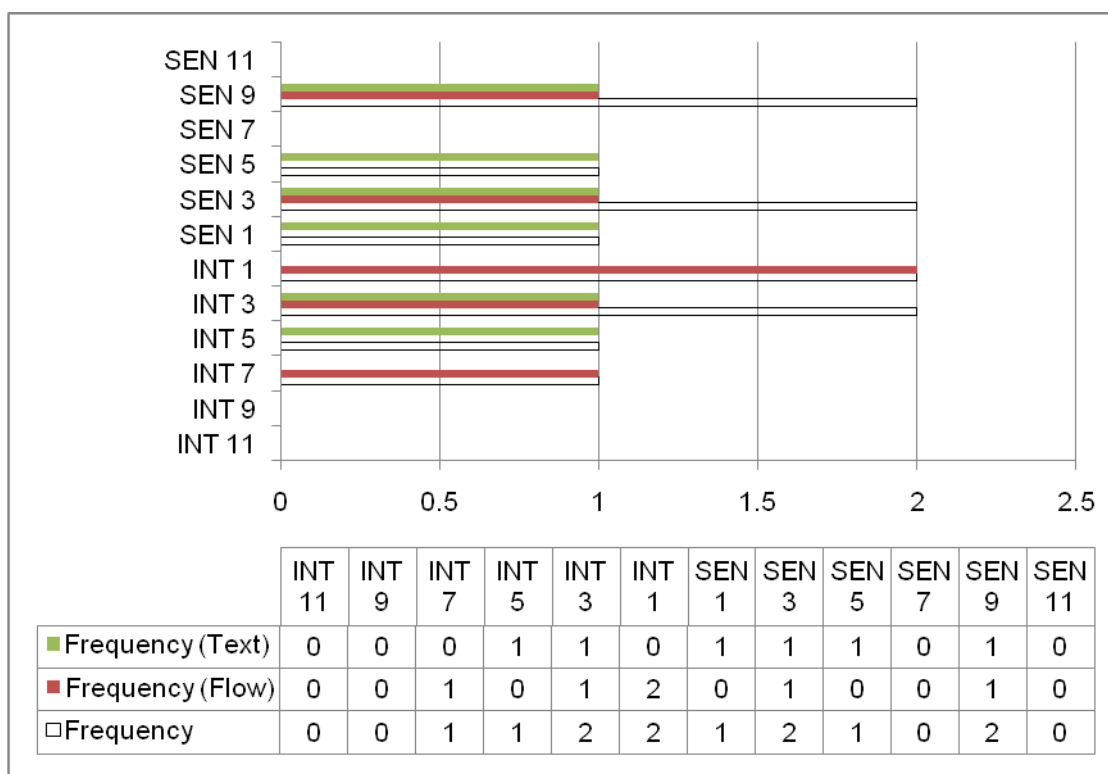


Figure 4.24 Second Pilot Sensing or Intuitive Frequency Distribution.

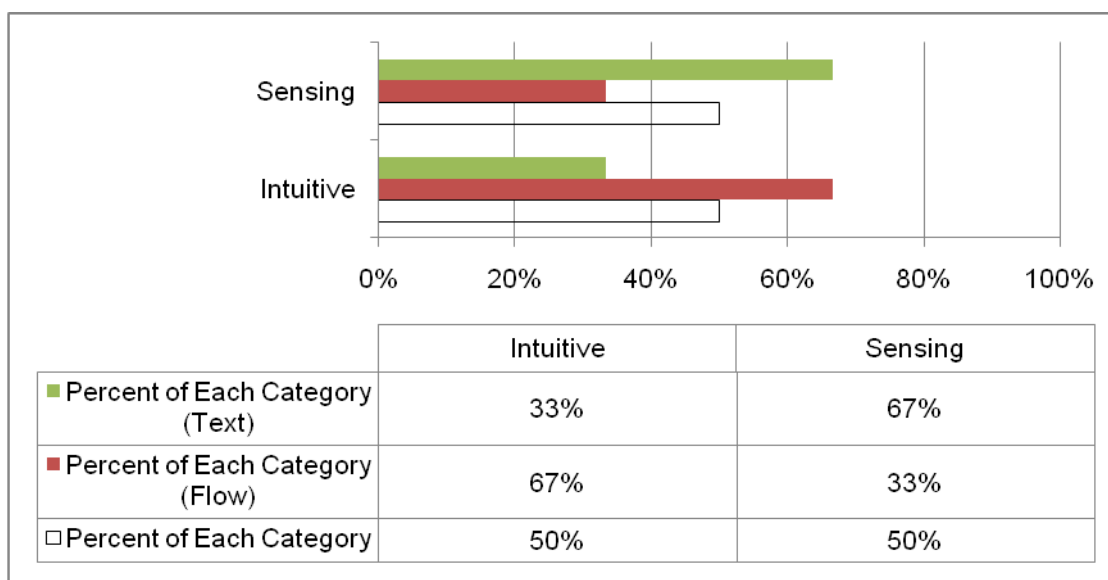


Figure 4.25 Second Pilot Percent of Sensing or Intuitive Students.

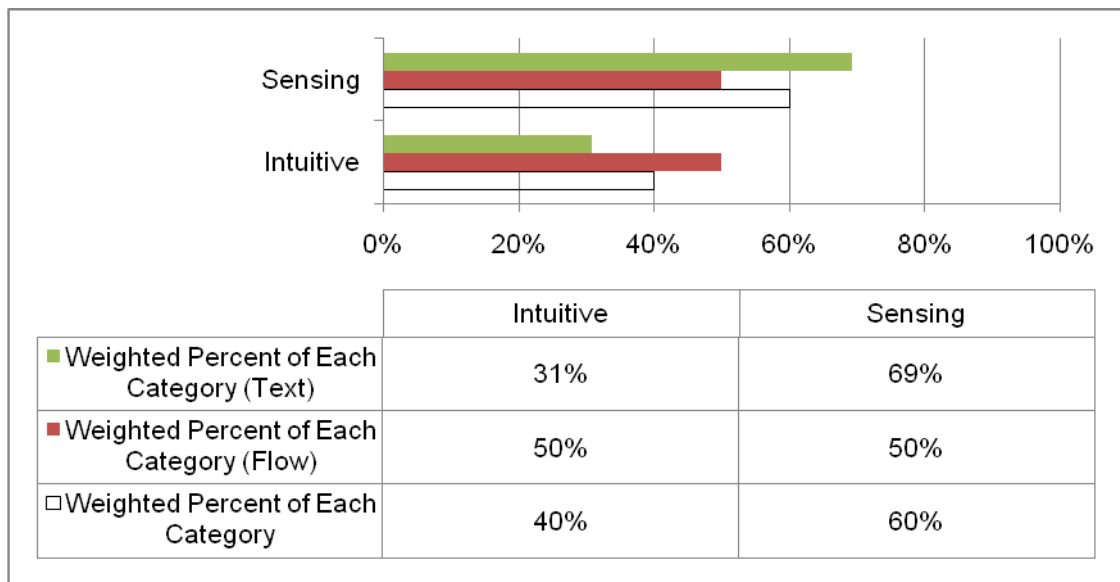


Figure 4.26 Second Pilot Weighted Percent of Sensing or Intuitive Scores.

The students' Visual or Verbal distribution is presented in Figure 4.27, Figure 4.28, and Figure 4.29.

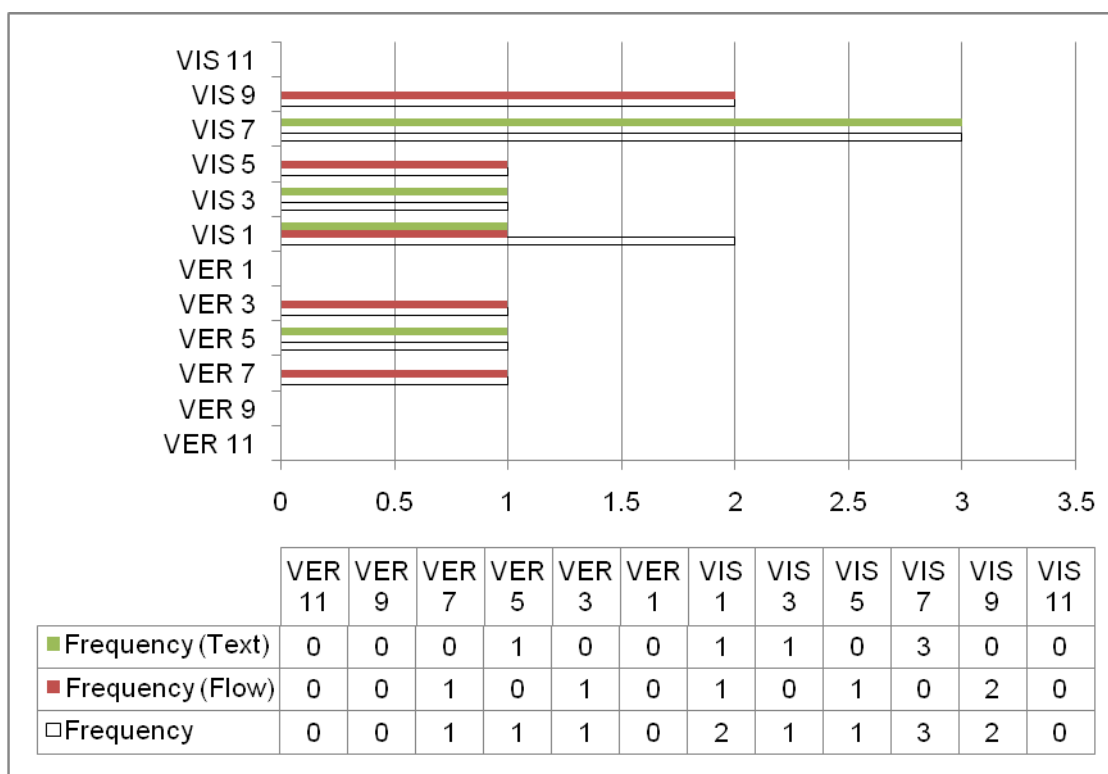


Figure 4.27 Second Pilot Visual or Verbal Frequency Distribution.

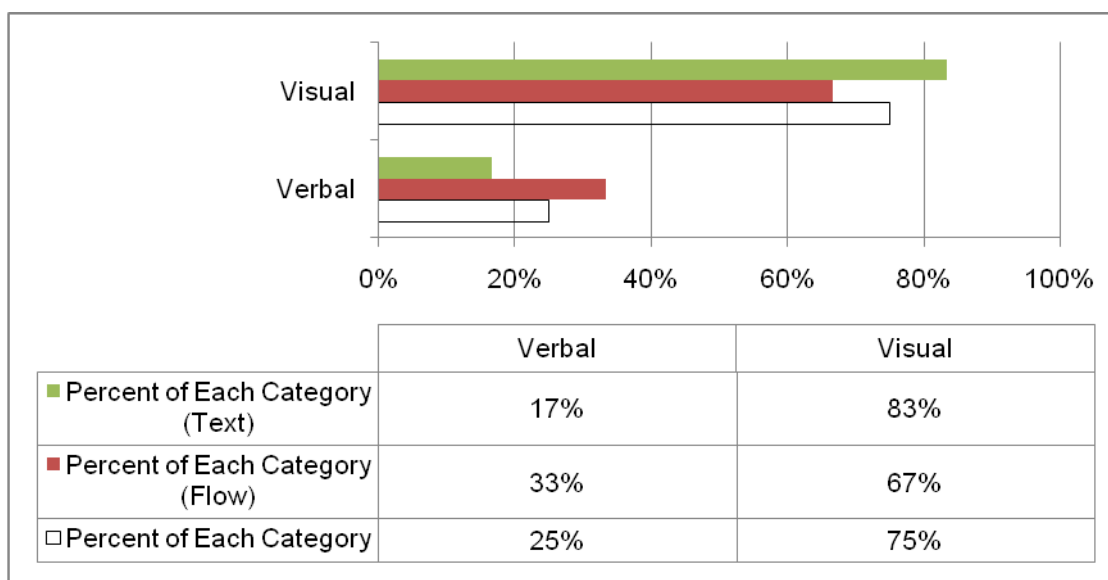


Figure 4.28 Second Pilot Percent of Visual or Verbal Students.

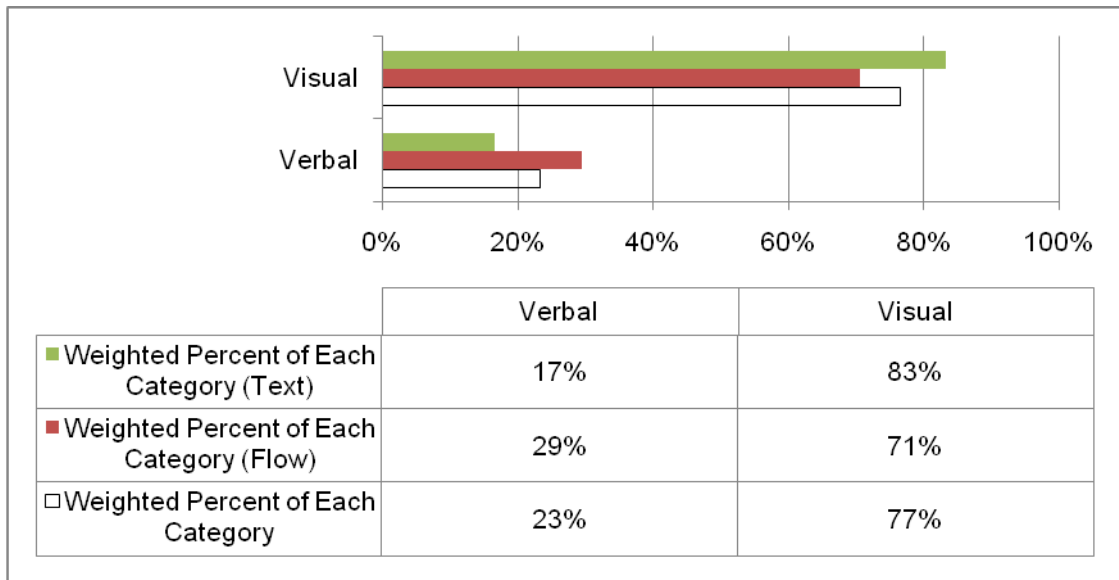


Figure 4.29 Second Pilot Weighted Percent of Visual or Verbal Scores.

The students' Sequential or Global distribution is presented in Figure 4.30, Figure 4.31, and Figure 4.32.

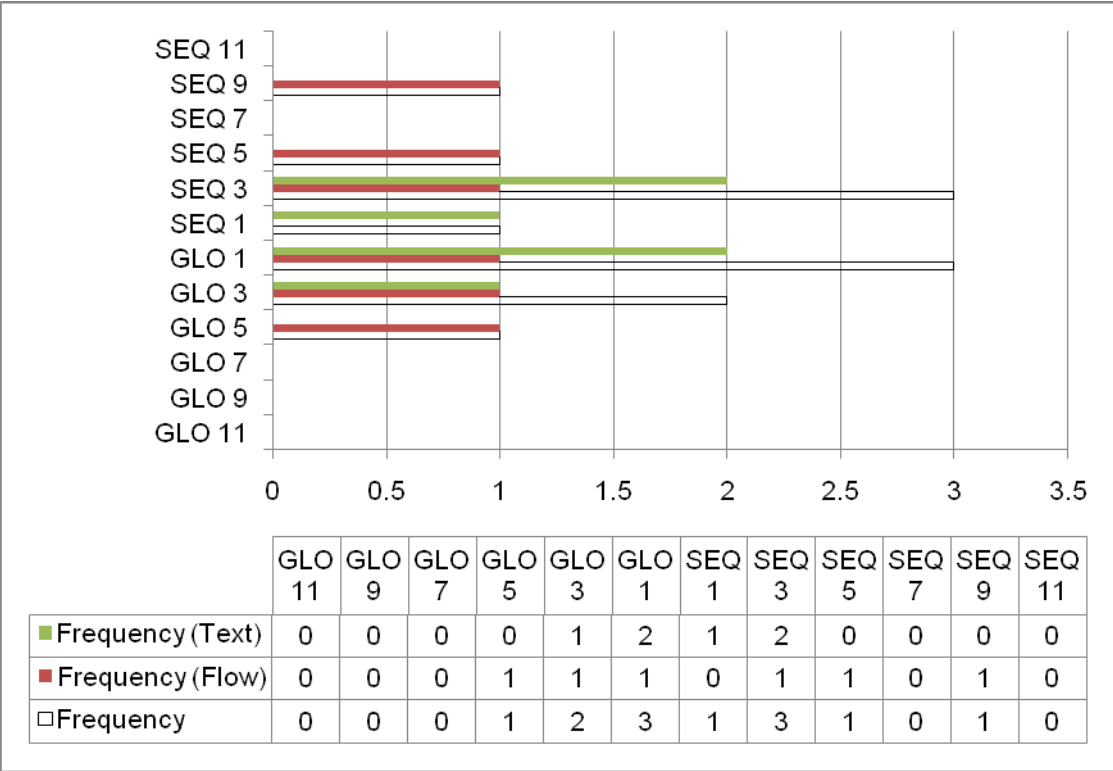


Figure 4.30 Second Pilot Sequential or Global Frequency Distribution.

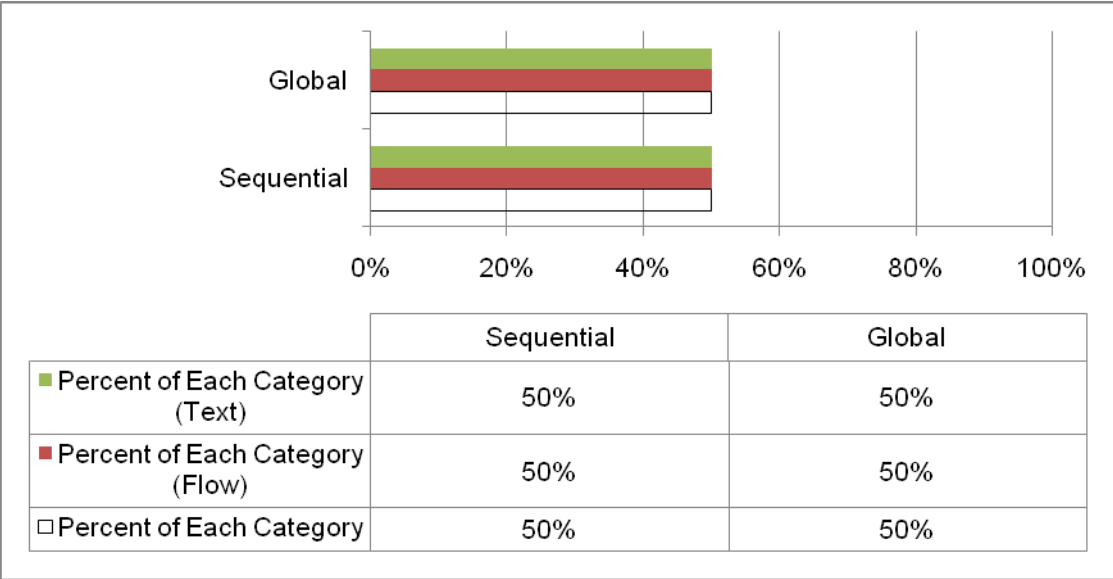


Figure 4.31 Second Pilot Percent of Global or Sequential Students.

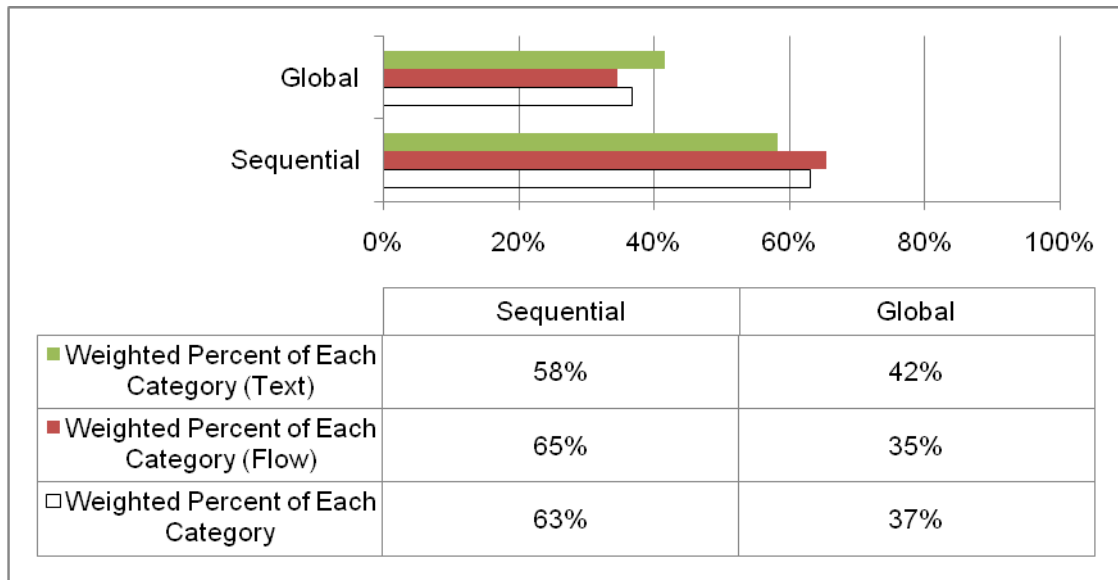


Figure 4.32 Second Pilot Weighted Percent of Global or Sequential Scores.

4.2.2 Group Statistics and Independent Samples Test

The Second Pilot's Test Score statistics and Independent Samples Test are presented in Table 4.2.

Table 4.2 Second Pilot Test Scores Statistics and Independent Samples Test.

Flow or Text Treatment	n	Mean	Std. Deviation	Std. Error Mean	t	df	Sig. (2-tailed)
Flow	6	46.33	24.59	10.04	-.31	10	.77
Text	6	50.00	16.13	6.58			

4.2.3 Internal Reliability using Cronbach's Alpha

The Second Pilot's Cronbach's Alpha is presented in Table 4.3 and the Second Pilot's Cronbach's Alpha with Questions 2 and 8 removed is presented in Table 4.4. We note that a Cronbach's Alpha of greater than .70 is sought for this work.

Table 4.3 The Second Pilot's Entire Test's Reliability Statistics.

Cronbach's Alpha	Number of Items
.789	10

Table 4.4 The Second Pilot's Test's Reliability Statistics, Removed Questions 2 and 8.

Cronbach's Alpha	Number of Items
.811	8

4.3 The Experiment

Sections 4.3.1 and 4.3.2 describe the statistics of the actual experiment. The environment was a class room with a projector with a computer attached to it that was used to present the slides to the students. Ninety (90) students were in the control group, and eighty-two (82) students were in the test group. The experiment took place during twenty one-hour time periods during the second and third week of the fall 2010 semester. Finally, Section 4.3.3 describes Cronbach's Alpha used for measuring the reliability of the test.

4.3.1 The Frequency Distribution

The students' age distribution is presented in Figure 4.33.

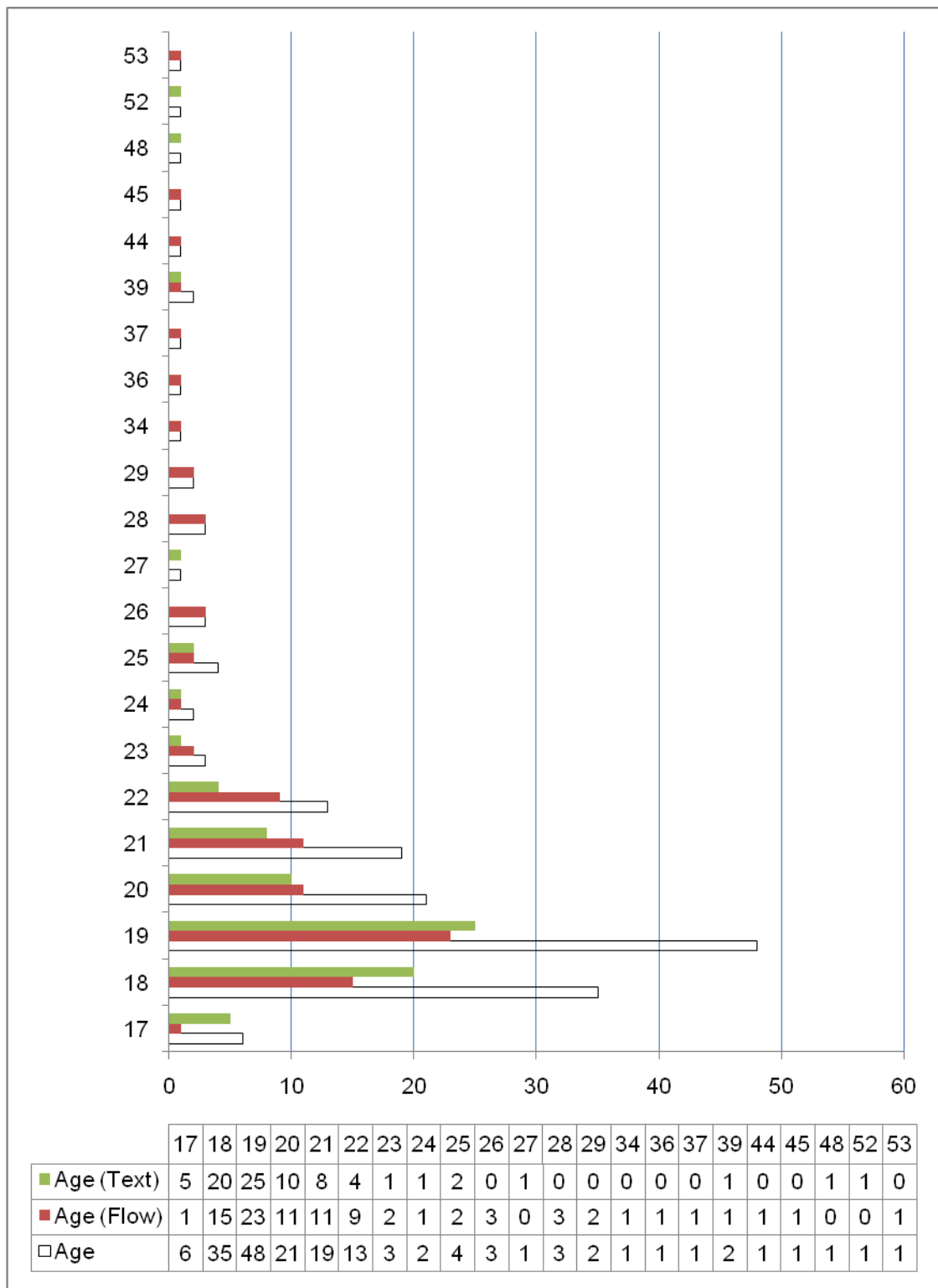


Figure 4.33 The Experiment's Age Distribution.

The students' gender distribution is presented in Figure 4.34.

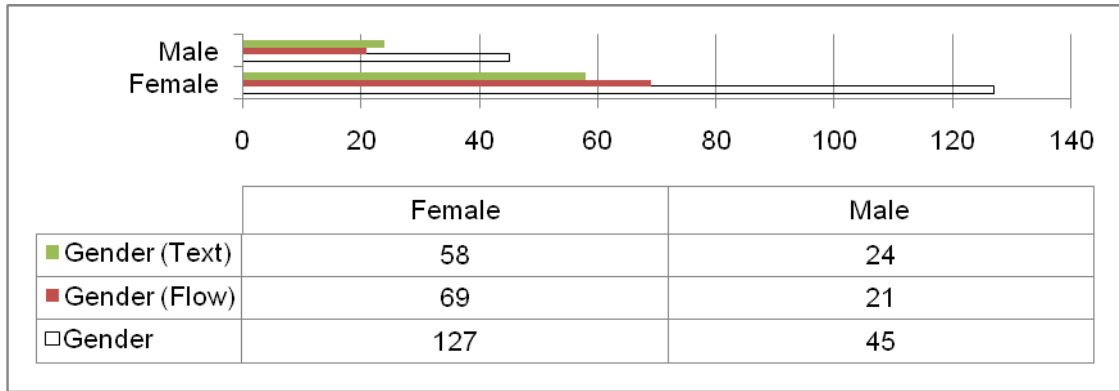


Figure 4.34 The Experiment's Gender Distribution.

The students' class level distribution is presented in Figure 4.35.

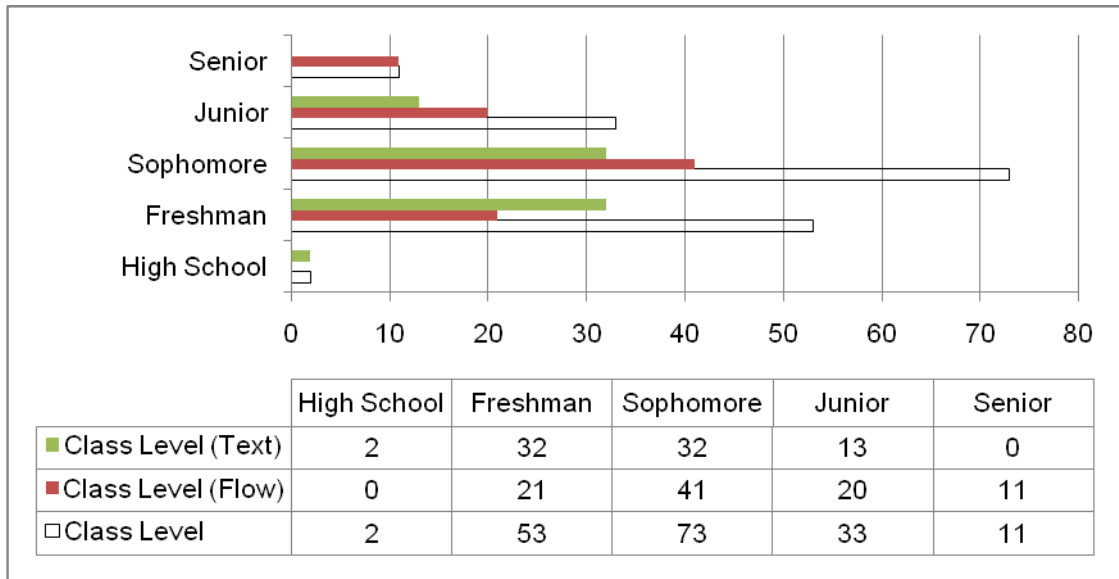


Figure 4.35 The Experiment's Class Level Distribution.

The students' GPA Range distribution is presented in Figure 4.36.

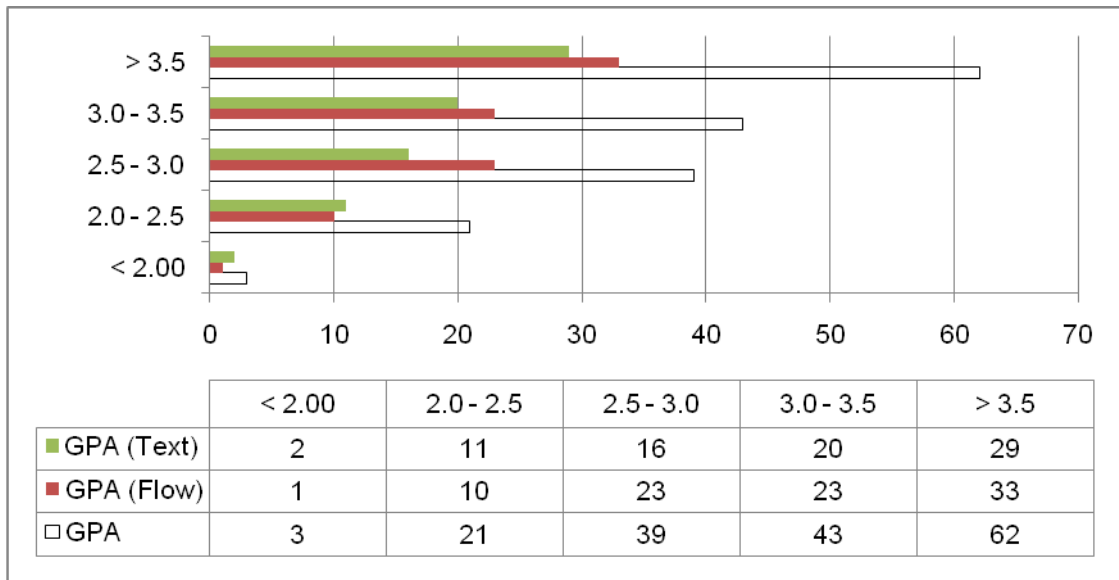


Figure 4.36 The Experiment's GPA Distribution.

The students' Left and Right Handedness distribution is presented in Figure 4.37.

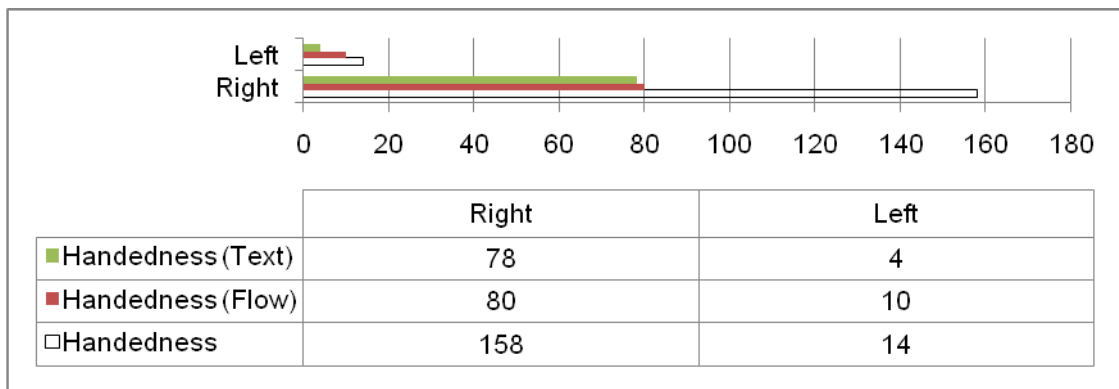


Figure 4.37 The Experiment's Right and Left Handedness.

The students' Test Time distribution is presented in Figure 4.38.

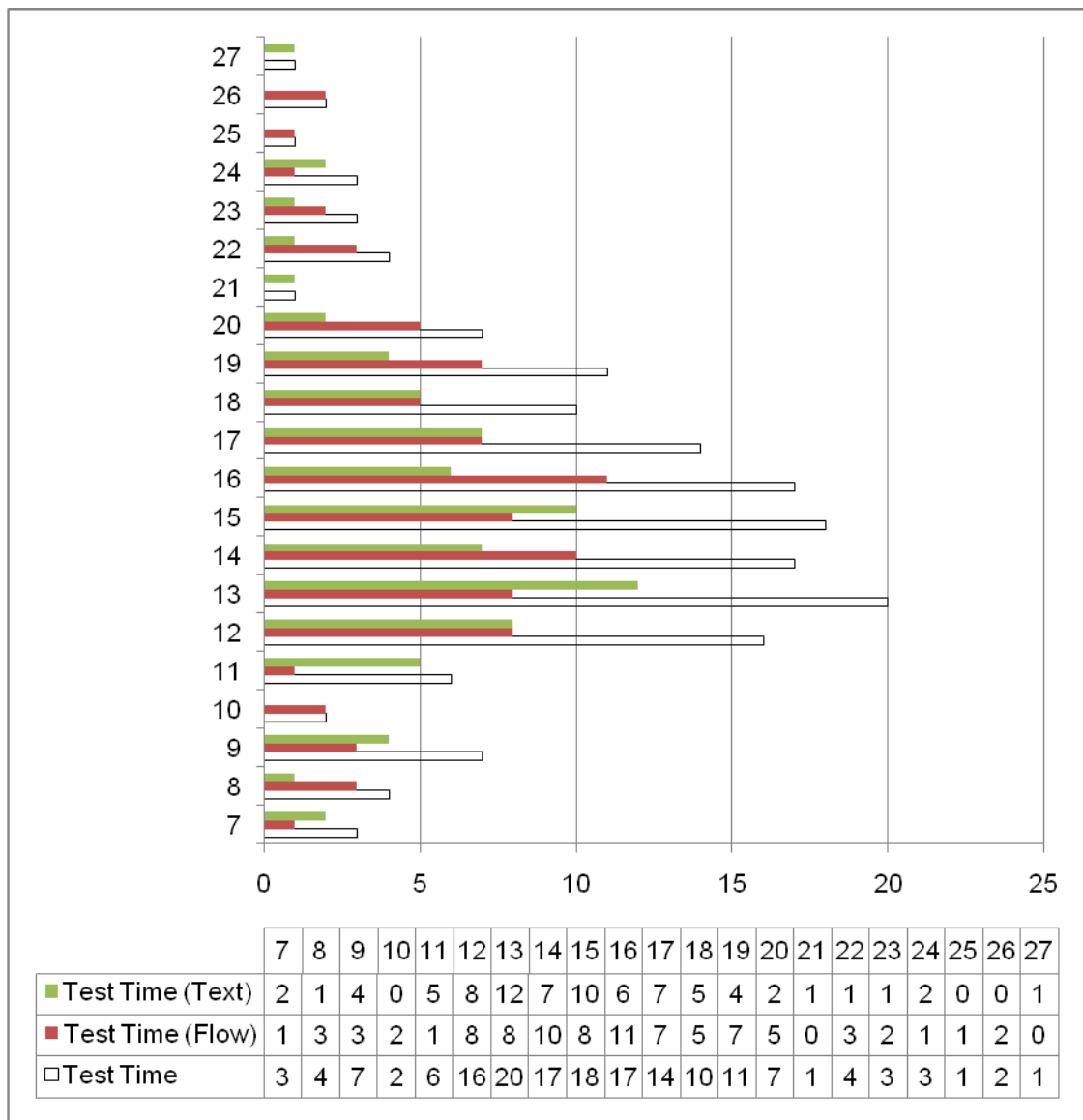


Figure 4.38 The Experiment's Test Time Distribution.

The students' Active or Reflective distribution is presented in Figure 4.39, Figure 4.40, and Figure 4.41.

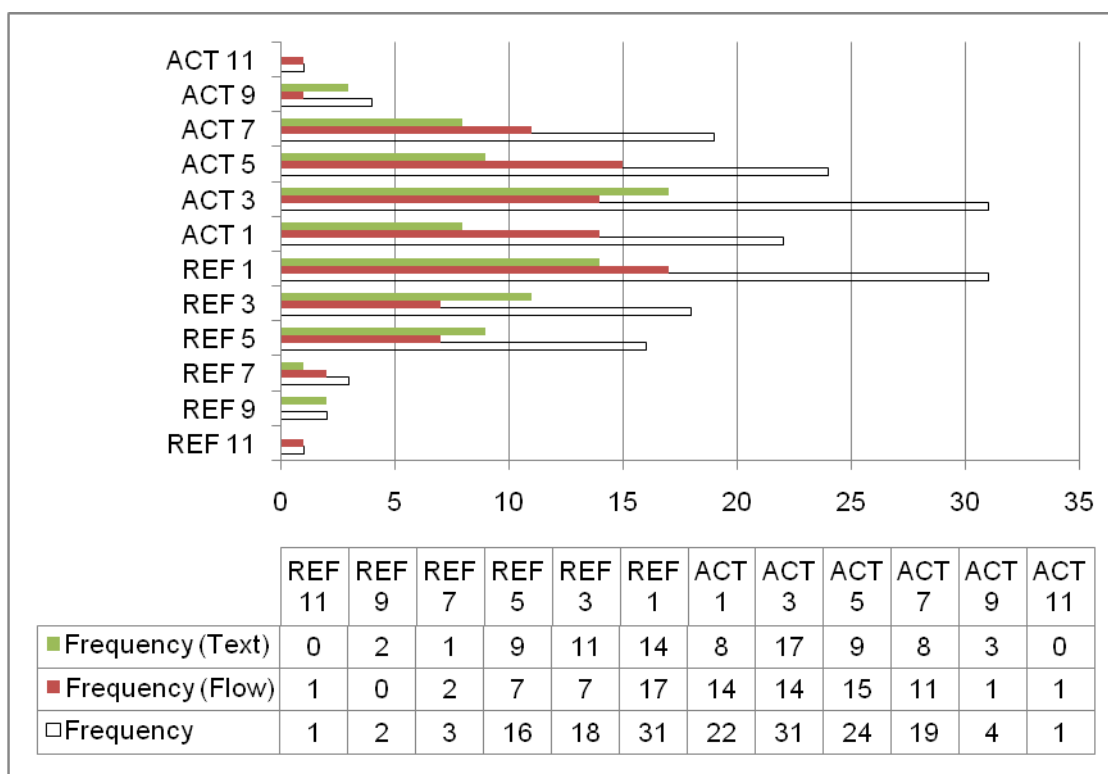


Figure 4.39 The Experiment's Active or Reflective Frequency Distribution.

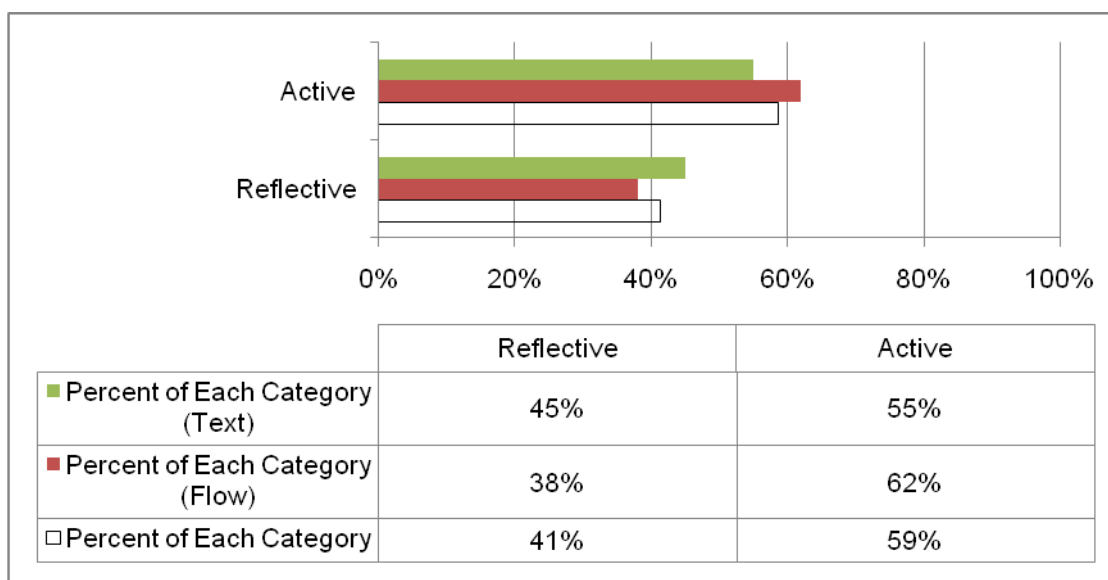


Figure 4.40 The Experiment's Percent of Active or Reflective Students.

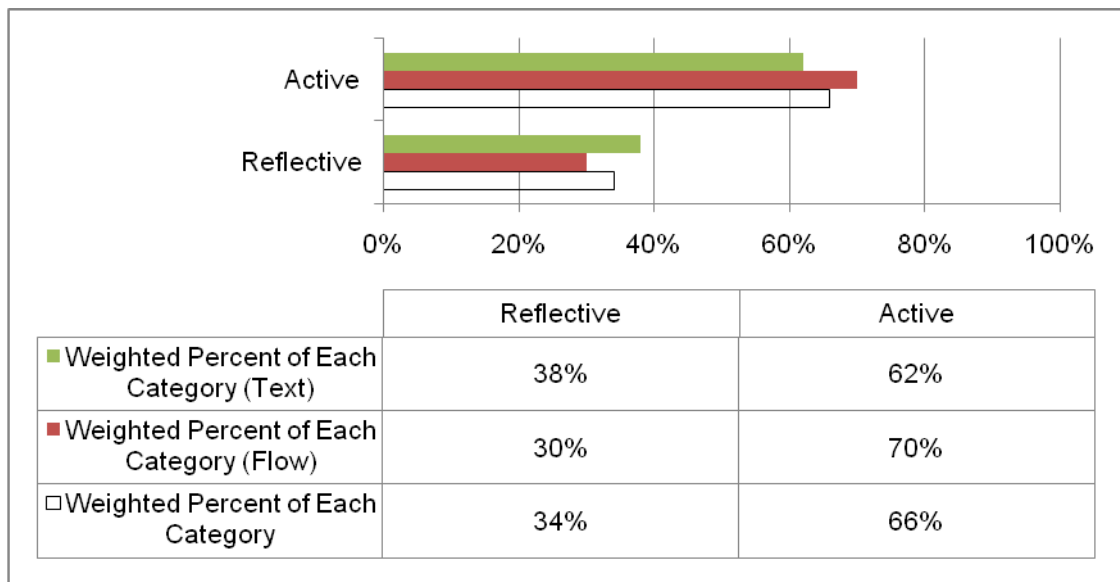


Figure 4.41 The Experiment's Weighted Percent of Active or Reflective Scores.

The students' Sensing or Intuitive distribution is presented in Figure 4.42, Figure 4.43, and Figure 4.44.

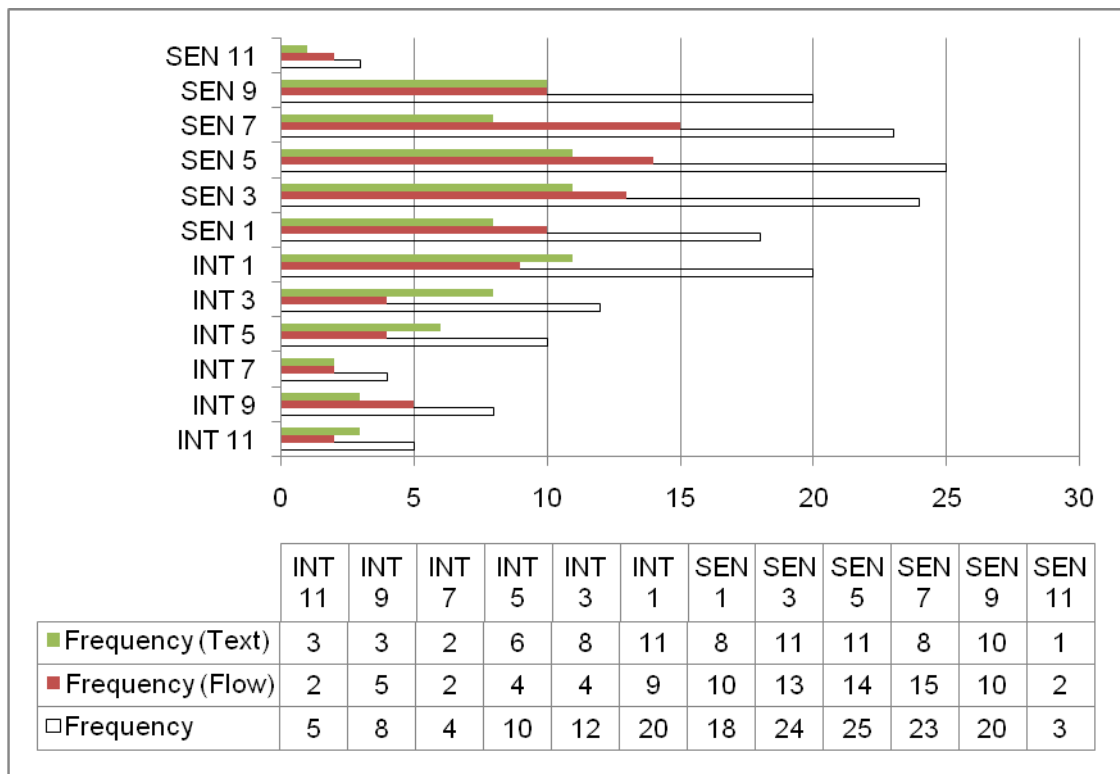


Figure 4.42 The Experiment's Sensing or Intuitive Frequency Distribution.

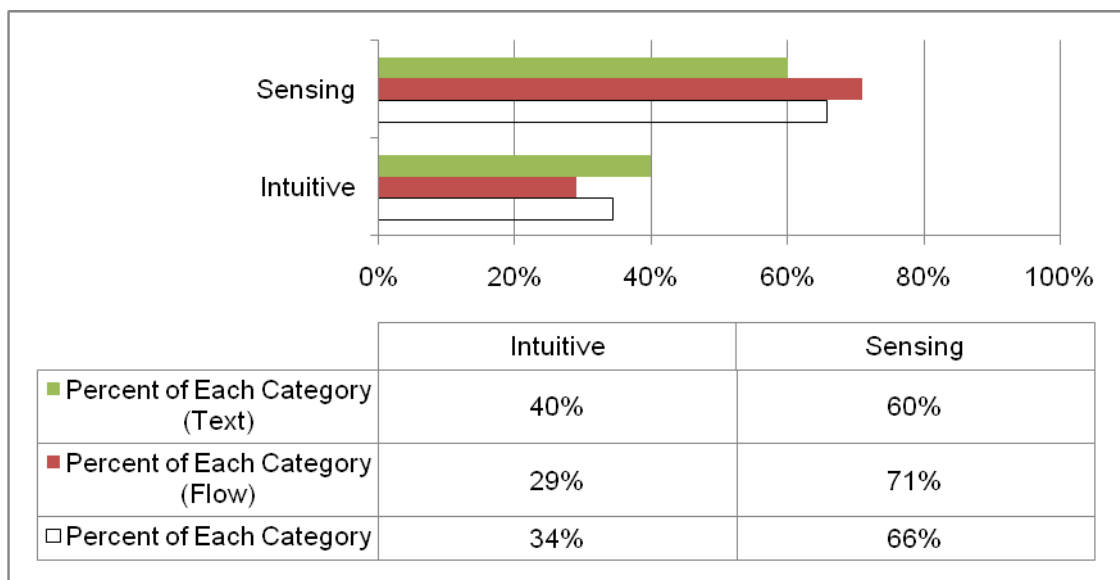


Figure 4.43 The Experiment's Percent of Sensing or Intuitive Students.

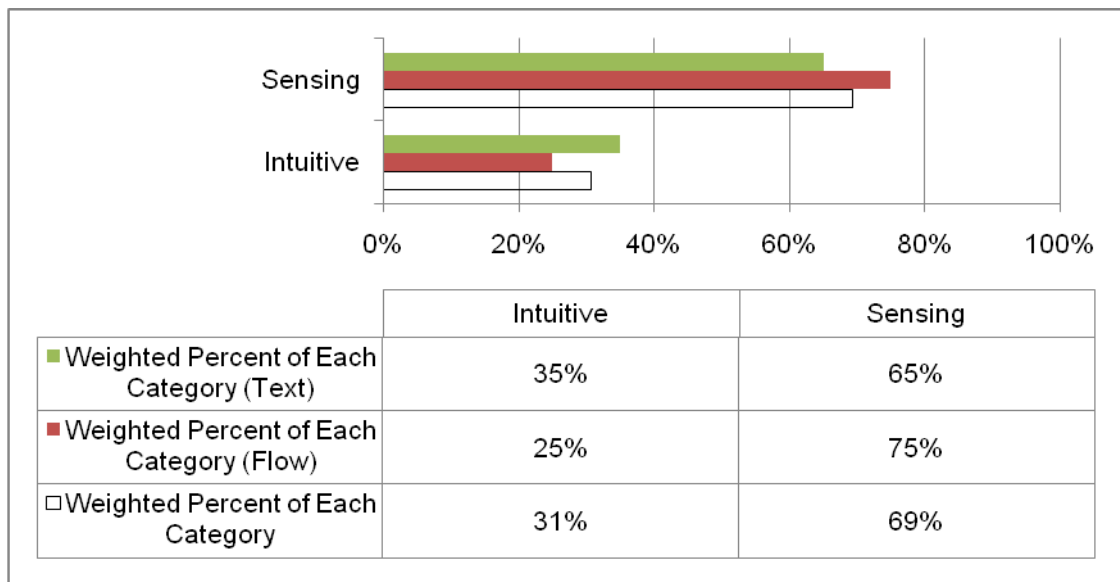


Figure 4.44 The Experiment's Weighted Percent of Sensing or Intuitive Scores.

The students' Visual or Verbal distribution is presented in Figure 4.45, Figure 4.46, and Figure 4.47.

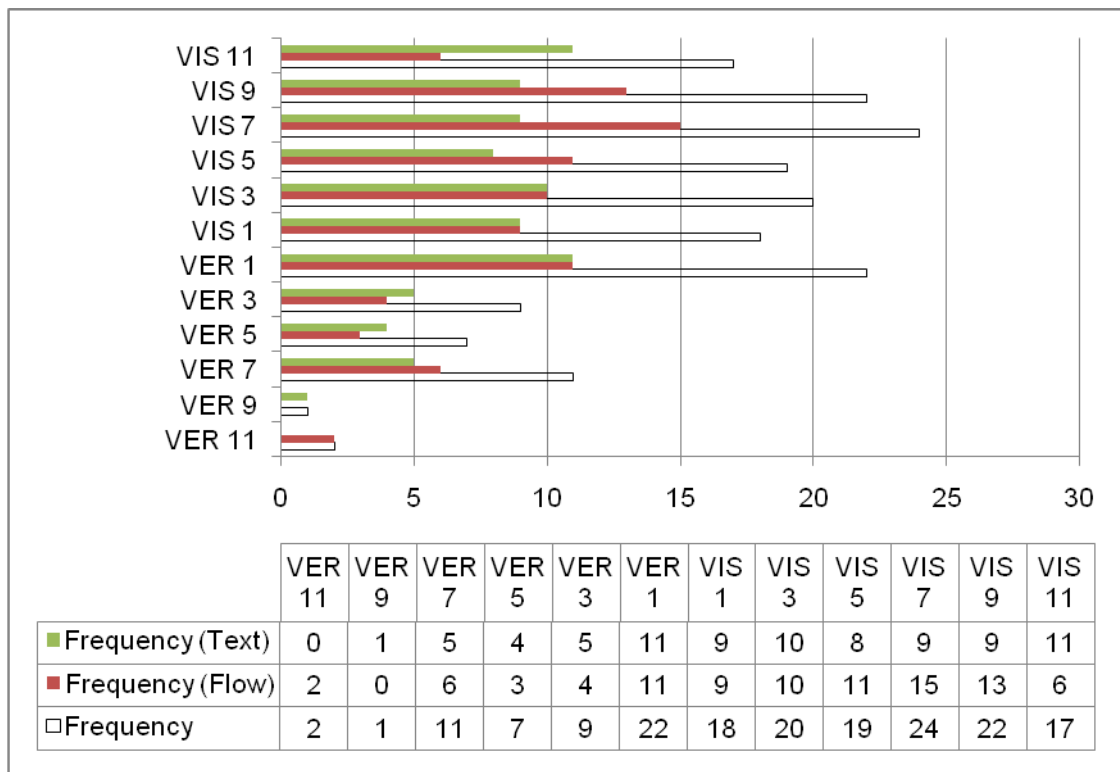


Figure 4.45 The Experiment's Visual or Verbal Frequency Distribution.

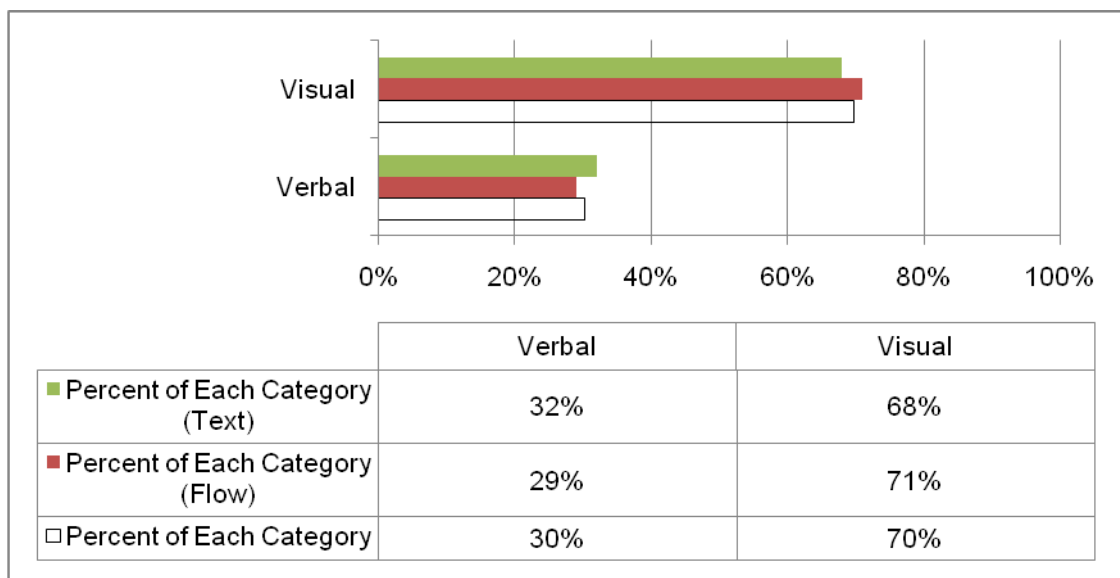


Figure 4.46 The Experiment's Percent of Visual or Verbal Students.

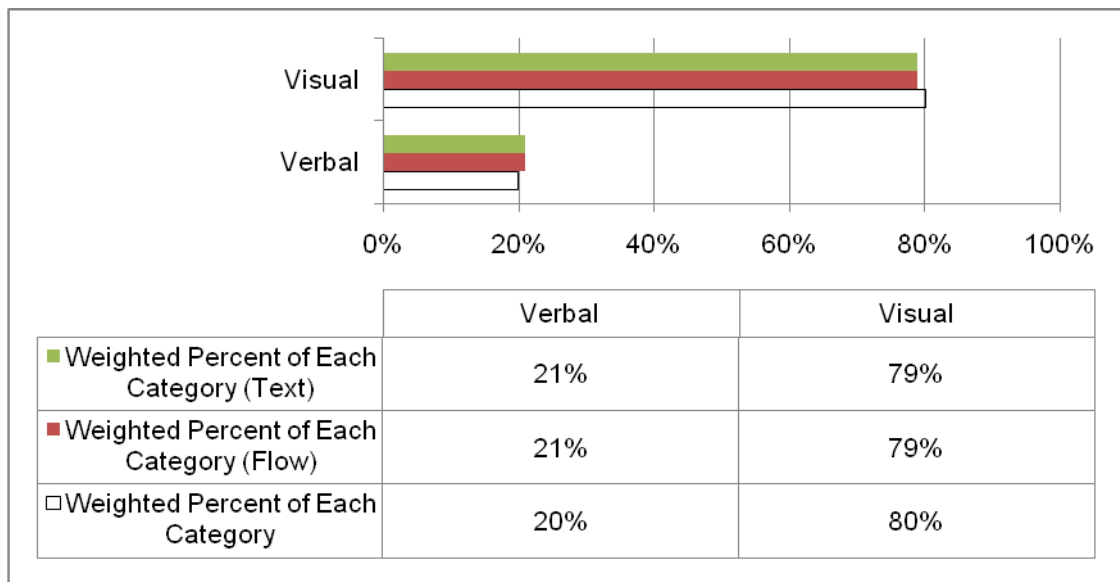


Figure 4.47 The Experiment's Weighted Percent of Visual or Verbal Scores.

The students' Sequential or Global distribution is presented in Figure 4.48, Figure 4.49, and Figure 4.50.

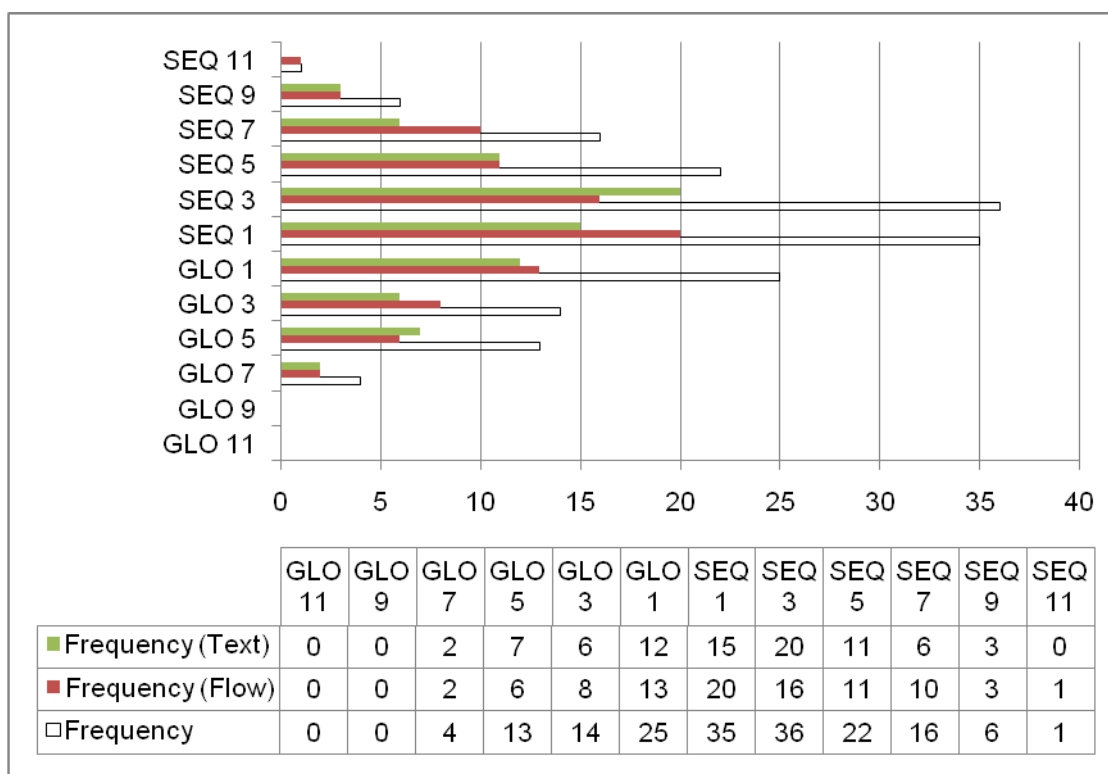


Figure 4.48 The Experiment's Sequential or Global Frequency Distribution.

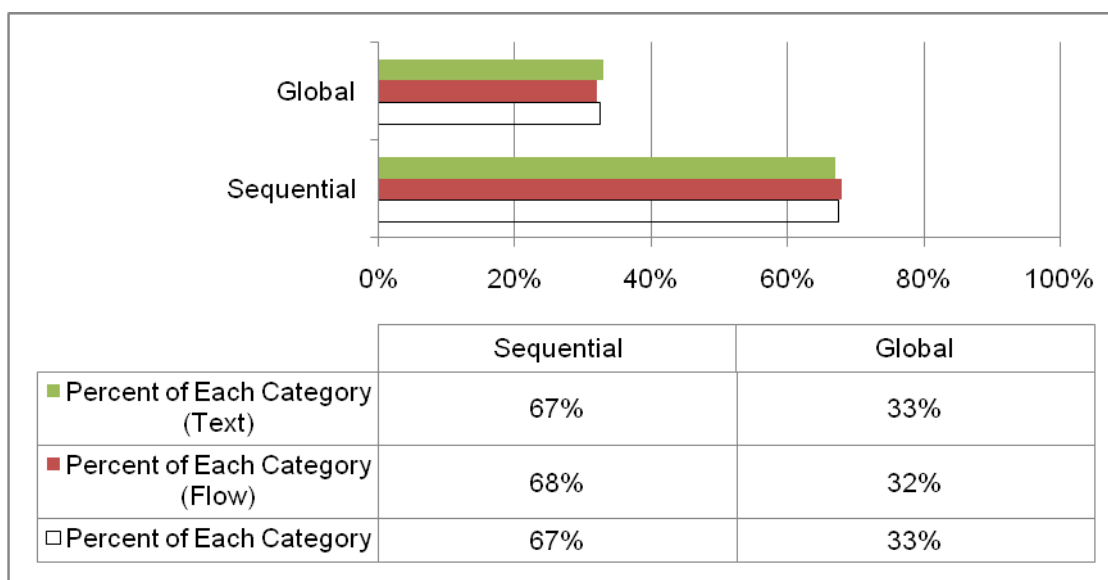


Figure 4.49 The Experiment's Percent of Global or Sequential Students.

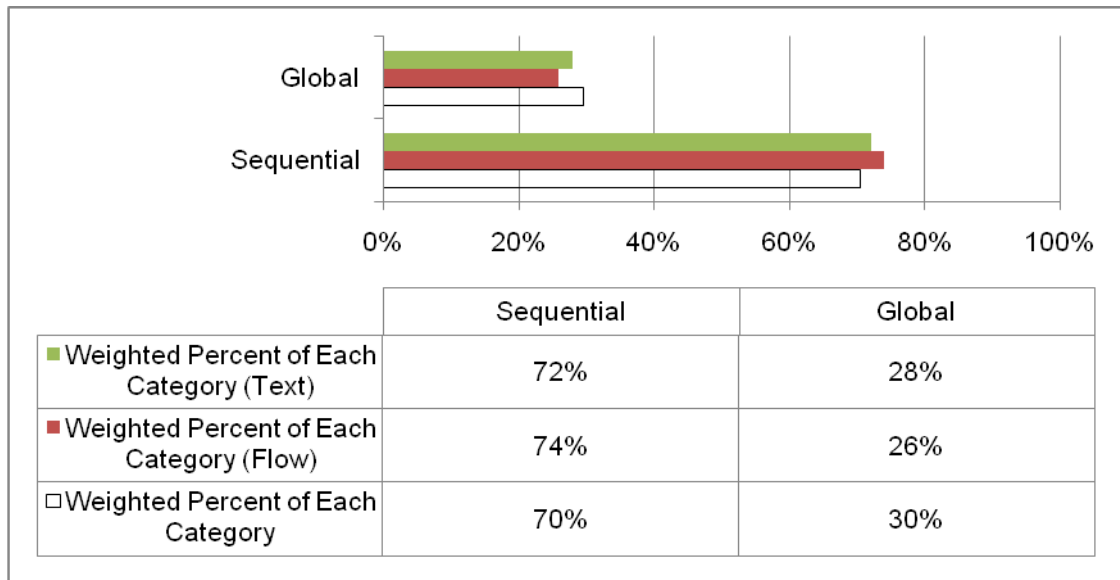


Figure 4.50 The Experiment's Weighted Percent of Global or Sequential Scores.

4.3.2 Overall Group Statistics and Independent Samples Test

The Experiment's statistics and the Experiment's Independent Samples Test, grouped by comparing Flow and Text, are presented in Table 4.5 and the Experiment's Independent Samples Test, comparing the Computer Science concepts of Sequence, Alternative, and Iteration, grouped by Flow and Text, are presented in Table 4.6.

Table 4.5 The Experiment's Statistics and the Experiment's Independent Samples Test, grouped by Flow and Text.

	Flow or Text	n	Mean	Std. Deviation	Std. Error Mean	t	df	Sig (2-tailed)
Test Grade	Flow	90	2.73	.93	.10	2.39	170	.02*
	Text	82	2.39	.89	.10			
Class Level	Flow	90	2.17	.89	.09	2.78	170	< .01*
	Text	82	1.79	.87	.10			
Age	Flow	89	22.17	6.39	.67	1.72	170	.09
	Text	81	20.58	5.59	.62			
GPA	Flow	90	3.17	.57	.06	.38	166	.71
	Text	78	3.13	.64	.07			
Test Time	Flow	90	15.68	4.12	.44	1.30	165	.20
	Text	79	14.87	3.88	.44			
Active/Reflective	Flow	90	-1.44	4.16	.44	-.91	170	.36
	Text	82	-.85	4.32	.48			
Sensing/Intuitive	Flow	90	-2.38	5.45	.58	-1.18	170	.24
	Text	82	-1.39	5.54	.61			
Visual/Verbal	Flow	90	-3.11	5.48	.58	-.02	170	.99
	Text	82	-3.10	5.55	.61			
Sequential/Global	Flow	90	-1.67	3.92	.41	-.34	170	.73
	Text	82	-1.46	3.80	.42			
Question 1	Flow	90	3.50	2.30	.24	.07	170	.95
	Text	82	3.48	2.32	.26			
Question 2	Flow	90	.06	.53	.06	-1.11	170	.27
	Text	82	.18	.94	.10			
Question 3	Flow	90	1.89	2.44	.26	-.00	170	1.00
	Text	82	1.89	2.44	.27			
Question 4	Flow	90	3.28	2.39	.25	.78	170	.44
	Text	82	2.99	2.47	.27			
Question 5	Flow	90	2.78	2.50	.26	.09	170	.93
	Text	82	2.74	2.50	.28			
Question 6	Flow	90	2.89	2.48	.26	1.66	170	.10
	Text	82	2.26	2.50	.28			
Question 7	Flow	90	2.17	2.49	.26	-.71	170	.48
	Text	82	2.44	2.51	.28			
Question 8	Flow	90	2.92	1.31	.14	.84	170	.40
	Text	82	2.74	1.40	.15			
Question 9	Flow	90	3.06	1.29	.14	.30	170	.77
	Text	82	2.99	1.69	.19			
Question 10	Flow	90	2.28	1.48	.16	3.63	170	< .01*
	Text	82	1.43	1.57	.17			
Question 11	Flow	90	1.83	2.42	.26	.18	170	.86
	Text	82	1.77	2.41	.27			
Question 12	Flow	90	2.78	2.50	.26	.57	170	.57
	Text	82	2.56	2.51	.28			
Question 13	Flow	90	2.06	2.47	.26	-1.98	170	.05
	Text	82	2.81	2.50	.28			
Question 14	Flow	90	1.83	2.42	.26	1.03	170	.31
	Text	82	1.46	2.29	.25			
Question 15	Flow	90	3.11	2.44	.26	-1.18	170	.24
	Text	82	3.54	2.29	.25			
Question 16	Flow	90	4.22	1.82	.19	-.40	170	.69
	Text	82	4.33	1.71	.19			
Question 17	Flow	90	4.06	1.97	.21	6.03	170	< .01*
	Text	82	2.01	2.47	.27			
Question 18	Flow	90	3.43	2.32	.24	-.47	170	.64
	Text	82	3.60	2.26	.25			
Question 19	Flow	90	3.44	2.33	.24	4.83	170	< .01*
	Text	82	1.71	2.39	.26			
Question 20	Flow	90	2.99	2.46	.26	5.89	170	< .01*
	Text	82	.98	1.99	.22			

Table 4.6 The Experiment's Independent Samples Test, comparing the Computer Science concepts of Sequence, Alternative, and Iteration, grouped by Flow and Text.

	Flow or Text	n	Mean	Std. Deviation	Std. Error Mean	t	df	Sig (2-tailed)
Sequence (8, 9, 16)	Flow	90	3.40	1.08	.11	.26	170	.795
	Text	82	3.35	1.16	.13			
Alternative (1, 6, 7, 8, 9, 12, 13, 14, 17, 19, 19, 20)	Flow	90	2.88	1.02	.11	3.54	170	.001
	Text	82	2.31	1.09	.12			
Iteration (3, 7, 11, 12, 13, 14)	Flow	90	2.09	1.51	.16	-.29	170	.772
	Text	82	2.15	1.26	.14			

4.3.3 Internal Reliability using Cronbach's Alpha

We note that a Cronbach's Alpha of greater than .70 is sought for this work.

Table 4.7 The Experiment's Test's Reliability Statistics.

Cronbach's Alpha	Number of Items
.749	20

4.4 Summary

We have presented the raw results that we have found in our two pilot studies and the final experiment. We presented some results and found that Visual/Verbal scores do not indicate performance in the Flow or Text Learning Style group. Note that not all data were used in correlations, and certainly not all permutations were examined. We collected the participants Major and Math Classes, but we did not use them in our data analysis, as there were too many different Majors and too many different Math classes, but we leave that for future work. We will discuss the results that we have found and further analyze the data that we have collected in the next chapter.

CHAPTER V

CORRELATIONS AND COMPARISONS FOUND IN THE FINAL EXPERIMENT

In this section, we analyze the results that we collected in the previous chapter in search of testing the hypothesis that *computer science novices understand computer science concepts (here sequence, iteration, alternative) better, and test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. We define a proper test of this hypothesis as an environment where there exists two groups, each of which has at least twenty students (all of whom have taken the ILS Questionnaire), where one group uses a Flow-based visual programming language (here, RAPTOR) as the teaching style to aid program understanding, while the other group does not. Our null hypothesis is that *computer science novices will understand computer science concepts (here sequence, iteration, alternative) the same, and will not test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. We will review the data, paying particular attention to the correlation between visual and verbal learning styles and the impact that a Flow-based visual programming language has on their understanding measured by comparing the test grade of both groups.

We recall that the Pearson correlation coefficient is a measure of the correlation (linear dependence) between two variables, and is a value between 1 and -1. If the Pearson correlation coefficient is positive, then it means that as one of the variables increases in size, the other variable increases in size. If the Pearson correlation coefficient is negative, then it means that as one of the variables increases in size, the other variable decreases in size. We only consider $p < .05$.

We recall that the Flow group was exposed to the Flow Teaching Style, the Text group was exposed to the Text Teaching Style, and the Overall group contains participants from the Flow group and the Text group. We perform the data collection activities in Section 5.1 to test our hypothesis, Section 5.2 to explore other hypotheses (such as the effect of Gender and Time of Day on the Test Grade), and Section 5.3 as a validation of the ILS (where we will compare the four Index of Learning Style dimensions against, Ages, Class Level, GPA Range, and Test Time). We will perform Correlations and Regression Analyses with these variables to determine their effect on the Test Grade in the Section 5.2.

5.1 Comparisons to Measure our Hypothesis

In this section, we will discuss the data collected to test our hypothesis. When we considered the Flow Teaching Style and Text Teaching Style and compared the Test Grade, our results show a significant difference with $t(170) = 2.39$, $p = .018$ and the Flow Mean Test Grade is 2.73, while the Text Mean Test Grade is 2.39. When we considered the Flow Teaching Style and Text Teaching Style and compared the Class Level, our results show a significant difference with $t(170) = 2.78$, $p = .006$ and the Flow Mean

Class Level is 2.17, while the Text Mean Class Level is 1.79. When we considered the Flow Teaching Style and Text Teaching Style and compared the questions on the Test that tested the concept of alternative, our results show a significant difference with $t(170) = 3.54$, $p = .001$ and the Flow Mean Alternative Grade is 2.88, while the Text Mean Alternative Grade is 2.31.

Table 5.1 Comparing General Demographics considering Teaching Style.

	Flow or Text	n	Mean	Std. Deviation	Std. Error Mean	t	df	Sig (2-tailed)
Test Grade	Flow	90	2.73	.93	.10	2.39	170	.02*
	Text	82	2.39	.89	.10			
Class Level	Flow	90	2.17	.89	.09	2.78	170	< .01*
	Text	82	1.79	.87	.10			
Alternative (1, 6, 7, 8, 9, 12, 13, 14, 17, 18, 19, 20)	Flow	90	2.88	1.02	.11	3.54	170	.001
	Text	82	2.31	1.09	.12			

Unfortunately, this means that Class Level is a confounding variable, and we must measure its effect. We performed an Analysis of Variance Test using a Univariate General Linear Model, and found that the exposure to Flow or Text had a greater effect on Test Grade than the Class Standing did, as shown in Table 5.2.

Table 5.2 The ANCOVA to determine the effects of Class Level and Flow or Text exposure on Test Grade.

	df	F	Sig
Corrected Model	8	1.770	.032
Intercept	1	393.085	.000
Flow or Text	1	1.374	.196
Class Standing	4	1.816	.068
Flow or Text * Class Standing	3	.924	.337
Error	163	.814	
Total	172		
Corrected Total	171		

5.2 Correlations and Comparisons Involving Test Grade

In this section, we will discuss the results that we have found within our Overall group, our Flow group, and our Text group regarding their Test Grade scores. We will attempt to correlate the four Index of Learning Style dimensions, the Ages, Class Level, GPA Range, Test Time, Gender, and Time of Day with the Test Grade scores. We will present the measures that correlate (including results of Regression Analyses) in the next numbered sections, and summarize the measures that did not correlate at the end of each section.

We recall that when a Linear Regression is performed, the resulting independent variable values (or coefficients) are used to predict the dependent variable. We will present our coefficients as standardized coefficients (so that their variances are 1) to allow the reader to more easily compare these values to each other, even when they are of different unit scales.

We perform a set of statistical tests on each for each of our variables. The format for these statistical tests is shown in Table 5.3. In this example, we are correlating our Test Grade with different sets of variables that include Visual/Verbal scores. Since our data did not show significance for many of our variables, we will not include entire rows that are insignificant, and we will use the abbreviation N.S. for cells in the table that are not significant. Please note that each of these statistical tests was performed for Sections 5.2.1 through 5.2.12, even though the results are not explicitly shown. Note that Sections 5.2.13 through 5.2.15 are the same as Table 5.3 except that the Index of Learning Style dimensions are not included and that Group 1 and Group 2 represent the

groups being compared, such as Males and Females, Morning and Afternoon, or Text and Flow.

Table 5.3 A Sample Format for our Statistical Tests.

	Group 1		Group 2	
	p	Pearson Coefficient	p	Pearson Coefficient
Visual/Verbal	N.S. ²			
Visual/Verbal, Age				
Visual/Verbal, Age, GPA Range				
Visual/Verbal, Age, GPA Range, Test Time				
Visual/Verbal, Age, GPA Range, Class Level				
Visual/Verbal, Age, GPA Range, Class Level, Test Time				
Visual/Verbal, Test Time				
Visual/Verbal, GPA Range				
Visual/Verbal, GPA Range, Test Time				
Visual/Verbal, Class Level				
Visual/Verbal, Class Level, Test Time				
Visual/Verbal, Class Level, Age				
Visual/Verbal, Class Level, Age, Test Time				
Visual/Verbal, Class Level, GPA Range				
Visual/Verbal, Class Level, GPA Range, Test Time				

5.2.1 Correlations and Comparisons Involving the Visual/Verbal Style

When we considered the Overall group, Flow group and Text group and correlated Test Grade and Visual/Verbal scores, our results did not show any correlation. We then performed a linear regression by adding additional variables as Independent variables to our Regression, and our results did not show any correlation.

When we subdivided the Flow group and Text group by their Visual/Verbal score (with a midpoint at 0 and at Visual = 3) and compared their Test Grades, we did not find any acceptable evidence, as defined in [25].

² Not Significant.

5.2.2 Correlations Involving the Visual/Verbal Considering Style Gender

When we considered Males in the Flow group and performed a linear regression with Test Grade as the Dependent Variable and Visual/Verbal scores and GPA Range as Independent Variables, our results did show a positive correlation with Visual/Verbal values of $p = .045$ and Beta = .436 and GPA Range values of $p = .046$ and Beta = .434.

We then performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.4, and our results did not show any correlation.

Table 5.4 Test Grade Correlations Involving Visual/Verbal scores considering Gender.

	Females		Males	
	p	Pearson Coefficient	p	Pearson Coefficient
Visual/Verbal		N.S.	.045	.436
Age			.046	.434

5.2.3 Correlations Involving the Visual/Verbal Style Considering Time of Day

When we considered Morning groups and Afternoon groups and correlated Test Grade with Visual/Verbal scores, our results did not show any correlation. We then performed a linear regression by adding variable as Independent variables to our Regression, and our results did not show any correlation.

5.2.4 Correlations and Comparisons Involving the Sequential/Global Style

When we considered the Overall group, the Text group, and the Flow group and correlated Test Grade and Sequential/Global scores, our results did not show any

correlation. We then performed a linear regression by adding additional variables as Independent variables to our Regression, and our results did not show any correlation.

When we subdivided the Flow group and Text group by their Sequential/Global score (with a midpoint at 0 and at Sequential = 1) and compared their Test Grades, we did not find any acceptable evidence, as defined in [25].

5.2.5 Correlations Involving the Sequential/Global Style considering Gender

When we considered Males and Females and correlated Test Grade and Sequential/Global scores, our results did not show any correlation. We then performed a linear regression by adding additional variables as Independent variables to our Regression, and our results did not show any correlation.

5.2.6 Correlations Involving the Sequential/Global Style considering Time of Day

When we considered Morning groups and Afternoon groups and correlated Test Grade and Sequential/Global scores, our results did not show any correlation. We then performed a linear regression by adding variable as Independent variables to our Regression, and our results did not show any correlation.

5.2.7 Correlations and Comparisons Involving the Sensing/Intuitive Style

When we considered the Overall group and correlated Test Grade with Sensing/Intuitive scores, our results showed a positive correlation with $p = .003$ and Beta = .227.

When we considered the Flow group and correlated Test Grade with Sensing/Intuitive scores, our results show a positive correlation with $p = .01$ and $Beta = .258$.

When we considered the Text group and correlated Test Grade with Sensing/Intuitive scores, our results show a positive correlation with $p = .03$ and $Beta = .238$.

When we considered the Overall group and performed a linear regression with Test Grade as the Dependent Variable and Sensing/Intuitive scores and GPA Range as the Independent Variables, our results show a positive correlation with Sensing/Intuitive values of $p = .001$ and $Beta = .245$ and GPA Range values of $p = .044$ and $Beta = .154$. We did not show this correlation for the Flow group or the Text group.

When we considered the Overall group and performed a linear regression with Test Grade as the Dependent Variable and Sensing/Intuitive scores, Age, and Test Time as the Independent Variables, our results show a correlation with Sensing/Intuitive values of $p = .008$ and $Beta = .193$, Age values of $p = .038$ and $Beta = -.151$, and Test Time values of $p < .001$ and $Beta = .372$. We did not show this correlation for the Flow group or the Text group.

We performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.5, and our results did not show any additional correlation.

Table 5.5 Test Grade Correlations Involving Sensing/Intuitive scores.

	Overall group		Flow group		Text group	
	p	Standardized Coefficient	p	Standardized Coefficient	p	Standardized Coefficient

Sensing/Intuitive	.003	.227	.01	.258	.03	.238
Sensing/Intuitive, GPA Range	.001 .044	.245 .154	N.S.			
Sensing/Intuitive, Age, GPA Range	.008 .038 <.001	.193 -.151 .372				

When we subdivided the Flow group and Text group by their Sensing/Intuitive score (with a midpoint at 0 and at Sensing = 2) and compared their Test Grades, we did not find any acceptable evidence, as defined in [25].

5.2.8 Correlations Involving the Sensing/Intuitive Style Considering Gender

When we considered the Females and correlated Test Grade with the Sensing/Intuitive scores, our results show a positive correlation with $p = .023$ and Beta = .201.

When we considered Females and performed a linear regression with Test Grade as the Dependent Variable and Sensing/Intuitive scores, GPA Range, and Test Time as the Independent Variables, our results show a positive correlation with Sensing/Intuitive values of $p = .027$ and Beta = .182, GPA Range values of $p = .023$ and Beta = .187, and Test Time values of $p < .001$ and Beta = .393.

When we considered Females and performed a linear regression with Test Grade as the Dependent Variable and Sensing/Intuitive scores and GPA Range as the Independent Variables, our results show a positive correlation with Sensing/Intuitive values of $p = .010$ and Beta = .226 and GPA Range values of $p = .014$ and Beta = .216. We performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.6, and our results did not show any additional correlation.

Table 5.6 Test Grade Correlations Involving Sensing/Intuitive scores considering Gender.

	Females		Males	
	p	Pearson Coefficient	p	Pearson Coefficient
Sensing/Intuitive	.023	.201	N.S.	
Sensing/Intuitive	.010	.226		
GPA Range	.014	.216		
Sensing/Intuitive	.027	.182		
GPA Range	.023	.187		
Test Time	<.001	.393		

5.2.9 Correlations Involving the Sensing/Intuitive Style Considering Time of Day

When we considered the Morning group and correlated Test Grade with the Sensing/Intuitive scores, our results show a positive correlation with $p = .027$ and Beta = .277.

When we considered the Morning group and performed a linear regression with Test Grade as the Dependent Variable and Sensing/Intuitive scores and Test Time as the Independent Variables, our results show a positive correlation with Sensing/Intuitive values of $p = .025$ and Beta = .264 and Test Time values of $p = .001$ and Beta = .416.

We performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.7, and our results did not show any additional correlation.

Table 5.7 Test Grade Correlations Involving Sensing/Intuitive scores considering Time of Day.

	Morning		Afternoon	
	p	Pearson Coefficient	p	Pearson Coefficient
Sensing/Intuitive	.027	.277	N.S.	
Sensing/Intuitive	.025	.264		

Test Time	.001	.416	
-----------	------	------	--

5.2.10 Correlations and Comparisons Involving the Active/Reflective Style

When we considered the Overall group, Flow group and Text group and correlated Test Grade and Active/Reflective scores, our results did not show any correlation. We then performed a linear regression by adding additional variables as Independent variables to our Regression, and our results did not show any correlation.

When we subdivided the Flow group and Text group by their Active/Reflective score (with a midpoint at 0 and at Active = 1) and compared their Test Grades, we did not find any acceptable evidence, as defined in [25].

5.2.11 Correlations Involving the Active/Reflective Style considering Gender

When we considered Males and Females and correlated Test Grade and Active/Reflective scores, our results did not show any correlation. We then performed a linear regression by adding additional variables as Independent variables to our Regression, and our results did not show any correlation.

5.2.12 Correlations Involving the Active/Reflective Style considering Time of Day

When we considered Morning groups and Afternoon groups and correlated Test Grade with Active/Reflective scores, our results did not show any correlation. We then performed a linear regression by adding variable as Independent variables to our Regression, and our results did not show any correlation.

5.2.13 Correlations Involving Test Grade with General Measures

When we considered the Overall group and correlated Test Grade with Test Time, our results show a positive correlation with $p < .001$ and Beta = .366.

When we considered the Flow group and correlated Test Grade with Test Time, our results show a positive correlation with $p < .001$ and Beta = .393.

When we considered the Text group and correlated Test Grade with Test Time, our results show a positive correlation with $p = .043$ and Beta = .266.

When we considered the Overall group and correlated Test Grade with Age, our results show a negative correlation with $p = .04$ and Beta = -.157.

When we considered the Overall group and performed a linear regression with Test Grade as the Dependent Variable and Age and Test Time as the Independent Variables, our results show a positive correlation with Age values of $p = .037$ and Beta = -.154 and Test Time values of $p < .001$ and Beta = .380.

We performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.8, and our results did not show any additional correlation.

Table 5.8 Test Grade Correlations Involving General Measures.

	Overall group		Flow group		Text group	
	p	Standardized Coefficient	p	Standardized Coefficient	p	Standardized Coefficient
Test Time	< .001	.366	< .001	.393	.043	.26
Age	.04	-.157	N.S.			
Age	.037	-.154				
Test Time	< .001	.380				

5.2.14 Correlations Involving Test Grade with General Measures considering Gender

Gender

When we considered the Females and correlated Test Grade with GPA Range, our results show a positive correlation with $p = .024$ and $Beta = .202$.

When we considered the Females and correlated Test Grade with Test Time, our results show a positive correlation with $p < .001$ and $Beta = .395$.

We performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.9, and our results did not show any additional correlation.

Table 5.9 Test Grade Correlations Involving General Measures considering Gender.

	Females		Males	
	p	Pearson Coefficient	p	Pearson Coefficient
GPA Range	.024	.202	N.S.	
Test Time	< .001	.395		

5.2.15 Correlations Involving Test Grade with General Measures considering Time of Day

When we considered the Morning group and correlated Test Grade with Test Time, our results show a positive correlation with $p = .001$ and $Beta = .410$.

When we considered the Afternoon group and correlated Test Grade with Test Time, our results show a positive correlation with $p = .002$ and $Beta = .292$.

We performed a linear regression by adding additional variables as Independent variables to our Regression, as summarized in Table 5.10, and our results did not show any additional correlation.

Table 5.10 Test Grade Correlations Involving General Measures considering Time of Day.

	Morning		Afternoon	
	p	Pearson Coefficient	p	Pearson Coefficient
Test Time	.001	.410	.002	.292

5.2.16 Comparisons of Test Grade considering Gender

When we considered the Females and Males and compared the Test Grade, our results show a significant difference with $t(170) = -3.352$, $p = .001$.

Table 5.11 Comparing General Demographics considering Gender.

	Male and Female		
	t	N	Sig
Test Grade	-3.352	170	.001

5.2.17 Comparisons of Test Grade considering Time of Day

When we considered the Morning group and the Afternoon group and compared the Test Grade, our results show a significant difference with $t(170) = 2.592$, $p = .01$.

Table 5.12 Comparing General Demographics considering Time of Day.

	Morning and Afternoon		
	t	N	Sig
Test Grade	2.592	170	.01

5.3 Correlations and Comparisons Involving the Index of Learning Styles

In this section, we will discuss the statistical tests that we have run and the correlations that we have found within our Overall group, our Flow group, and our Text group when not considering Test Grade (since we are not including Test Grade, we expect the data to be the same for all groups).

5.3.1 Correlations Involving the Visual/Verbal Style

We recall that *visual* learners are expected to prefer to remember best what they see, pictures, diagrams, flow charts, time lines, films and demonstration while *verbal* learners get more out of words, written and spoken explanations.

When we considered the Overall group and correlated the Visual/Verbal scores with the Active/Reflective scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .273.

When we considered the Flow group and correlated the Visual/Verbal scores with the Active/Reflective scores, our results show a positive correlation with $p = .007$ and a Pearson Correlation of .282.

When we considered the Text group and correlated the Visual/Verbal scores with the Active/Reflective scores, our results show a positive correlation with $p = .001$ and a Pearson Correlation of .365.

A summary of our results when correlating Visual/Verbal scores with ILS scores and when correlating Visual/Verbal scores with other General demographics are displayed in Table 5.13.

Table 5.13 Correlating Visual/Verbal scores with other ILS scores and other General demographics.

	Overall group		Flow group		Text group	
Visual/Verbal Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient	p	Pearson Coefficient
Active / Reflective (see 5.3.10)	< .001	.273	.007	.282	.001	.365
Sensing/Intuitive	N.S.					
Sequential / Global						
Test Time						
GPA Range						
Age						
Class Level						

5.3.2 Correlations Involving the Visual/Verbal Style considering Gender

When we considered the Females and correlated the Visual/Verbal scores with the Active/Reflective scores, our results show a positive correlation with $p = .002$ and a Pearson Coefficient of .275.

When we considered the Females and correlated the Visual/Verbal scores with the Sequential/Global scores, our results show a negative correlation with $p = .043$ and a Pearson Coefficient of -.180.

When we considered the Males and correlated the Visual/Verbal scores with the Sensing/Intuitive scores, our results show a positive correlation with $p = .025$ and a Pearson Coefficient of .335.

A summary of our results when considering Gender and correlating Visual/Verbal scores with ILS scores and when correlating Visual/Verbal scores with other General demographics are displayed in Table 5.14.

Table 5.14 Correlating Visual/Verbal scores with other ILS scores and other General demographics.

	Females		Males	
Visual/Verbal Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Active/Reflective	.002	.275	N.S.	
Sequential/Global	.043	-.180		
Sensing/Intuitive	N.S.		.025	.335
Visual/Verbal			N.S.	
Test Time				
Class Level				
Age				
GPA Range				

5.3.3 Correlations Involving the Visual/Verbal Style considering Time of Day

When we considered the Morning group and correlated the Visual/Verbal scores with the Sequential/Global scores, our results show a positive correlation with $p = .001$ and a Pearson Coefficient of .405.

When we considered the Afternoon group and correlated the Visual/Verbal scores with the Sequential/Global scores, our results show a positive correlation with $p = .046$ and a Pearson Correlation of .193.

A summary of our results when considering Time of Day and correlating Visual/Verbal scores with ILS scores and when correlating Visual/Verbal scores with other General demographics are displayed in Table 5.15.

Table 5.15 Correlating Visual/Verbal scores with other ILS scores and other General demographics considering Time of Day.

	Morning		Afternoon	
Visual/Verbal Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Sequential/Global	.001	.405	.046	.193
Active/Reflective	N.S.			
Test Time				
Sensing/Intuitive				
Class Level				
Age				
GPA Range				

5.3.4 Correlations Involving the Sequential/Global Style

We recall that *sequential* learners are expected to prefer to learn in linear steps, with each step following logically from the previous one while *global* learners learn in large jumps, absorbing material almost randomly without seeing connections, and then suddenly “getting it.”

When we considered the Overall group and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .382.

When we considered the Flow group and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p = .001$ and a Pearson Coefficient of .359.

When we considered the Text group and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .407.

When we considered the Flow group and correlated the Sequential/Global scores with the GPA Range, our results show a positive correlation with $p = .030$ and a Pearson Coefficient of .229. When we considered the Text group and the Overall group and

correlated the Sequential/Global scores with the GPA Range, our results did not show any correlation.

When we considered the Overall group, Flow group and Text group and correlated the Sequential/Global scores with Active/Reflective scores, Visual/Verbal scores, Age, Class Level, GPA Range and Test Time, our results did not show any correlation.

A summary of our results when correlating Sequential/Global scores with ILS scores and when correlating Sequential/Global scores with other General demographics are displayed in Table 5.16.

Table 5.16 Correlating Sequential/Global scores with other ILS scores and other General demographics.

	Overall Group		Flow Group		Text Group	
Sequential/Global Correlation with ...	p	Pearson Coefficient	p	Pearson Coefficient	p	Pearson Coefficient
Sensing/Intuitive	< .001	.382	.001	.359	< .001	.407
GPA Range	N.S.		.030	-.229	N.S.	
Active/Reflective	N.S.					
Visual/Verbal						
Age						
Class Level						
Test Time						

5.3.5 Correlations Involving the Sequential/Global Style considering Gender

When we considered the Females and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .368.

When we considered the Males and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p = .008$ and a Pearson Coefficient of .392.

When we considered the Females and correlated the Sequential/Global scores with the Visual/Verbal scores, our results show a negative correlation with $p = .043$ and a Pearson Coefficient of $-.180$.

When we considered the Males and correlated the Sequential/Global scores with Age, our results show a positive correlation with $p = .001$ and a Pearson Coefficient of .322.

When we considered the Males and correlated the Sequential/Global scores with Class Level, our results show a positive correlation with $p = .031$ and a Pearson Coefficient of .322.

A summary of our results when considering Gender and correlating Sequential/Global scores with ILS scores and when correlating Sequential/Global scores with other General demographics are displayed in Table 5.17.

Table 5.17 Correlating Sequential/Global scores with other ILS scores and other General demographics considering Gender.

Sequential/Global Correlation with...	Females		Males	
	p	Pearson Coefficient	p	Pearson Coefficient
Sensing/Intuitive	< .001	.368	.008	.392
Visual/Verbal	.043	-.180	N.S.	
Age	N.S.		.001	.471
Class Level			.031	.322
Active/Reflective	N.S.			
GPA Range				
Test Time				

5.3.6 Correlations Involving the Sequential/Global Style considering Time of Day

When we considered the Afternoon group and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .408.

When we considered the Morning group and correlated the Sequential/Global scores with the Sensing/Intuitive scores, our results show a positive correlation with $p = .005$ and a Pearson Correlation of .345.

When we considered the Afternoon group and correlated the Sequential/Global scores with the GPA Range, our results show a negative correlation with $p = .009$ and a Pearson Coefficient of -.252.

A summary of our results when considering Time of Day and correlating Sequential/Global scores with ILS scores and when correlating Sequential/Global scores with other General demographics are displayed in Table 5.18.

Table 5.18 Correlating Sequential/Global scores with other ILS scores and other General demographics considering Time of Day.

	Morning		Afternoon	
Sequential/Global Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Sensing/Intuitive	.005	.345	< .001	.408
GPA Range	N.S.		.009	-.252
Active/Reflective	N.S.			
Visual/Verbal				
Age				
Class Level				
Test Time				

5.3.7 Correlations Involving the Sensing/Intuitive Style

We review that *sensors* (from the ILS) are expected to prefer to draw on physical sensation, are practical and observing, prefer the concrete: facts and data, and prefer repetition while *intuitors* draw on insight, are imaginative and interpretive, prefer the abstract: theory and modeling, and prefer variation.

When we considered the Flow group and correlated the Sensing/Intuitive scores with the Visual/Verbal scores, our results show a positive correlation with $p = .007$ and a Pearson Coefficient of .282. When we considered the Text group and the Overall group and correlated the Sensing/Intuitive scores with the Visual/Verbal scores, our results did not show any correlation.

When we considered the Flow group and correlated the Sensing/Intuitive scores and the GPA Range, our results show a negative correlation with $p = .034$ and a Pearson Coefficient of -.224. When we considered the Text group and the Overall group and correlated the Sensing/Intuitive scores with the GPA Range, our results did not show any correlation.

When we considered the Overall group, Flow group and Text group and correlated the Sensing/Intuitive scores with other with the Active/Reflective scores, Visual/Verbal scores, Age, Class Level, and Test Time, our results did not show any correlation.

A summary of our results when correlating Sensing/Intuitive scores with ILS scores and when correlating Sensing / Intuitive scores with other General demographics are displayed in Table 5.19.

Table 5.19 Correlating Sensing/Intuitive scores with other ILS scores and other General demographics.

	Overall group		Flow group		Text group	
Sensing/Intuitive Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient	p	Pearson Coefficient
Sequential / Global (see 5.3.1)	< .001	.382	.001	.359	< .001	.407
Visual/Verbal	N.S.		.007	.282	N.S.	
GPA Range			.034	-.224		
Age			N.S.			
Class Level						
Active/Reflective						
Test Time						

5.3.8 Correlations Involving the Sensing/Intuitive Style considering Gender

When we considered the Males and correlated Sensing/Intuitive scores and Visual/Verbal scores, our results show a positive correlation with $p = .025$ and a Pearson Coefficient of .335.

When we considered the Males and correlated Sensing/Intuitive scores and Sequential/Global scores, our results show a positive correlation with $p = .008$ and a Pearson Coefficient of .392.

When we considered the Females and correlated Sensing/Intuitive scores and Sequential/ Global scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .368.

When we considered the Females and correlated Sensing/Intuitive scores and Active/Reflective scores, our results show a negative correlation with $p = .015$ and a Pearson Coefficient of -.215.

When we considered the Males and correlated Sensing/Intuitive scores and Class Level, our results show a positive correlation with $p = .049$ and a Pearson Coefficient of .296.

A summary of our results when considering Gender and correlating Sensing/Intuitive scores with ILS scores and when correlating Sensing/Intuitive scores with other General demographics are displayed in Table 5.20.

Table 5.20 Correlating Sensing/Intuitive scores with other ILS scores and other General demographics considering Gender.

	Females		Males	
Sensing/Intuitive Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Sequential/Global (see 5.3.5)	< .001	.368	.008	.392
Visual/Verbal	N.S.		.025	.335
Active/Reflective	.015	-.215	N.S.	
Class Level	N.S.		.049	.296
Age			N.S.	
GPA Range				
Test Time				

5.3.9 Correlations Involving the Sensing/Intuitive Style considering Time of Day

When we considered the Morning group and correlated the Sensing/Intuitive scores and the Sequential/Global scores, our results show a positive correlation with $p = .005$ and a Pearson Coefficient of .345.

When we considered the Afternoon group and correlated the Sensing/Intuitive scores and the Sequential/Global scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .408.

A summary of our results when considering Time of Day and correlating Sensing/Intuitive scores with ILS scores and when correlating Sensing/Intuitive scores with other General demographics are displayed in Table 5.21.

Table 5.21 Correlating Sensing/Intuitive scores with other ILS scores and other General demographics considering Time of Day.

	Morning		Afternoon	
Sensing/Intuitive Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Sequential/Global (see 5.3.6)	.005	.345	< .001	.408
Visual/Verbal	N.S.			
Active/Reflective				
GPA Range				
Age				
Class Level				
Test Time				

5.3.10 Correlations Involving the Active/Reflective Style

We recall that *active* learners are expected to prefer to learn by trying things out and enjoy working in groups while *reflective* learners learn by thinking things through and prefer working alone or with a single familiar partner.

When we considered the Overall group and correlated Active/Reflective scores and Visual/Verbal scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .273.

When we considered the Flow group and correlated Active/Reflective scores and Visual/Verbal scores, our results show a positive correlation with $p = .007$ and a Pearson Coefficient of .282.

When we considered the Text group and correlated Active/Reflective scores and Visual/Verbal scores, our results show a positive correlation with $p = .001$ and a Pearson Coefficient of .365.

When we considered the Overall group and correlated the Active/Reflect scores and Test Time, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .247.

When we considered the Text group and correlated the Active/Reflective scores and Test Time, our results show a positive correlation with $p=.009$ and a Pearson Coefficient of .294.

When we considered the Flow group and correlated the Active/Reflective scores and Test Time, our results show a positive correlation with $p=.037$ and a Pearson Coefficient of .222.

When we considered the Text group and correlated the Active/Reflective scores and Test Time, our results show a positive correlation with $p=.009$ and a Pearson Coefficient of .294.

When we considered the Overall group, Flow group, and Text group and correlated the Active/Reflective scores and Age, Class Level, and GPA Range, our results did not show any correlation.

A summary of our results when correlating Active/Reflective scores with ILS scores and when correlating Active/Reflective scores with other General demographics are displayed in Table 5.22.

Table 5.22 Correlating Active/Reflective scores with other ILS scores and other General demographics.

	Overall group		Flow group		Text group	
Active/Reflective Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient	p	Pearson Coefficient
Visual/Verbal	< .001	.273	.007	.282	.001	.365
Test Time	.009	.294	.037	.222	.009	.294
Sensing/Intuitive	N.S.					
Sequential / Global						
GPA Range						
Age						
Class Level						

5.3.11 Correlations Involving the Active/Reflective Style considering Gender

When we considered the Females and correlated the Active/Reflective scores and the Sensing/Intuitive scores, our results show a negative correlation with $p = .015$ and a Pearson Coefficient of $-.215$.

When we considered the Females and correlated the Active/Reflective scores and the Visual/Verbal scores, our results show a positive correlation with $p = .002$ and a Pearson Coefficient of $.275$.

When we considered the Females and correlated the Active/Reflective scores and the Test Time, our results show a positive correlation with $p = .002$ and a Pearson Coefficient of $.281$.

A summary of our results when considering Gender and correlating Active/Reflective scores with ILS scores and when correlating Active/Reflective scores with other General demographics are displayed in Table 5.23.

Table 5.23 Correlating Active/Reflective scores with other ILS scores and other General demographics.

	Females		Males	
Active/Reflective Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Sensing/Intuitive	.015	-.215	N.S.	
Visual/Verbal	.002	.275		
Test Time	.002	.281		
Sequential/Global	N.S.			
Class Level				
Age				
GPA Range				

5.3.12 Correlations Involving the Active/Reflective Style considering Time of Day

When we considered the Morning group and correlated the Active/Reflective scores and Visual/Verbal scores, our results show a positive correlation with $p < .001$ and a Pearson Coefficient of .405.

When we considered the Afternoon group and correlated the Active/Reflective scores and Visual/Verbal scores, our results show a positive correlation with $p = .046$ and a Pearson Coefficient of .193.

When we considered the Afternoon group and correlated the Active/Reflective scores and Test Time, our results show a negative correlation with $p < .001$ and a Pearson Coefficient of .337.

A summary of our results when considering Time of Day and correlating Active/Reflective scores with ILS scores and when correlating Active/Reflective scores with other General demographics are displayed in Table 5.24.

Table 5.24 Correlating Active/Reflective scores with other ILS scores and other General demographics considering Time of Day.

	Morning		Afternoon	
Active/Reflective Correlation with...	p	Pearson Coefficient	p	Pearson Coefficient
Visual/Verbal	.001	.405	.046	.193
Test Time	N.S.		< .001	.337
Sensing/Intuitive	N.S.			
Sequential/Global				
Class Level				
Age				
GPA Range				

5.3.13 Comparisons of General Demographics considering Gender

When we considered the Males and Females and compared the Sensing/Intuitive scores, our results show a significant difference with $t(170) = -2.26$, $p = .025$.

When we considered the Males and Females and compared the GPA Range, our results show a significant difference with $t(166) = 2.424$, $p = .016$.

Table 5.25 Comparing General Demographics considering Gender.

	Males		Females		Males and Females		
	Males Mean	n	Females Mean	n	t	N	Sig
Sensing/Intuitive	-.33	45	-2.46	127	-2.26	170	.025
GPA Range	2.97	43	3.22	125	3.434	166	.016
Class Level	N.S.						
Sequential/Global							
Active/Reflective							
Visual/Verbal							
Test Time							
Age							

5.3.14 Comparisons of General Demographics considering Time of Day

When we considered the Morning group and Afternoon group and compared Age, our results show a significant difference with $t(168) = 2.456$, $p = .015$.

Table 5.26 Comparing General Demographics considering Time of Day.

	Morning		Afternoon		Time of Day		
	Morning Mean	n	Afternoon Mean	n	t	N	sig
Age	22.86	64	20.54	106	2.456	168	.015
Sensing/Intuitive	N.S.						
Class Level							
Sequential/Global							
Active/Reflective							
Visual/Verbal							
Test Time							
GPA Range							

5.4 Summary

In this chapter we correlated data to determine the validity of the ILS and we compared General measures of our groups to discover differences between our groups. We also correlated and compared data to determine what effects the Test Grade. We did not find any acceptable evidence, as defined in [25], to support the claim that any of the dimensions in the Index of Learning Styles are valid at determining Test Grade. In the next chapter we discuss the most interesting correlations and propose explanations.

CHAPTER VI

DISCUSSION

In this chapter, we will discuss some of the results that we have found, and propose some novel explanations for some of the correlations, while referring to the literature for some existing explanations. Sections 6.1 and 6.2 discuss the findings relating to our hypothesis, while Section 6.3 discusses a significant comparison of Test Grades that we found when exploring our data and Section 6.4 discusses an existing experiment that we could use to frame a new hypothesis. Finally, Sections 6.5, 6.6, and 6.7 discuss other ILS correlations that we found while exploring our data, but did not test for explicitly.

6.1 Test Grade and Visual/Verbal Correlation Explanation

In our work, we tested the hypothesis that *computer science novices understand computer science concepts (here sequence, iteration, alternative) better, and test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. In Section 5.2.1, we performed a statistical test to measure correlation between the test grade of the Visual and Verbal participants in the Flow group and the test grade of the Visual and Verbal participants in

the Text Group. Our results did not show any statistically significant correlation between Test Grade and Visual/Verbal scores. As this dimension was a major component of our research, it casts doubt on the hypothesis. There is evidence that our hypothesis is flawed, as suggested by [25]. The authors performed a thorough literature review and also did not find any experiments that correlate learning styles with actual results.

Although we did not show a significant understanding difference within the Flow group and the Text group when we divided each of these groups into Visual and Verbal learners; it is important to remember that the Flow Group outperformed the Text group, regardless of either of their Visual or Verbal scores, which leads us to believe that either this dimension of the ILS may not be a good measurement for our experiment, or our experiment did not actually present computer science concepts in a well-divided Visual and Verbal way; however, the Flow Teaching Style scored statistically significantly higher than the Text Teaching Style (2.88/5.00 compared to 2.31/5.00) in our experiment for the computer science concept of alternative.

6.2 Test Grade and Flow Teaching Style and Text Teaching Style Explanation

In Section 5.1, we performed a statistical test for comparison between the Flow group and the Text group. The Flow Teaching Style was more effective for novice students than our Text Teaching Style. More specifically, the Flow Teaching Style was more effective for understanding the computer science concept of alternative for novice students than our Text Teaching Style. There could be several explanations for this.

1. We suggest that even the most visual language has some text present, and this perhaps allows for the novice's brain to process the shapes with the right

hemisphere of their brain and the text with the left hemisphere of their brain, as suggested in [35].

2. The concept understood most correctly by the Flow Group is the concept of Alternative. Alternative is represented in textual programming languages with groups of commands surrounded by opening “{” and closing “}” brackets that indicate the beginning and the end of a group of commands. We suggest that there is nothing intuitive about brackets to differentiate what code to execute and what code to skip. We suggest that instead, there is something very intuitive about following different arrowed lines if a statement is true or false.

These results are meaningful because we have shown that the existing way of understanding programming can be improved, and we have showed one way that the existing way of representing program code can be implemented. Our study was limited to computer science novices, so we do not know if the Flow group still outperformed the Text group when both groups have more computer science experience. Our study only tested three different concepts, so we do not know the effect on difference concepts. Our study was limited in time to present to our novices. Our study did not allow our novices to immerse themselves in the software environment, so we could not measure if the Flow group’s time on task was increased compared to the Text group’s time on task.

6.3 Sensing/Intuitive and Test Grade Correlation Explanation

In Section 5.2.7, 5.2.8, and 5.2.9, we performed statistical tests for correlation between the Sensing/Intuitive Learning Style and the Test Grade. We propose as an explanation that those who scored higher on the intuitive dimension correlated with

higher test scores because for a novice, a software program is very abstract, there is nothing created by a software program except what gets printed to the screen, the rest must be modeled in some sort of abstract representation. We are encouraged by these results, and would consider testing the effect of Sensing/Intuitive scores on performance for future work.

6.4 Sensing/Intuitive Experiment

We will now discuss an existing experiment that was used to test the Sensing/Intuitive dimension. This experiment is in the medical-education context [111] that examined the hypothesis that sensing learners would do better when given instruction in which the problem was presented prior to the content information used to solve the problem, and intuitive learners would do better with the reverse. While they were not able to find support for their hypothesis with 123 internal medicine residents covering four ambulatory medicine topics, they do present a format for a sensing-intuitive experiment. We did not use this experiment for our work, but we would consider this for future work.

6.5 Correlations between Sensing/Intuitive and Sequential/Global

In Section 5.3.1, we performed statistical tests for a correlation between Sensing/Intuitive and Sequential/Global learning styles, but we did not show any correlation with our data. Richard Felder explains correlation between and Sequential/Global scales and Sensing/Intuitive as

“Sequential learners, who acquire understanding in logical connected steps, could be either sensors or intuitors, but global

learners, whose thinking processes tend to be nonlinear and who acquire understanding holistically, would seem much more likely to be intuitive than sensing.” [45, p. 104]

While Felder’s explanation seems intuitive, we were surprised to not find this correlation in our data. Our study was not designed to address these particular learning styles. We cannot confirm nor refute this claim.

6.6 Sequential/Global and Visual/Verbal

In Section 5.3.1, we performed statistical tests for a correlation between Sequential/Global and Visual/Verbal learning styles, but we did not show any correlation with our data. This is counter to [112], which suggests that based on evidence from brain hemisphere research and clinical observation that global (or ‘visual-spatial’) learners are more likely to be visual processors and sequential (‘auditory-sequential’) learners are more likely to be verbal processors. Again, our study was not designed to address these particular learning styles, so we cannot confirm nor refute this observation.

6.7 Sensing/Intuitive and Visual/Verbal Correlation Explanation

In Section 5.3.7, we performed statistical tests for a correlation between Sensing/Intuitive and Visual/Verbal learning styles, and we showed a positive correlation with our Flow group. We propose as an explanation for our positive Flow group correlation between the Sensing/Intuitive scale and the Sequential/Global scales are related based on [45], and that the Sequential/Global scale and the Visual/Verbal scales are related based on [112], and that could explain the small positive correlation between

the Sensing/Intuitive scale and the Visual/Verbal scale for our Flow group. Further research would be needed to verify this theory.

CHAPTER VII

FUTURE WORK

While we did test at least one of the original 28 topics discussed in Table 1.2 and Table 3.7, we did not test the visual understanding of all 28 topics, as not all could be tested in a reasonable amount of time, and not all topics will benefit from a visual representation. We present suggestions of work that could continue and improve this line of research in the future.

7.1 Improvements to RAPTOR

We believe that RAPTOR can be improved, too. While performing our research, several potential areas of improvement were identified.

- Round-tripping. By making modifications to the text code and then have those imported back into the flow-model design, the novice will be able to make changes in either text or the flowchart, and will only need to know one in order to be able to see the other.
- Concurrent displays of the flowchart and a particular language's text, which will be necessary to teach syntax to novices.

- Clear expansion and contraction of user-defined functions. To begin, novices will need the details of a function, but to save space, and also to take out unnecessary complexities after they understand the function, they will want to simplify the visualization of the function.
- Multidimensional flowchart. If the flowchart were to be able to grow two-dimensionally, then screen space will be more effectively utilized.
- Finally, as part of a more complete solution, the Map metaphor [55] should be explored.

7.2 Suggestions to Improve Computer Science Curriculum

We would like to incorporate our Flow Teaching Style into an entire introductory computer science course. If these longitudinal tests, which would augment all of the concepts already being taught in an introductory computer science course, determine that this teaching style positively affects outcomes, then we should create a working group to develop and support a robust visual programming tool. With this foundation, we would develop a computer science curriculum using this visual programming tool, and analyze the results.

7.2.1 Flow and Text Separately

Although we have found that the Flow Teaching Style is statistically significantly better than the Text Teaching Style in our research, we believe that there is more data to be collected. Another experiment, one that combines both our Flow and Text slides together is one such experiment that can be performed to measure the effectiveness of

representing code as a flow chart or as text. This experiment should follow the previous experimental design that we have used, but additionally collect data to measure if left handed novices score higher than right handed novices, as 19.8% of the left-handed have bilateral language functions [36]. An example of what this would look like is presented in Figure 7.1.

1st Program

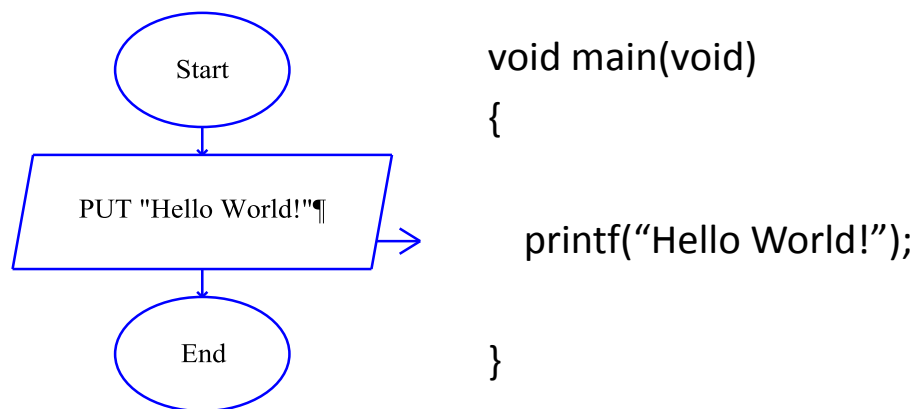


Figure 7.1 An example of what Flow and Text would look like together.

7.2.2 Flow and Text Together

We could also try to add the most effective uses of our Flow teaching style (alternative) together with our existing Text curriculum. Our existing Test Question 1 would now look like Figure 7.2.

Test Question 1

1. If the user types in *2000* when prompted, what, if anything, will print to the screen?

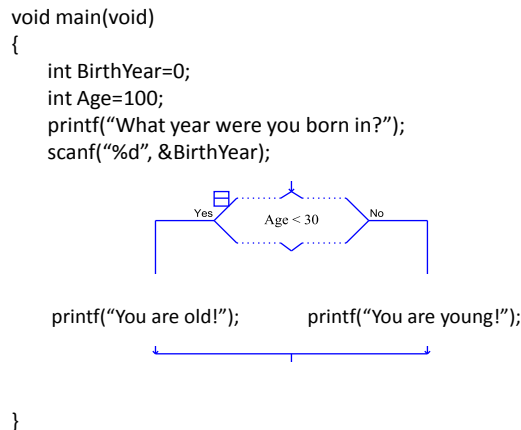


Figure 7.2 An example of inserting Flow control into our Text slide.

7.3 What's Next?

There are at least two paths forward. In both paths, we measure the effectiveness of our tool as we iterate towards an enterprise-level visual programming language. We have already shown that this Flow Method benefits novices' understanding of the computer science concept of alternative better than the current Text Method; however, there are many other computer science concepts that must be understood in order to be an expert computer scientist. As we move forward, there is existing support for learning object-oriented computer science using RAPTOR [78] , and we can use this research along with executable Universal Modeling Language (UML)[113] to determine the best method to understand object-oriented programming. If, in our research, we learn that a visual programming language is not superior to the current textual programming

languages in understanding all computer science concepts, then we must pursue a second path of developing a visual programming language that maps directly to a textual programming language.

By developing a visual programming language that maps directly to a textual programming language, the user gets to learn programming in the way easiest for them, and then directly transfer that understanding to the other representation. Even an expert computer scientist is not an expert in all domains, and this method can help those not familiar with a domain to quickly and easily understand existing code, which is important in a software maintenance task [114].

CHAPTER VIII

CONCLUSION

People understand differently. In our work, we tested the hypothesis that *computer science novices understand computer science concepts (here sequence, iteration, alternative) better, and test statistically significantly higher, if those concepts are presented to them with a flow-model language, than by the text-based way alone*. We initially thought people who preferred learning through a Visual Learning Style would benefit from a Flow Teaching Style and that people who preferred learning through a Verbal Learning Style would benefit from a Text Teaching Style. However, we did not show that with this work. We did show, however, that all novices (regardless of preferred Learning Style) understand the computer science concept of *alternative* better when presented with a Flow Teaching Style than with the Text Teaching Style.

Although our analysis was complete for this work, it was not exhaustive, and we present our data online for further review.

- We used the Statistical Package for the Social Sciences to collect our data, and that data can be found at <http://bit.ly/c7FNTh>
- The presentation that we used for the Text group can be found at <http://bit.ly/aUvrKd>

- The presentation that we used for the Flow group can be found at
<http://bit.ly/a84LcG>
- The answer sheet that was used to grade the tests can be found at
<http://bit.ly/bwy1Vd>
- The signup sheet that was used to collect volunteers can be found at
<http://bit.ly/bHmuUt>
- A video of the author presenting one of the experiments can be found at
<http://bit.ly/aOSczm>

APPENDICES

Appendix A

The Questionnaire

This questionnaire is given to both the test and control group to measure certain common variables that we will like to measure for statistical purposes. Following this is the 44 question Index of Learning Styles Questionnaire.

ID: _____

Age: _____

Sex: **Male** or **Female**

Left or **Right** Handed

Class Rank: Freshman Sophomore Junior Senior

Overall GPA: <2.0 2.0-2.5 2.5-3.0 3.0-3.5 >3.5

What Grade do you expect to make in the course? A B C D F

What is your major?

If your major is not Computer Science, why did you take this class?

List the math classes have you taken, including the grades for each math class:

Previous Programming Experience

If you have learned programming before, when did you learn it? If never, circle never. If other, please write in where.

Never Before High School High School College Other: _____

Do you have any experience programming outside a classroom environment? If so, what experience do you have?

1. I understand something better after I
(a) try it out (b) think it through
2. I would rather be considered
(a) realistic (b) innovative.
3. When I think about what I did yesterday, I am most likely to get
(a) a picture. (b) words.
4. I tend to
(a) understand details of a subject but may be fuzzy about its overall structure. (b) understand the overall structure but may be fuzzy about details.
5. When I am learning something new, it helps me to
(a) talk about it. (b) think about it.
6. If I were a teacher, I would rather teach a course
(a) that deals with facts and real life situations. (b) that deals with ideas and theories.
7. I prefer to get new information in
(a) pictures, diagrams, graphs, or maps. (b) written directions or verbal information.
8. Once I understand
(a) all the parts, I understand the whole thing. (b) the whole thing, I see how the parts fit.
9. In a study group working on difficult material, I am more likely to
(a) jump in and contribute ideas. (b) sit back and listen.
10. I find it easier
(a) to learn facts. (b) to learn concepts.
11. In a book with lots of pictures and charts, I am likely to
(a) look over the pictures and charts carefully. (b) focus on the written text.
12. When I solve math problems
(a) I usually work my way to the solutions one step at a time. (b) I often just see the solutions but then have to struggle to figure out the steps to get to them.
13. In classes I have taken

- (a) I have usually gotten to know many of the students. (b) I have rarely gotten to know many of the students.
14. In reading nonfiction, I prefer (a) something that teaches me new facts or tells me how to do something. (b) something that gives me new ideas to think about.
15. I like teachers (a) who put a lot of diagrams on the board. (b) who spend a lot of time explaining.
16. When I'm analyzing a story or a novel (a) I think of the incidents and try to put them together to figure out the themes. (b) I just know what the themes are when I finish reading and then I have to go back and find the incidents that demonstrate them.
17. When I start a homework problem, I am more likely to (a) start working on the solution immediately. (b) try to fully understand the problem first.
18. I prefer the idea of (a) certainty. (b) theory.
19. I remember best (a) what I see. (b) what I hear.
20. It is more important to me that an instructor (a) lay out the material in clear sequential steps. (b) give me an overall picture and relate the material to other subjects.
21. I prefer to study (a) in a study group. (b) alone.
22. I am more likely to be considered (a) careful about the details of my work. (b) creative about how to do my work.
23. When I get directions to a new place, I prefer (a) a map. (b) written instructions
24. I learn (a) at a fairly regular pace. If I study hard, I'll "get it." (b) in fits and starts. I'll be totally confused and then suddenly it all "clicks."
25. I would rather first (a) try things out. (b) think about how I'm going to do it
26. When I am reading for enjoyment, I like writers to

- (a) clearly say what they mean. (b) say things in creative, interesting ways.
27. When I see a diagram or sketch in class, I am most likely to remember
(a) the picture. (b) what the instructor said about it.
28. When considering a body of information, I am more likely to
(a) focus on details and miss the big picture. (b) try to understand the big picture before getting into the details.
29. I more easily remember
(a) something I have done. (b) something I have thought a lot about.
30. When I have to perform a task, I prefer to
(a) master one way of doing it. (b) come up with new ways of doing it.
31. When someone is showing me data, I prefer
(a) charts or graphs. (b) text summarizing the results.
32. When writing a paper, I am more likely to
(a) work on (think about or write) the beginning of the paper and progress forward. (b) work on (think about or write) different parts of the paper and then order them.
33. When I have to work on a group project, I first want to
(a) have "group brainstorming" where everyone contributes ideas. (b) brainstorm individually and then come together as a group to compare ideas.
34. I consider it higher praise to call someone
(a) sensible. (b) imaginative.
35. When I meet people at a party, I am more likely to remember
(a) what they looked like. (b) what they said about themselves.
36. When I am learning a new subject, I prefer to
(a) stay focused on that subject, learning as much about it as I can. (b) try to make connections between that subject and related subjects.
37. I am more likely to be considered
(a) outgoing. (b) reserved.
38. I prefer courses that emphasize
(a) concrete material (facts, data). (b) abstract material (concepts, theories).
39. For entertainment, I would rather
(a) watch television. (b) read a book.

40. Some teachers start their lectures with an outline of what they will cover. Such outlines are
(a) somewhat helpful to me. (b) very helpful to me.
41. The idea of doing homework in groups, with one grade for the entire group,
(a) appeals to me. (b) does not appeal to me.
42. When I am doing long calculations,
(a) I tend to repeat all my steps and check my work carefully. (b) I find checking my work tiresome and have to force myself to do it.
43. I tend to picture places I have been
(a) easily and fairly accurately. (b) with difficulty and without much detail.
44. When solving problems in a group, I would be more likely to
(a) think of the steps in the solution process. (b) think of possible consequences or applications of the solution in a wide range of areas.

Appendix B

The Script Followed During the Experiment

In an effort to reduce the author's bias when performing the experiments, we present the script that was followed during the experiments for analysis that the two groups were treated equally when learning computer science concepts. The following sections correspond to the slides for both the Text and Flow groups. A "T" will indicate that the slide was presented to the Text group, and an "F" will indicate that the slide was presented to the Flow group. In the event that both groups were presented the same slide, then a "S" will indicate the slide (which typically occurred at the beginning of the experiment). For example, 5-F indicates that the script corresponds to the fifth slide that the Flow group was exposed to and 6F, 10T would indicate that the slide corresponds to the sixth slide that the Flow group was exposed to and the tenth slide that the Text group was exposed to (which typically occurs when displaying the output of both programs). If the differences in the script between both the Flow and Text groups are more than a few sentences, then we have displayed the scripts in a table.

B.1 Introduction and Agenda

S1. Hello and thanks for volunteering for this experiment. I'm B.J. Smith and I'm working on computer science understanding. S2. I'm going to ask you to fill out a survey, and then we are going to learn about some computer science concepts for about 25 minutes, and then test these concepts. You may now fill out your 5 page survey. I've allocated 15 minutes for you to do this.

B.2A Computer Program

S3. Generally, a computer program is a series of commands that are understood by a computer. It's written in an EXACT language that is unambiguous; a computer can understand a program in only one way. English wouldn't work as a programming language because of such ambiguities as the line "Herb roasted chicken," which could mean that a guy named Herb is roasting a chicken, or it could describe a chicken that has been roasted with herbs and spices. Software runs commands in order, and will do whatever you ask it to do, as long as you ask in a language that it understands. Some popular computer languages are C, C++, C#, Java, and Python.

B.3 Sequence

S4. So this is our first concept that we will learn. Sequence. Programs are read and understood from top to bottom, in order, following the arrows, and until a line has been reached, the computer doesn't know about it. Let's look at an example.

Table B.1 Slide 5 Flow and Text scripts.

Slide 5 Flow Script	Slide 5 Text Script
So, this is probably the first program that you guys have ever seen. Most computer languages have some sort of "start" and an "end" as shown in Figure 3.6.	So, this is probably the first program that you guys have ever seen. Every C program has a "main" function. The words "void" here are more advanced for this class, but you have to have something close to that there.

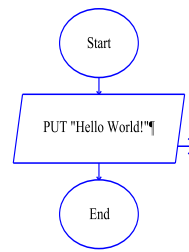


Figure B.1 Slide 5F, presenting “Hello World!” using RAPTOR.

Table B.2 Slide 6 Flow and Text scripts.

Slide 6 Flow Script	Slide 6 Text Script
Note that there are arrows. There are arrows between the shapes, vertically, which indicate to you, and to the computer, the direction that the code travels. There is also an arrow that points to the right, which indicates that data is going out to the screen.	Note that for every opening bracket, this is a corresponding closing bracket, and that each command and most lines, here, “printf”, ends with a semicolon

S7. So this particular program would print “Hello World” to the screen as shown in Figure 3.7.

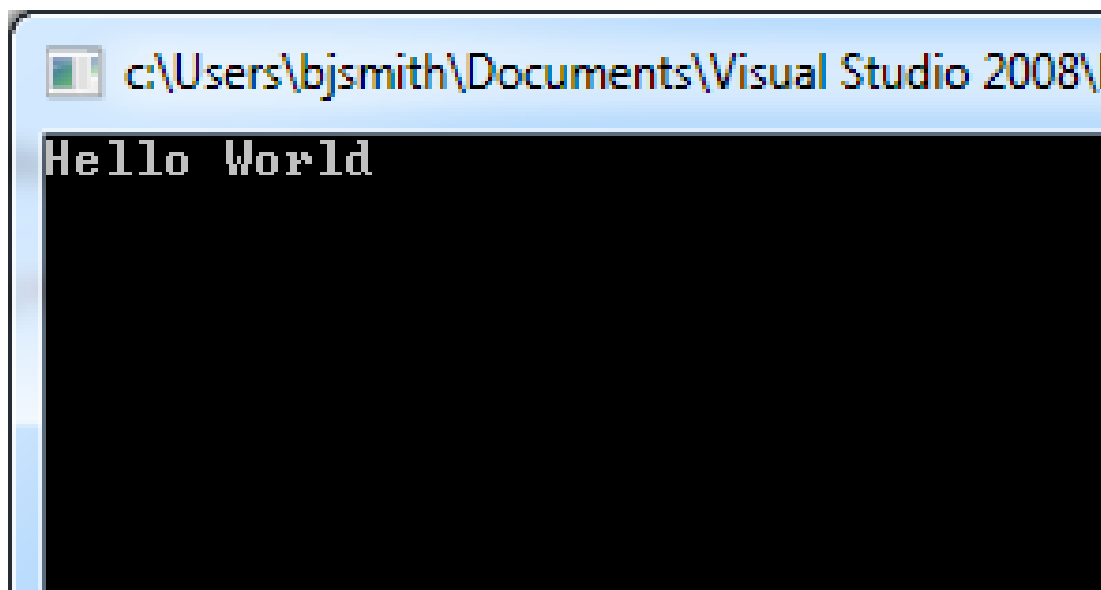


Figure B.2 Slide 8, displaying Hello World in a command prompt.

S8. Now this is a real simple program, but let's walk through it.

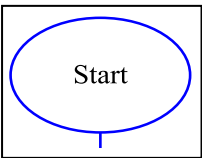


Figure B.3 Slide 9F, RAPTOR's Start.

Table B.3 Slide 9 Flow and Text scripts.

Slide 9 Flow Script	Slide 9 Text Script
This is where you start. Remember, the computer doesn't know what all is in the program at the beginning, even though we do, it only sees the Start.	You start at the "main." Remember, the computer doesn't know what all is in the program at the beginning, even though we do, it only sees the beginning line.

10T. The program then reads the next line, finds an opening bracket, which indicates to the computer that there is something inside that the computer can understand. It also means that there must be a corresponding closing bracket, and we'll see that later.

Table B.4 Slide 10 Flow and Slide 11 Text scripts.

Slide 10 Flow Script	Slide 11 Text Script
We only have one direction to go from Start, which is to the PUT command. This command, PUT, means print to the screen, and whatever comes after PUT, in quotation marks mean "put what is in the quotation marks on the screen."	The next line is a command. This command, "printf," means print to the screen, and whatever comes after printf, surrounded by quotation marks, and these quotation marks enclosed by parenthesis, will be printed to the screen.

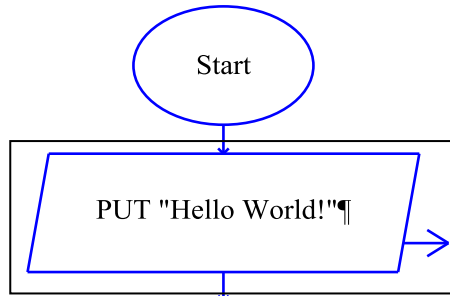


Figure B.4 Slide 10F, RAPTOR’s PUT statement, PUT “Hello World!”

Table B.5 Slide 11 Flow and Slide 12 Text scripts.

Slide 11 Flow Script	Slide 12 Text Script
There is only one way to go from here, and that is to the End.	The next line is the end of the program.

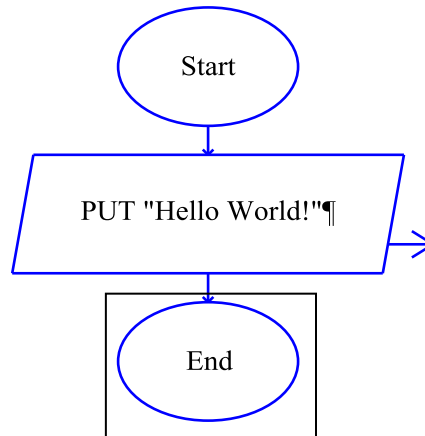


Figure B.5 Slide 11F, RAPTOR’s End.

12F, 13T. Now, this program was not very interesting, because we knew already, just by looking at it, what the program was going to do. What we need is a variable to hold some value that could change.

B.4 Variable

Table B.6 Slides 13-14 Flow and Slides 14-15 Text scripts.

Slide 13 Flow Script	Slide 14 Text Script
<p>A variable is a symbolic name associated with a value whose associated value may be changed, or vary, hence the name. Every variable has a type, which tells you if the variable will contain a whole number, or a decimal number, or a character. A variable only has a value if the variable is explicitly set, which is typically done with some sort of equation, where the variable is on the left, and the value that you want to give it is on the right, and there is an arrow, pointing to the variable, in between. If the variable is not set, then the variable's value is unknown, or colloquially known as garbage. Here is an example of a variable, number, being created and initialized to 0. It's important to remember that if the program doesn't set a value to the variable, then the computer has absolutely no idea what that variable is worth.</p>	<p>A variable is a symbolic name associated with a value whose associated value may be changed, or vary, hence the name. Every variable has a type, which tells you if the variable will contain a whole number, or a decimal number, or a character. A variable only has a value if the variable is explicitly set, which is typically done with some sort of equation, where the variable is on the left, and the value that you want to give it is on the right, and there is an equal sign in between. If the variable is not set, then the variable's value is unknown, or colloquially known as garbage. Here is an example of a variable, number, being created and initialized to 0. It's important to remember that if the program doesn't set a value to the variable, then the computer has absolutely no idea what that variable is worth.</p>
Slide 14 Flow Script	Slide 15 Text Script
<p>Here we have our first variable, BirthYear. It can take different values. So let's run through this computer program.</p>	<p>Here we have our first variable, BirthYear. The word before BirthYear, "int", means that the values of BirthYear are going to be integers. So let's run through this computer program.</p>

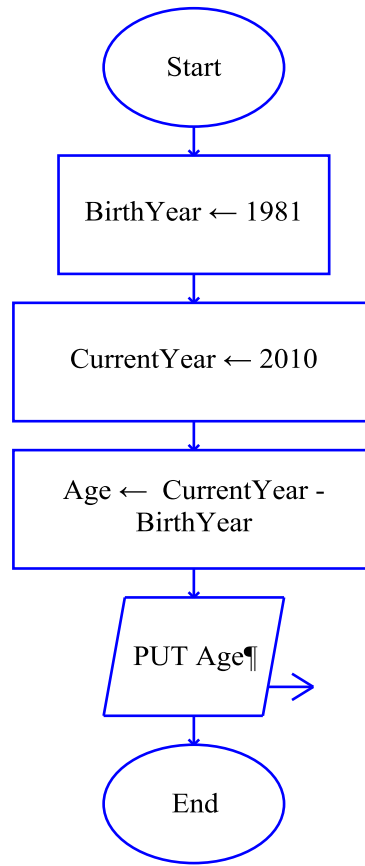


Figure B.6 Slide 14F, program to calculate Age, hardcoded values.

Table B.7 Slide 15 Flow and Slides 16-17 Text scripts.

Slide 15 Flow Script	Slide 16 Text Script
We start at Start, then follow the arrows.	We start with “main”, and then read each line in order.
	Slide 17 Text Script
	We read the next line, which is an opening bracket.

16F, 18T. And now read the first command. We create a variable, BirthYear, and set its value to 1981.

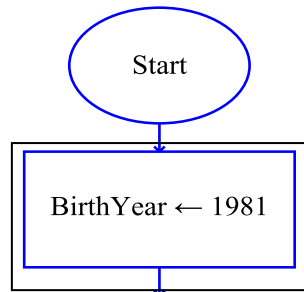


Figure B.7 Slide 16F, BirthYear = 1981 highlighted.

17F, 19T. Next we create a variable called CurrentYear, and set its value to 2010.

18F, 20T. Then we subtract BirthYear from CurrentYear.

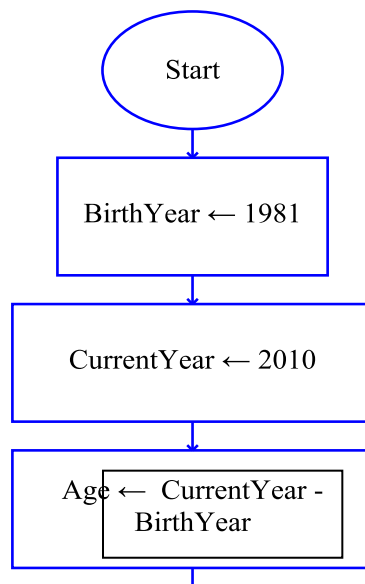


Figure B.8 Slide 18F, calculating Age.

19F, 21T. And then we create a variable, Age, and set its value to what we just calculated, CurrentYear minus BirthYear.

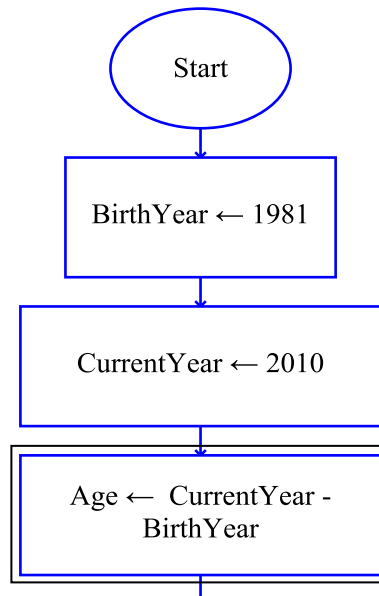


Figure B.9 Slide 19F, setting value of Age.

20F, 22T. And we print out Age to the screen, which here would be 29.

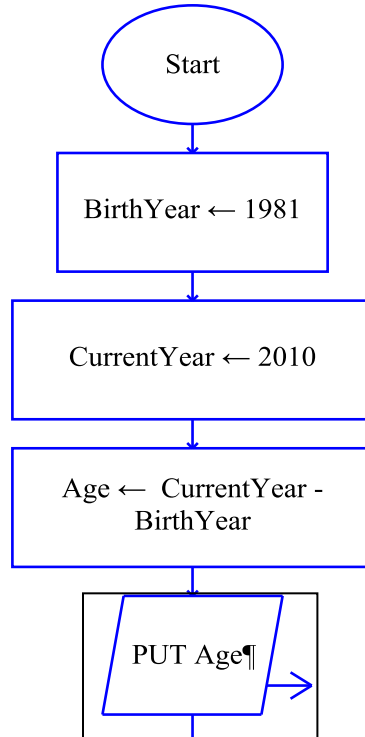


Figure B.10 Slide 20F, printing out Age.

21F, 23T. As shown.

Table B.8 Slide 22 Flow and Slide 24 Text scripts.

Slide 22 Flow Script	Slide 24 Text Script
We follow the last arrow to “End,” where we end the program.	We read the next line, the closing bracket, that indicates that we are at the end of the program.

23F, 25T. Now this is not very useful either, because if we already knew the values, a program wouldn’t be that powerful. Instead, programs get powerful when you have more user interaction, when software gets information from outside the program.

B.5 User input

Table B.9 Slide 24 Flow and Slide 26 Text scripts.

Slide 24 Flow Script	Slide 26 Text Script
Since computers know so very little on their own, they often have to get their information from somewhere else. Often times, this can be another file, or a user sitting at the computer feeding the computer answers to questions that it has. In order to get information from the user, the program would prompt the user to enter certain information, such as your birth year, like here where we use a PUT statement, and then immediately read in the information that is typed in to the command window, like here where we use a GET statement.	Since computers know so very little on their own, they often have to get their information from somewhere else. Often times, this can be another file, or a user sitting at the computer feeding the computer answers to questions that it has. In order to get information from the user, the program would prompt the user to enter certain information, such as your birth year, like here where we use a printf statement, and then immediately read in the information that is typed in to the command window, like here where we use a scanf statement.

User input

- Usually consists of two parts:
 - PUT, informing the user of something
 - GET, reading a value from the command line

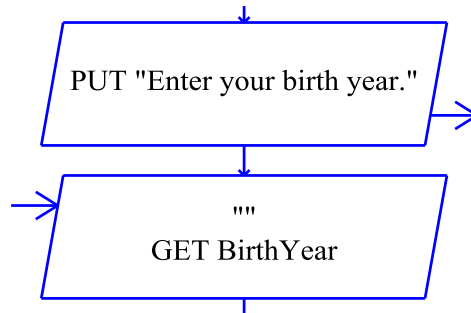


Figure B.11 Slide 24F, introducing the concept of User Input.

Table B.10 Slide 25 Flow and Slide 27 Text scripts.

Slide 25 Flow Script	Slide 27 Text Script
So this program asks the user “What year were you born in?” and the user can type in the year. We trust that the user is going to type in a whole positive number, so we are not going to check what the user types in. Then the program is going to print to the screen “You were born in 1990” or whatever year the user typed in. We get our information from the parallelogram, or right-leaning rectangle, that has an arrow pointing to it. The GET command reads in what has been typed in from the screen, and puts that value in the variable that is named after the GET command.	So this program asks the user “What year were you born in?” and the user can type in the year. We trust that the user is going to type in a whole positive number, so we are not going to check it to make sure. Then the program is going to print to the screen “You were born in 1990” or whatever year the user typed in. We get our information from the scanf statement. This scanf statement reads in from the command prompt a value. Here, we use %d to indicate that the user will enter an integer, and we use the ampersand and the variable name BirthYear, which means that the computer will be expecting the user to enter an integer, and the computer will store that value in the variable, BirthYear.

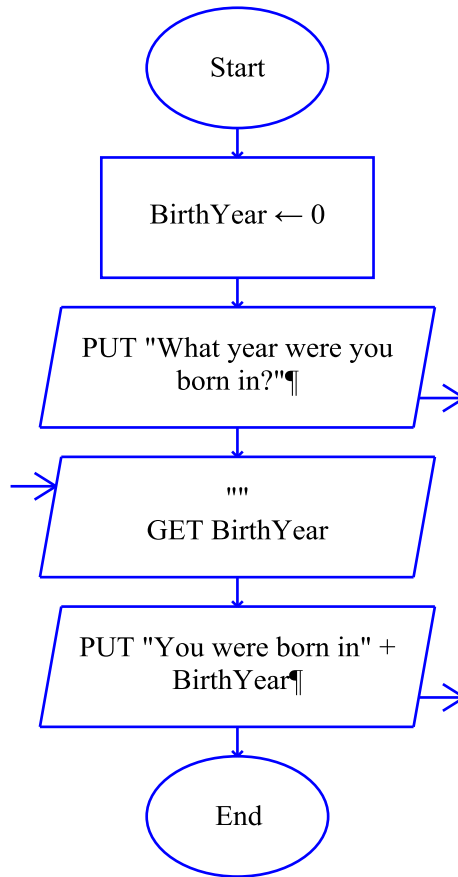


Figure B.12 Slide 25F, displaying a sample program that uses User Input.

Table B.11 Slide 26 Flow and Slides 28-29 Text scripts.

Slide 26 Flow Script	Slide 28 Text Script
So we find the Start shape, and follow the arrow.	So we find the main, and then read the next line.
	Slide 29 Text Script
	We read the opening curly bracket, indicating that we are going to start a program.

27F, 30T. So we set up the variable BirthYear. {28F, 31T} Then ask our question. 29F, 32T. “What year were you born in?” as shown here, and the computer politely waits until I do something. {30F, 33T} So I type in 1980, and press enter.

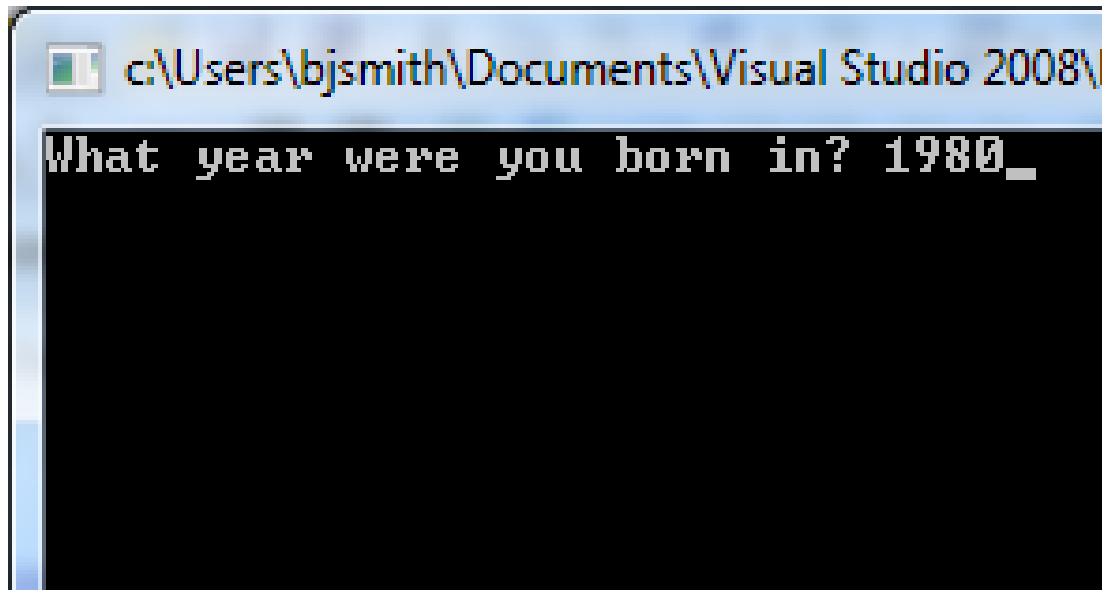


Figure B.13 Slide 30F, displaying the results of our program that demonstrates user input.

31F, 34T. Then we read in what the user has entered, and put that value in BirthYear.

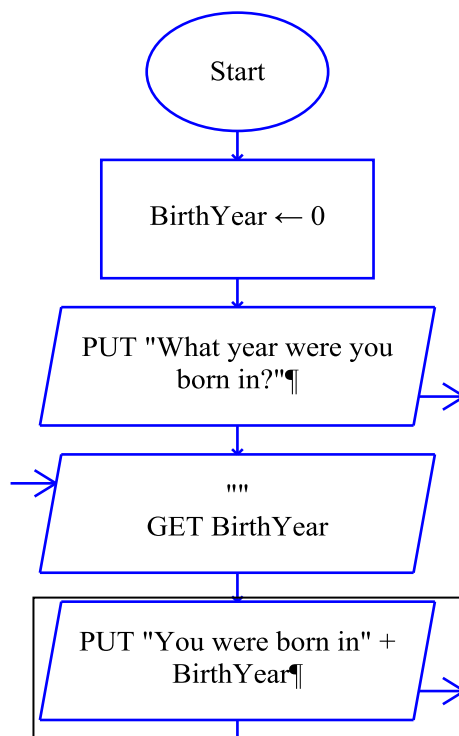


Figure B.14 Slide 31F, PUT statement highlighted.

32F, 35T. And then it prints out the year. 33F, 36T. “You were born in 1980!”

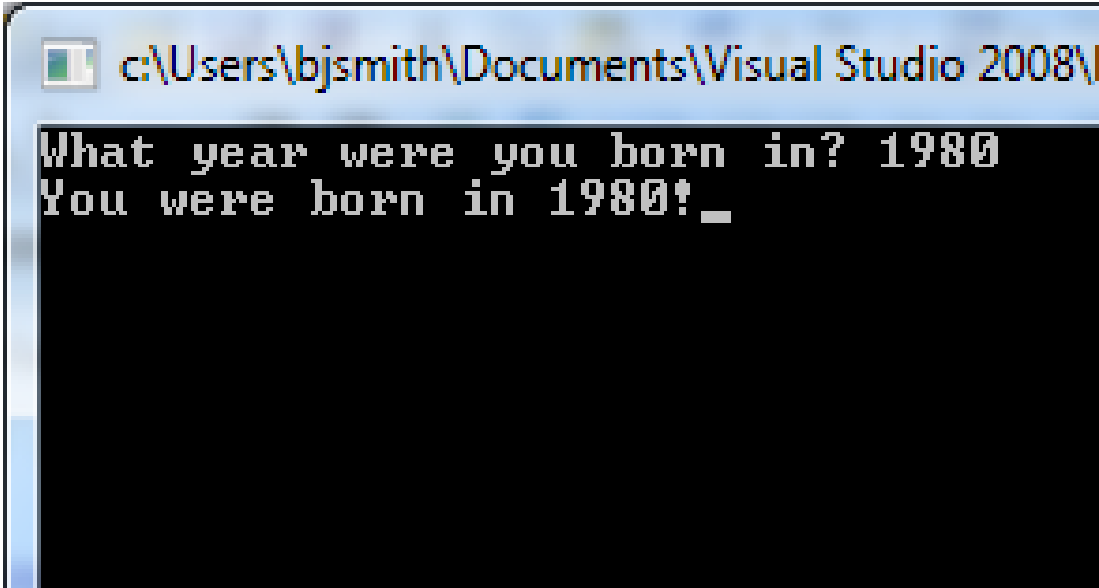


Figure B.15 Slide 33F, displaying when the user was born.

Table B.12 Slide 34 Flow and Slide 37 Text scripts.

Slide 34 Flow Script	Slide 37 Text Script
Following the arrows, we arrive at the “End.”	Reading the next line, we find the end of the program.

35F, 38T. So far everything has been done in order. Well now we are going to add a choice to the program.

B.6Alternative

Table B.13 Slide 36 Flow and Slide 39 Text scripts.

Slide 36 Flow Script	Slide 39 Text Script
The choice will be indicated with this wide hexagon. Notice that there are two arrows that point out of this hexagon. If what is inside of the hexagon is true, then	The choice, or alternative, will be performed with an “if” statement, and it takes the form if and then a conditional, and the conditional will either be true or false. The conditional is

we follow the arrow to the left that says “Yes,” and if the statement in the hexagon is false, we follow the arrow to the right that says “No.” Let’s review a program.

located right after the if statement, in parenthesis. If the conditional is true, then whatever statements are in the brackets will be executed. Let’s review a program.

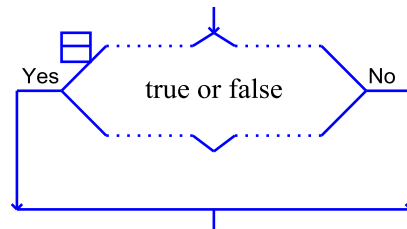


Figure B.16 Slide 36F, displaying the alternative structure.

37F, 40T. This program is going to ask the user what year they were born in, read in that year, then check if that number is negative. If it is negative, then the program will print out a message that you have to type in a positive number and then end. If the year entered is positive, the program will just end.

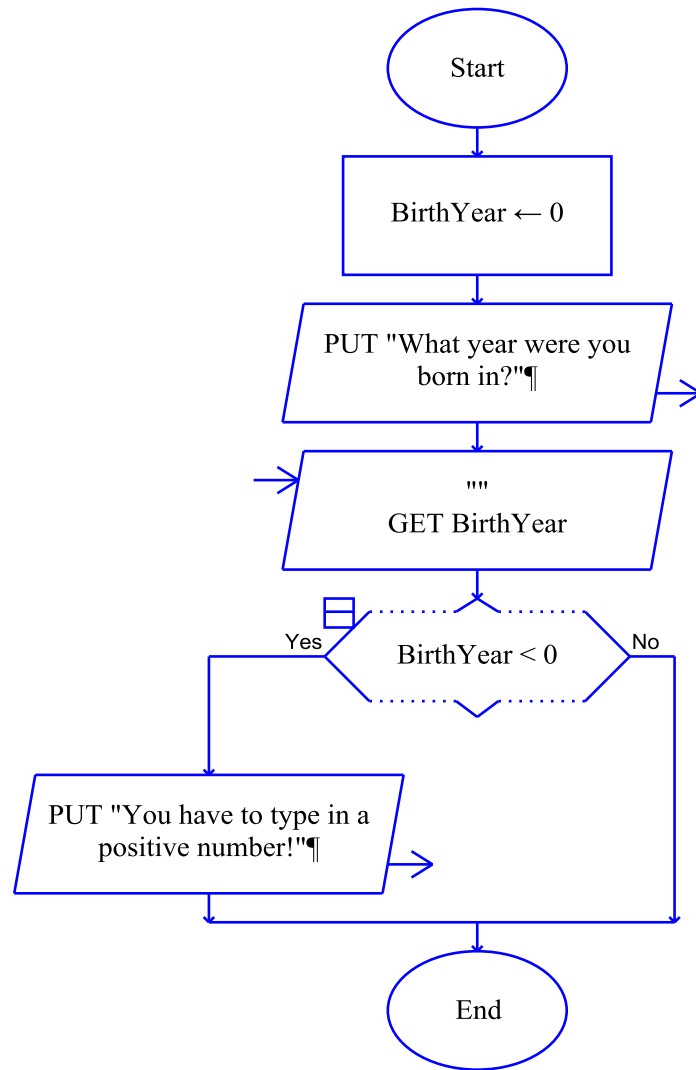


Figure B.17 Slide 37F, a sample program using Alternative.

Table B.14 Slide 38 Flow and Slides 41-42 Text scripts.

Slide 38 Flow Script	Slide 41 Text Script
We begin with Start.	We begin with main.
	Slide 42 Text Script
	Read the opening curly bracket.

39F, 43T. And here is our BirthYear variable that we have set to 0. 40F, 44T.

And here we ask the question “What year were you born in?” 41F, 45T.As shown. 42F,

46T. And we type in -5, because we want this program to be able to handle cases where the user types in something that might not be what the program is expecting.

Table B.15 Slides 43-44 Flow and Slides 47-49 Text scripts.

Slide 43 Flow Script	Slide 47 Text Script
Here, it GETS -5 and associates it with BirthYear.	Here, it reads -5 and associates it with BirthYear with the command scanf.
Slide 44 Flow Script	Slide 48 Text Script
It checks, is BirthYear less than 0. It is, so we follow the Yes arrow.	It checks, is BirthYear less than 0? It is, so the statements inside of the next brackets are executed.
	Slide 49 Text Script
	It finds the brackets, which means that whatever is inside the brackets gets executed if BirthYear is less than zero, or whatever is inside the brackets gets skipped if BirthYear is not less than zero.

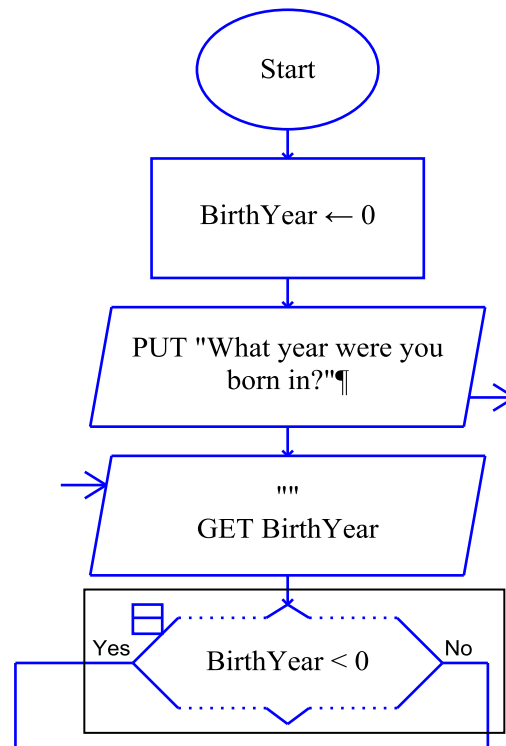


Figure B.18 Slide 44F, highlighting the alternative structure in use.

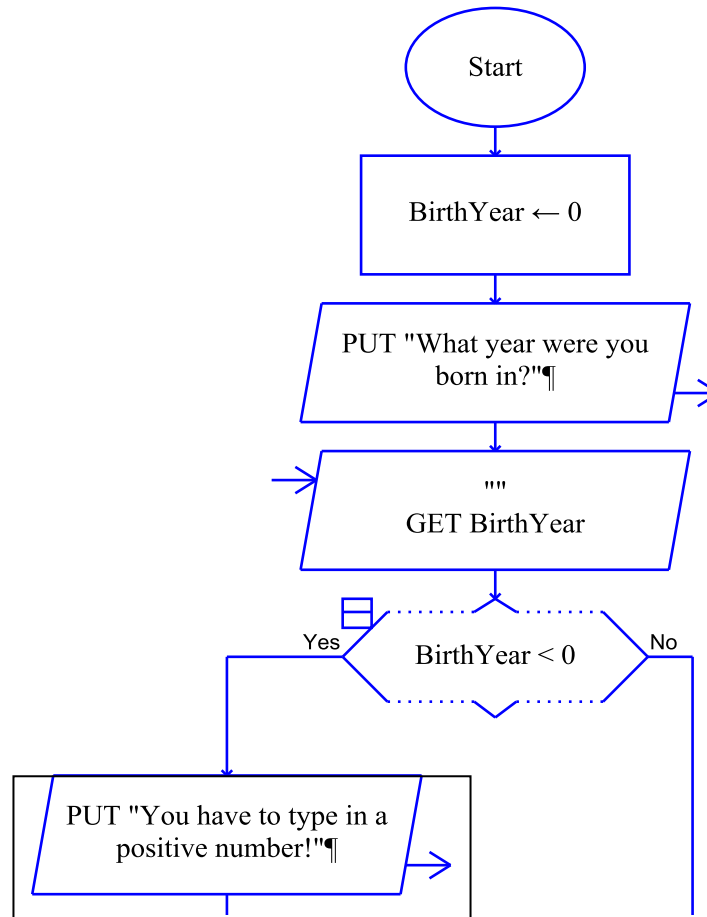


Figure B.19 Slide 45F, highlighting the PUT statement.

45F, 50T. And it prints out this statement. 46F, 51T. “You have to type in a positive number!” 52T. Reading the next line and finding the if’s closing bracket.

Table B.16 Slides 47-48 Flow and Slides 53-54 Text scripts.

Slide 47 Flow Script	Slide 53 Text Script
The computer program continues reading the arrows and finds the “End.”	And finally reading the next line and finding the program’s closing bracket.
Slide 48 Flow Script	Slide 54 Text Script
Now we are going to embellish on the hexagon. If the hexagon has a statement that is not true, then we now follow to the right, towards the labeled “No” and execute that statement or set of statements.	Now we are going to add to the if statement. If the if statement’s conditional is not true, then we have a command that will execute another statement or set of statements.

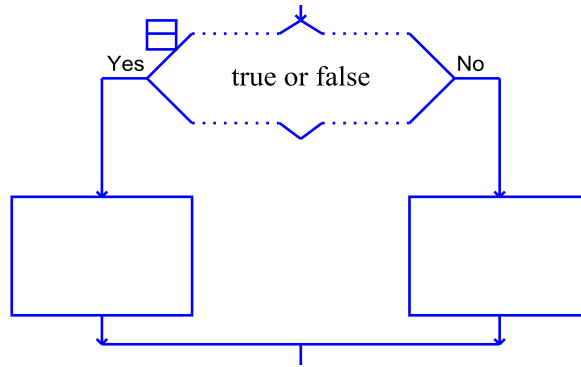


Figure B.20 Slide 48F, displaying the alternative structure.

49F, 55T. This program will create a variable, BirthYear, set its value to 0, then ask the user “What year were you born in?” The program then reads what the user types in, and checks what the user typed in. If the value is less than 1980, then the program prints out “You are old!” and then ends. If the value is not less than 1980, then the program prints out “You are young!” and then ends.

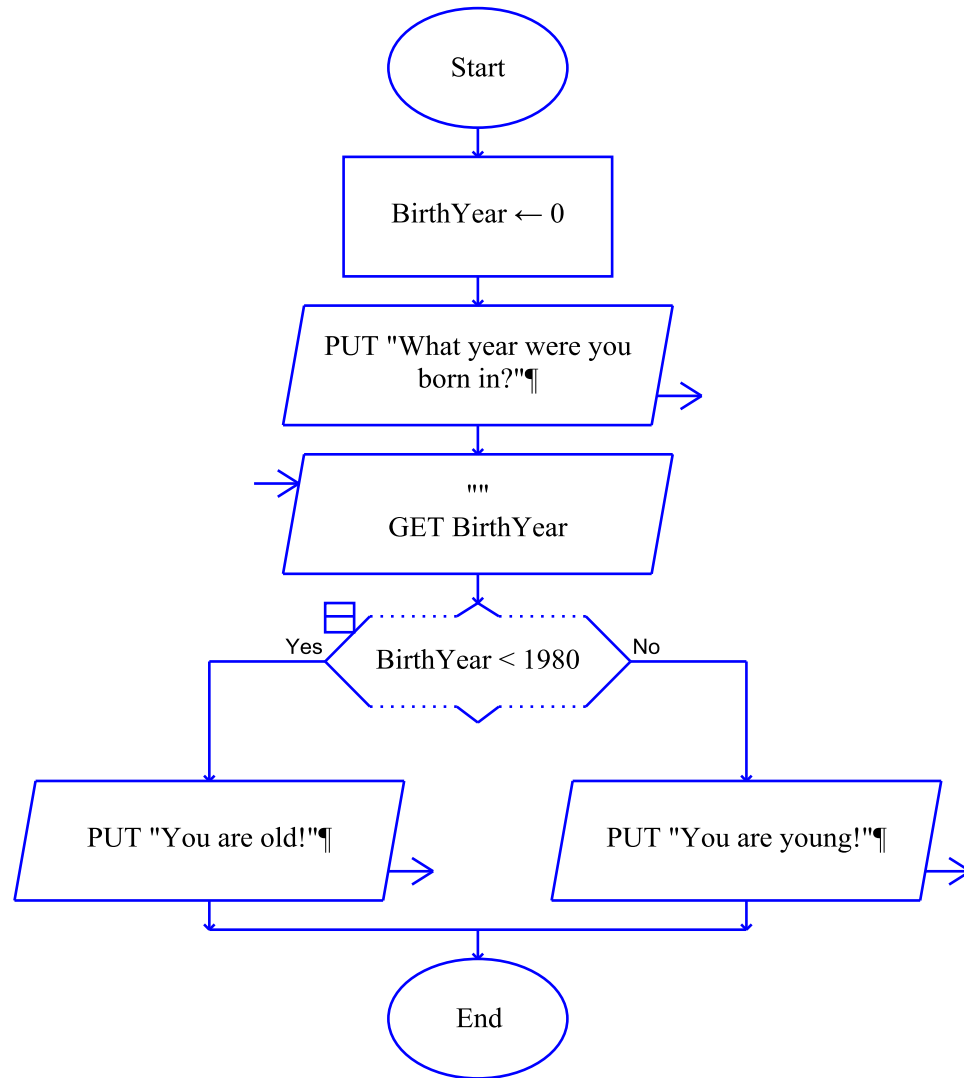


Figure B.21 Slide 49F, displaying a more advanced alternative structure in a program.

Table B.17 Slide 50 Flow and Slides 56-57 Text scripts.

Slide 50 Flow Script	Slide 56 Text Script
We begin at Start.	Again, when running a program, things go in order and we begin at main.
	Slide 57 Text Script
	And we read the next line, an opening curly bracket, indicating that a program is beginning.

51F, 58T. Then we create BirthYear and set BirthYear's value to 0. 52F, 59T. And we ask the user what year they were born in. 53F, 60T. As shown. 54F, 61T. And the user types in 1990. 55F, 62T. And then the program reads in what the user typed in, which is 1990. 56F, 63T. So next the computer checks is 1990 less than 1980? And the answer is no, 1990 is not less than 1980.

Table B.18 Slide 57 Flow and Slides 64-66 Text scripts.

Slide 57 Flow Script	Slide 64 Text Script
So we follow the "No" side of the conditional since the conditional is not true and print to the screen "You are young!"	Now else executes since the conditional is not true.
	Slide 65 Text Script
	So the program proceeds to the next line which is a bracket
	Slide 66 Text Script
	And executes the command found there, which here means print to the screen "You are young!"

58F, 67T. As shown. 68T. And the software reads the next line looking for a command.

Table B.19 Slides 59-60 Flow and Slides 69-70 Text scripts.

Slide 59 Flow Script	Slide 69 Text Script
So the program follows the arrows, and the next command it finds is the End, so that's the end.	Finding just a closing brace, it goes on to the next line, which indicates the end of the program.
Slide 60 Flow Script	Slide 70 Text Script
So far, we know that programs execute in order, and that an if statement controls which direction we travel while running a program and which direction gets skipped. Now, we are going to learn a new concept that allows code to be repeated.	So far, we know that programs execute in order, and that an if statement controls if certain lines are executed or skipped. Now, we are going to learn a new concept that allows a certain line or group of lines to be repeated.

B.7 Iteration

Table B.20 Slide 61 Flow and Slide 71 Text scripts.

Slide 61 Flow Script	Slide 71 Text Script
Just like an <i>if</i> statement, a while statement has a conditional that is either true or false. If the conditional is false, then we follow the path straight down and execute all the commands that we encounter, and then we follow the arrow back to the conditional, where we check its value again. We repeat this until the conditional is true, at which time we follow the path to the left, and do not repeat anymore. Let's look at an example.	Just like an <i>if</i> statement, a while statement has a conditional that is either true or false. If the conditional is true, then the statements inside of the brackets are executed, and if the conditional is false, then the computer skips the brackets. A while statement is different from an if statement in that once all of the statements inside of a while statement are read, the program returns to the conditional, checks its value, and may repeat those lines until the conditional is false. Let's look at an example.

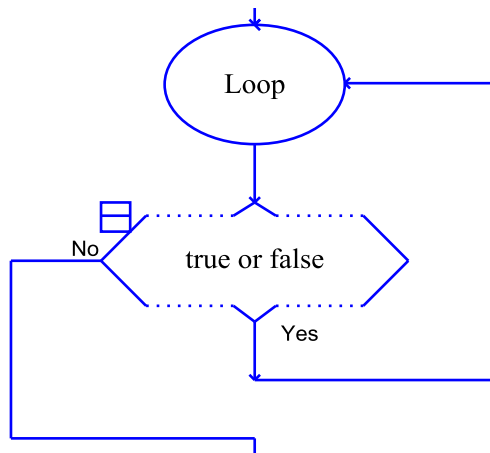


Figure B.22 Slide 61F, displaying the looping structure.

62F, 72T. This is our sixth and final program, where we introduce a while loop which will allow for repetition.

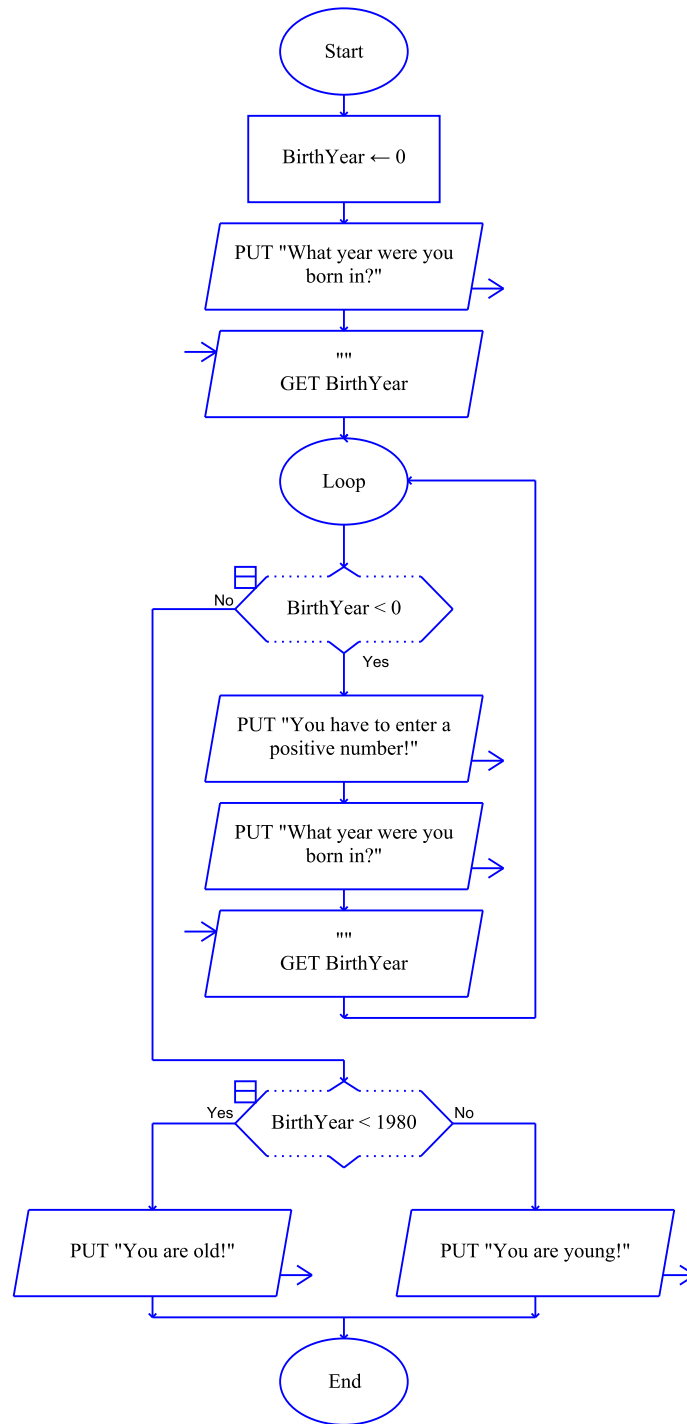


Figure B.23 Slide 62F, displaying the looping structure in a program.

63F, 73T. This is our familiar variable BirthYear. 64F, 74T. The computer asks “What year were you born in?” 65F, 75T. As shown. 66F, 76T. And we type in -10.

67F, 77T. The computer reads in -10 and puts that value in BirthYear. 68F. This loop statement is going to be used as a place to come back to.

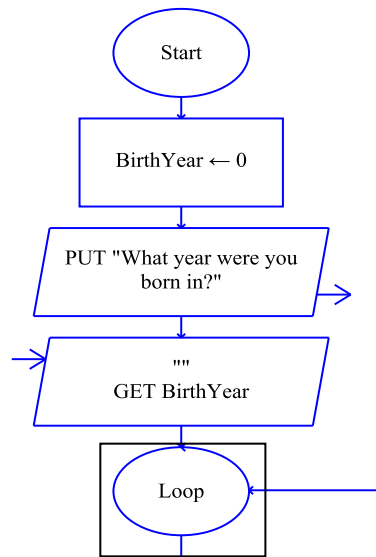


Figure B.24 Slide 68F, highlighting the Loop shape.

Table B.21 Slide 69 Flow and Slides 78-79 Text scripts.

Slide 69 Flow Script	Slide 78 Text Script
We are in a conditional, so we check, is BirthYear greater than 0? The answer is no, that BirthYear is not greater than 0, so we follow the No path down.	We check, is BirthYear less than 0? The answer is yes, that BirthYear is less than 0. So since we are in a while command, we look for the next bracket, and execute the statements that are between the next opening bracket and the next closing bracket.
	Slide 79 Text Script
	We find the opening bracket.

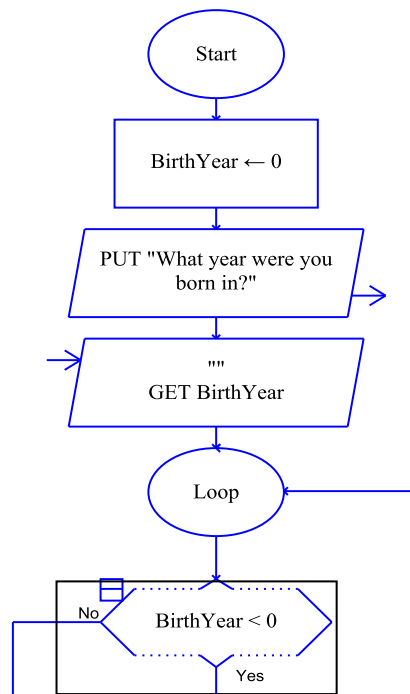


Figure B.25 Slide 69F, highlighting the conditional.

70F, 80T. And execute the next statement, which says “You have to enter a positive number!”

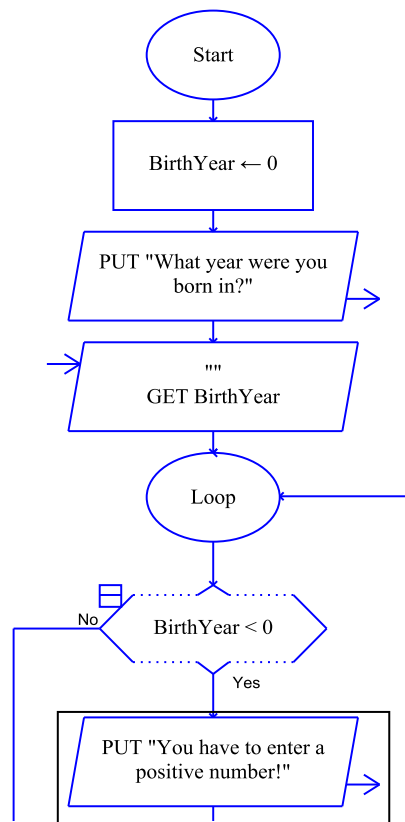


Figure B.26 Slide 70F, highlighting the first shape in the Yes branch.

71F, 81T. As shown. 72F, 82T. So the computer asks, “What year were you born in?”

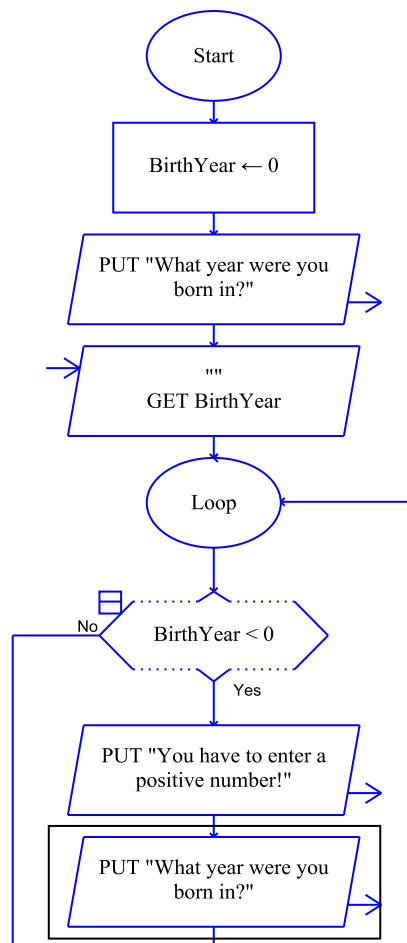


Figure B.27 Slide 72F, highlighting the second shape in the Yes branch.

73F, 83T. As shown. 74F, 84T. And we type in -5 to really show the power of the while loop. 75F, 85T. We read in the value of -5 and associate that with BirthYear. 86T. We find the closing bracket.

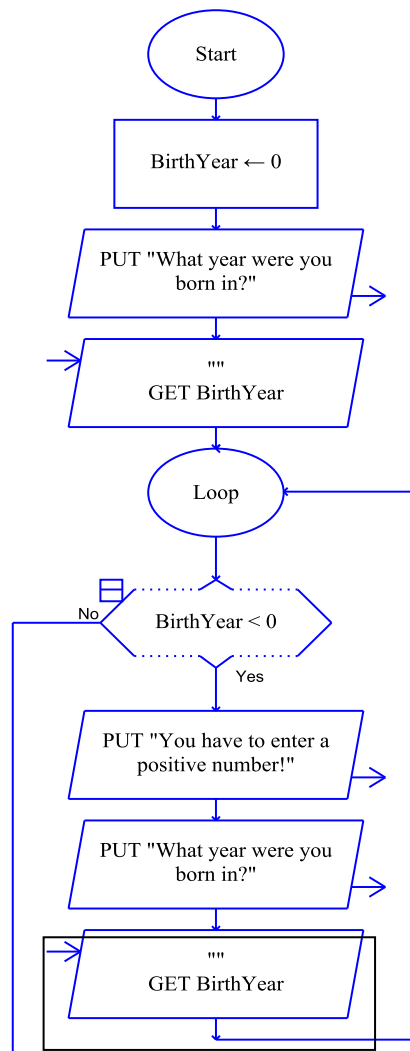


Figure B.28 Slide 75F, highlighting the third shape in the Yes branch.

76F. And we follow the arrows and go back to Loop.

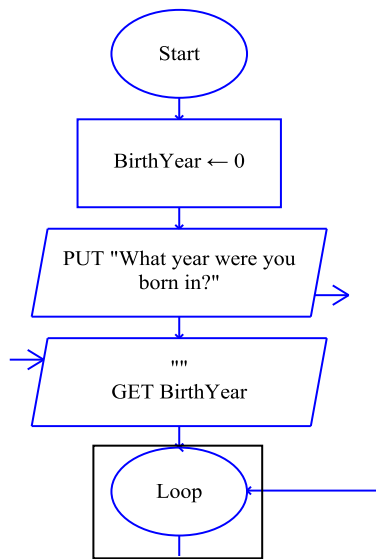


Figure B.29 Slide 76F, highlighting the Loop shape again.

Table B.22 Slide 77 Flow and Slides 87-88 Text scripts.

Slide 77 Flow Script	Slide 87 Text Script
BirthYear, with a value of -5, is less than 0, so this is true, and we follow the Yes arrow.	And go back up to the while loop. BirthYear, with a value of -5, is less than 0, so this is true again. And we execute what is in the brackets.
	Slide 88 Text Script
	So we find the opening bracket.

78F, 89T. And execute the next command, which prints out “You have to enter a positive number!” 79F, 90T. As shown. 80F, 91T. We read the next command and print out “What year were you born in?” 81F, 92T. As shown. 82F, 93T. And now we type in a useful number, 1990. 83F, 94T. And we associate 1990 with BirthYear.

Table B.23 Slides 84-85 Flow and Slides 95-96 Text scripts.

Slide 84 Flow Script	Slide 95 Text Script
And we follow the arrows back to loop.	And we read the next line and find the closing bracket, so we return to the top.
Slide 85 Flow Script	Slide 96 Text Script
Now, is BirthYear less than 0? Is 1990 less than 0? No it is not. So the Loop statement concludes here, and we follow the arrows.	Now, is BirthYear less than 0? Is 1990 less than 0? No it is not. So the while statement concludes here, and we skip to the end of the brackets associated with the while statement.

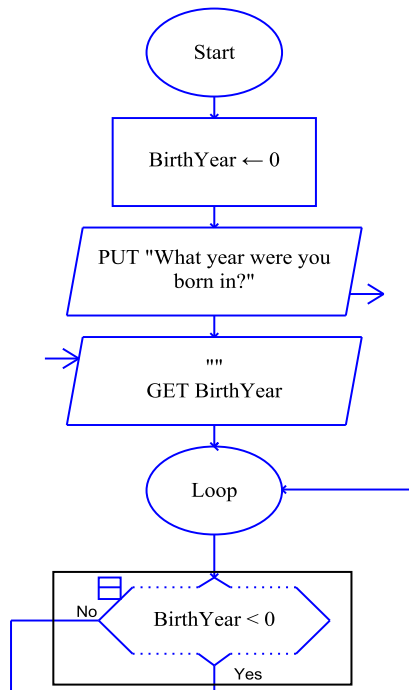


Figure B.30 Slide 85F, highlighting the conditional again.

Table B.24 Slide 86 Flow and Slides 97-99 Text scripts.

Slide 86 Flow Script	Slide 97 Text Script
Now, is BirthYear less than 1980? Is 1990 < 1980? No, it is not, so we follow the “No” path.	Now, is BirthYear less than 1980? Is 1990 < 1980? No, it is not, so this if statement is false, and we drop down to the else statement, and execute whatever is in those brackets.
	Slide 98 Text Script
	This is the else statement.
	Slide 99 Text Script
	And here is the opening bracket.

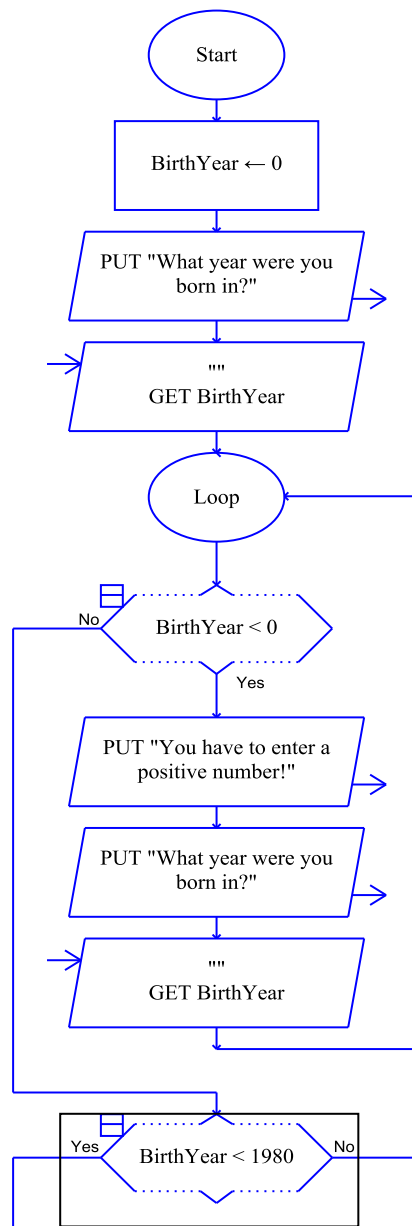


Figure B.31 Slide 86F, highlighting the conditional.

87F, 100T. And here we print out “You are young!”

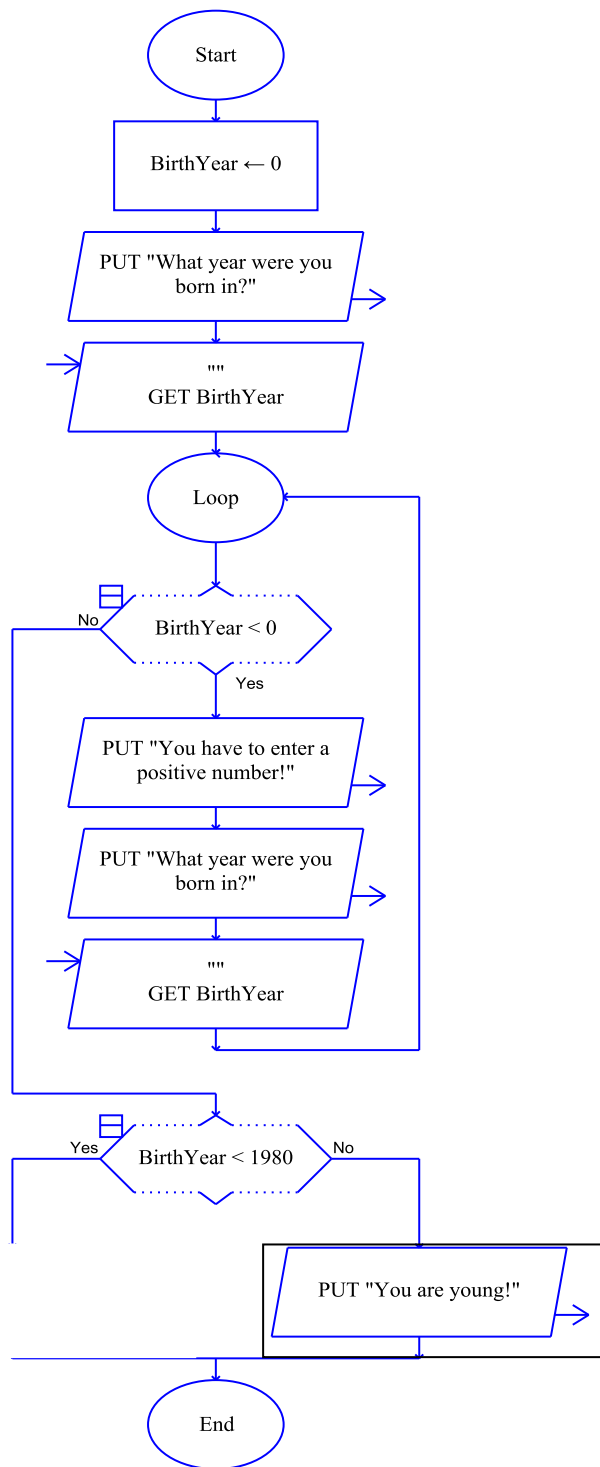


Figure B.32 Slide 87F, highlighting the No branch in the conditional.

88F, 101T. As shown.

Table B.25 Slide 89 Flow and Slides 102-103 Text scripts.

Slide 89 Flow Script	Slide 102 Text Script
And we follow the arrows to the end.	And now we have encountered the closing bracket of the else statement.
	Slide 103 Text Script
	And here we have found the closing bracket of the entire program.

Appendix C

The Concepts Presented to Participants in the Second Pilot and Experiment

The following figures represent the slides that were presented to the Flow and Text groups during both the second pilot and the full experiment. When appropriate, we have put corresponding slides on the same page for easier comparison.

Visual Computer Programming Study

By: B.J. Smith

Figure C.1 Slide 1 – Flow.

Textual Computer Programming Study

By: B.J. Smith

Figure C.2 Slide 1 - Text.

Agenda

- Fill out the Survey and Index of Learning Styles Questionnaire
- Introduce Computer Science Concepts
- Test these concepts

Figure C.3 Slide 2 – Flow and Text.

A Computer Program

- Executed in order, from the beginning to the end
- Written in an exact language that the computer understands
 - English does not work
 - Herb roasted chicken.
 - Fruit flies like an apple.
- Some popular computer languages are the family of C languages (C/C++/C#), Java, Python

Figure C.4 Slide 3 – Flow and Text.

Sequence

- Programs are executed in order.
- Until a line has been reached, the program does not know about it.

Figure C.5 Slide 4 – Flow and Text.

1st Program

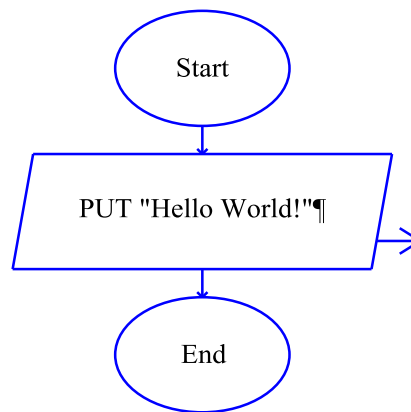


Figure C.6 Slide 5 – Flow.

1st Program

```
void main(void)
{
    printf("Hello World!");
}
```

Figure C.7 Slide 5 - Text.

1st Program

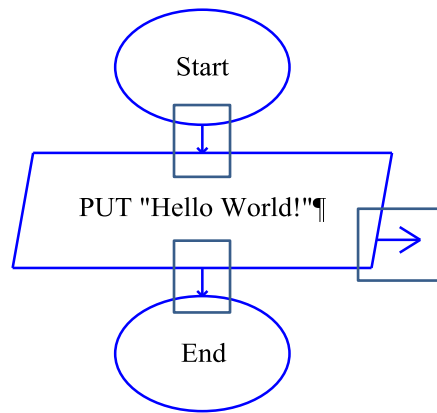


Figure C.8 Slide 6 – Flow.

1st Program

```
void main(void)
{
    printf("Hello World!");
}
```

Figure C.9 Slide 6 - Text.

1st Program

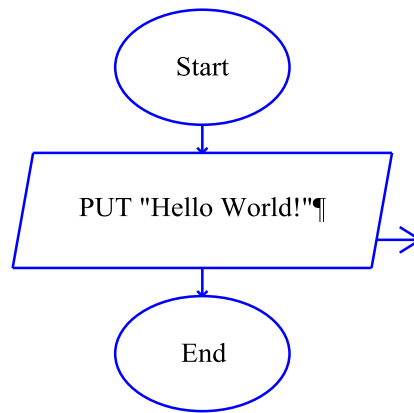


Figure C.10 Slide 7 – Flow.

1st Program

```
void main(void)
{
    printf("Hello World!");
}
```

Figure C.11 Slide 7 - Text.

1st Program

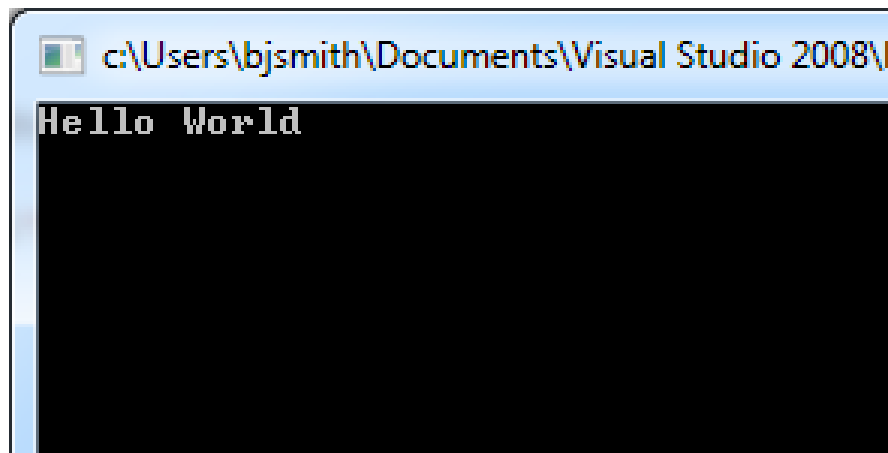


Figure C.12 Slide 8 – Flow and Text.

1st Program

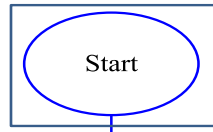


Figure C.13 Slide 9 – Flow.

1st Program

```
void main(void)
```

Figure C.14 Slide 9 - Text.

1st Program

```
void main(void)
```

```
{
```

Figure C.15 Slide 10 - Text.

1st Program

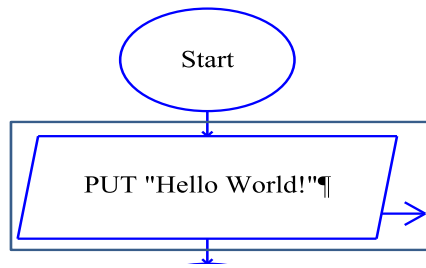


Figure C.16 Slide 10 – Flow.

1st Program

```
void main(void)
{
    printf("Hello World!");
}
```

Figure C.17 Slide 11 - Text.

1st Program

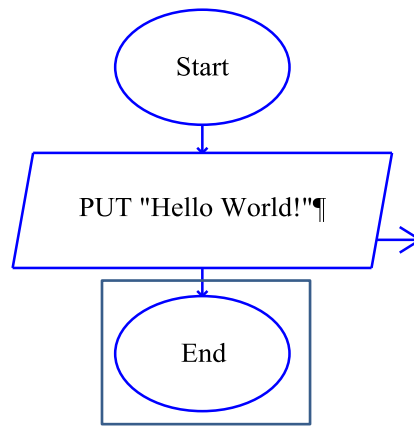


Figure C.18 Slide 11 – Flow.

1st Program

```
void main(void)
{
    printf("Hello World!");
}
```

Figure C.19 Slide 12 - Text.

1st Program

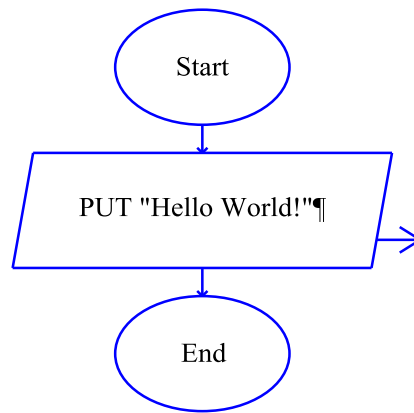


Figure C.20 Slide 12 – Flow.

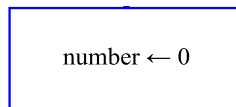
1st Program

```
void main(void)
{
    printf("Hello World!");
}
```

Figure C.21 Slide 13 - Text.

variable

- Variable – symbolic name associated with a value and whose associated value may be changed, or *vary*
 - Has a type (implied)
 - Can only have a value if the variable is explicitly set (on the left side of an arrow sign \leftarrow)
 - If it is not explicitly set, then the variable's value is unknown



```
number ← 0
```

Figure C.22 Slide 13 – Flow.

variable

- Variable – symbolic name associated with a value and whose associated value may be changed, or *vary*
 - Has a type
 - Can only have a value if the variable is explicitly set (on the left side of an equal sign =)
 - If it is not explicitly set, then the variable's value is unknown

```
int number = 0;
```

Figure C.23 Slide 14 - Text .

2nd Program

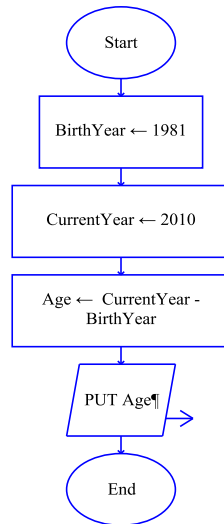


Figure C.24 Slide 14 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
    int age = CurrentYear - BirthYear;
    printf("%d", age);
}
```

Figure C.25 Slide 15 - Text.

2nd Program

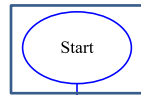


Figure C.26 Slide 15 – Flow.

2nd Program

```
void main(void)
```

Figure C.27 Slide 16 - Text.

2nd Program

```
void main(void)
```

```
{
```

Figure C.28 Slide 17 - Text.

2nd Program

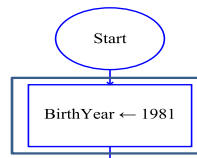


Figure C.29 Slide 16 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
```

Figure C.30 Slide 18 - Text.

2nd Program

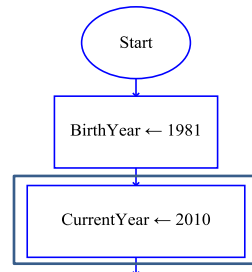


Figure C.31 Slide 17 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
```

Figure C.32 Slide 19 - Text.

2nd Program

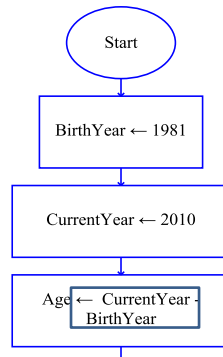


Figure C.33 Slide 18 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
    int age = CurrentYear - BirthYear;
```

Figure C.34 Slide 20 - Text.

2nd Program

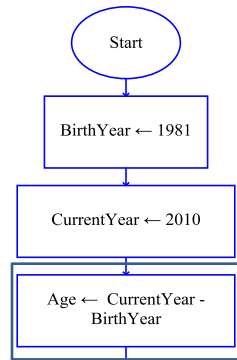


Figure C.35 Slide 19 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
    int age = CurrentYear - BirthYear;
```

Figure C.36 Slide 21 - Text.

2nd Program

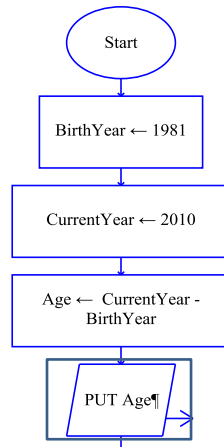


Figure C.37 Slide 20 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
    int age = CurrentYear - BirthYear;
    printf("%d", age);
}
```

Figure C.38 Slide 22 - Text.

2nd Program

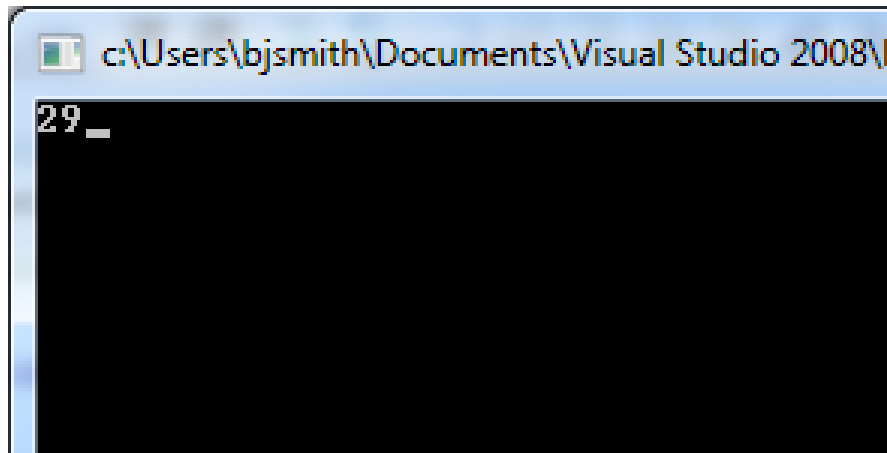


Figure C.39 Slide 21 – Flow and Slide 23 - Text.

2nd Program

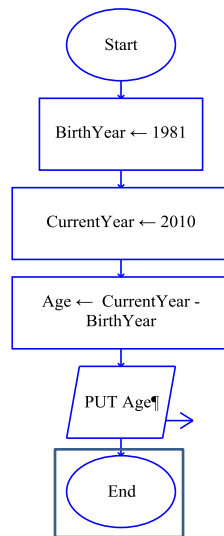


Figure C.40 Slide 22 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
    int age = CurrentYear – BirthYear;
    printf(“%d”, age);
}
```

Figure C.41 Slide 24 - Text.

2nd Program

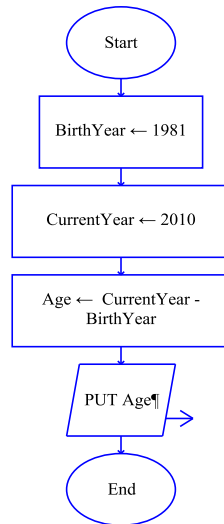


Figure C.42 Slide 23 – Flow.

2nd Program

```
void main(void)
{
    int BirthYear = 1981;
    int CurrentYear = 2010;
    int age = CurrentYear - BirthYear;
    printf("%d", age);
}
```

Figure C.43 Slide 25 - Text.

User input

- Usually consists of two parts:
 - PUT, informing the user of something
 - GET, reading a value from the command line

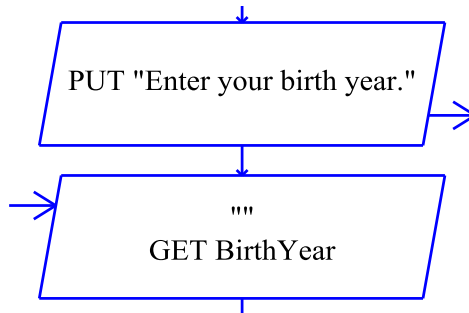


Figure C.44 Slide 24 – Flow.

User input

- Usually consists of two parts:
 - a printf, informing the user of something
 - a scanf, reading a value from the command line

```
printf("Enter your birth year.");  
scanf("%d", &BirthYear);
```

Figure C.45 Slide 26 - Text.

3rd Program

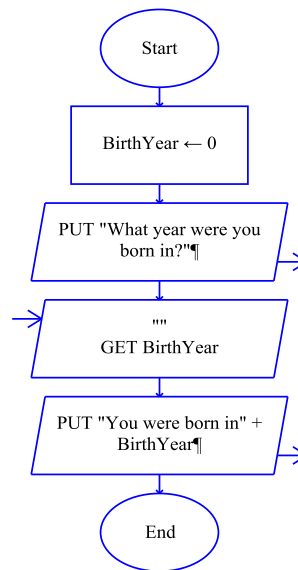


Figure C.46 Slide 24 – Flow.

3rd Program

```
void main(void)
{
    int BirthYear=0;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    printf("You were born in %d!", age);
}
```

Figure C.47 Slide 27 - Text.

3rd Program

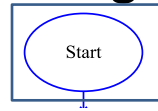


Figure C.48 Slide 25 – Flow.

3rd Program

```
void main(void)
```

Figure C.49 Slide 28 - Text.

3rd Program

```
void main(void)
```

```
{
```

Figure C.50 Slide 29 - Text.

3rd Program

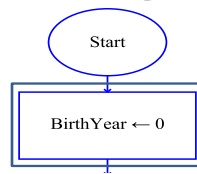


Figure C.51 Slide 27 – Flow.

3rd Program

```
void main(void)
```

```
{
```

```
    int BirthYear=0;
```

Figure C.52 Slide 30 - Text.

3rd Program

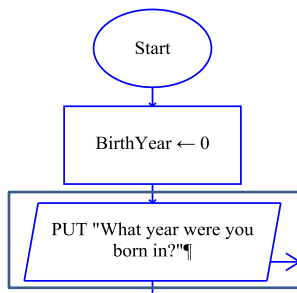


Figure C.53 Slide 28 – Flow.

3rd Program

```
void main(void)
{
    int BirthYear=0;
    printf("What year were you born in?");
```

Figure C.54 Slide 31 - Text.

3rd Program

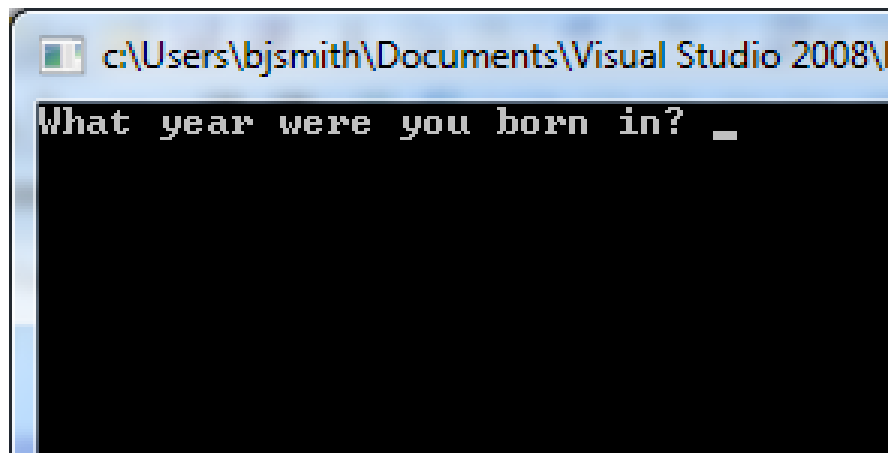


Figure C.55 Slide 29 – Flow and Slide 32 – Text.

3rd Program

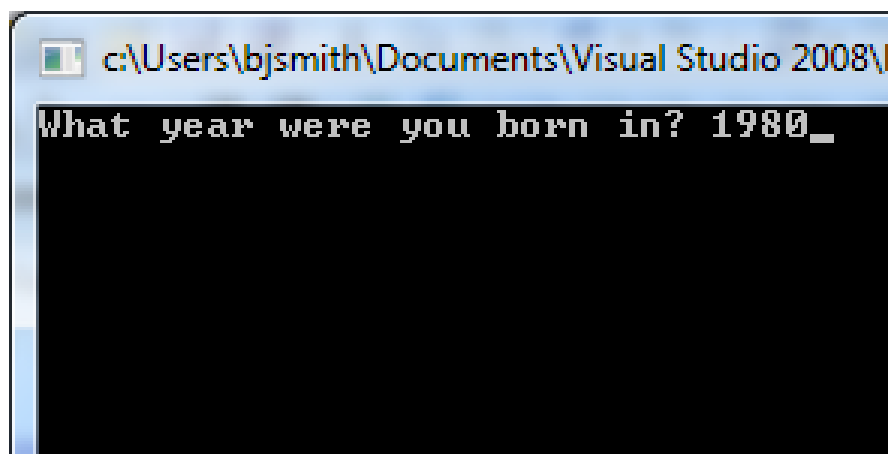


Figure C.56 Slide 30 – Flow and Slide 33 – Text.

3rd Program

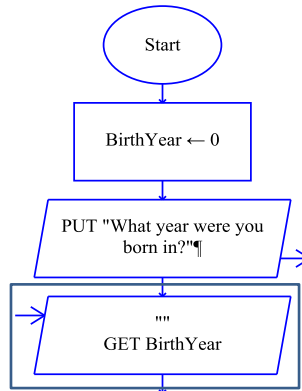


Figure C.57 Slide 31 – Flow.

3rd Program

```
void main(void)
{
    int BirthYear=0;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
```

Figure C.58 Slide 34 - Text.

3rd Program

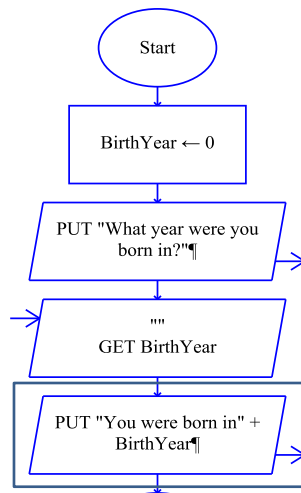


Figure C.59 Slide 32 – Flow.

3rd Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    printf("You were born in %d!", BirthYear);
}
```

Figure C.60 Slide 35 - Text.

3rd Program

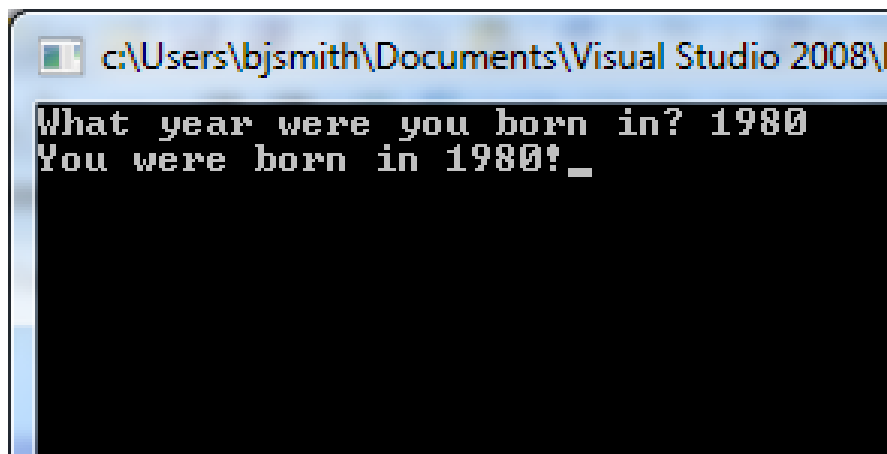


Figure C.61 Slide 33 – Flow and Slide 36 - Text.

3rd Program

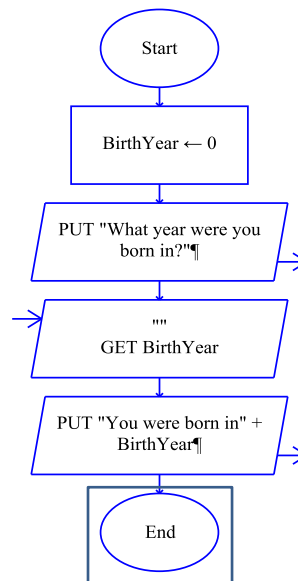


Figure C.62 Slide 34 – Flow.

3rd Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    printf("You were born in %d!", BirthYear);
}
```

Figure C.63 Slide 37 - Text.

3rd Program

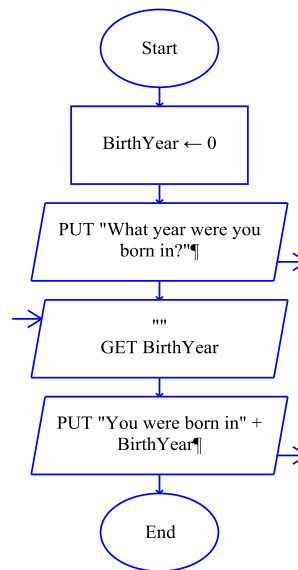


Figure C.64 Slide 35 – Flow.

3rd Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    printf("You were born in %d!", BirthYear);
}
```

Figure C.65 Slide 38 - Text.

Alternative, *if*

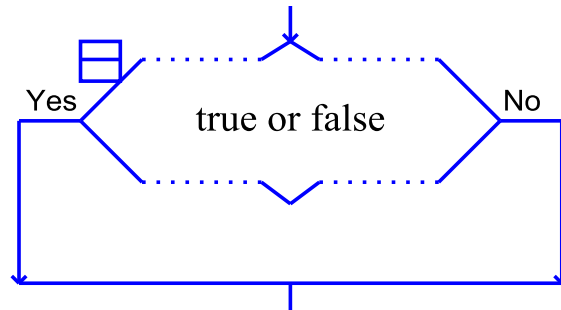


Figure C.66 Slide 36 – Flow.

Alternative, *if*

```
if(conditional)
{
}
```

Figure C.67 Slide 39 - Text.

4th Program

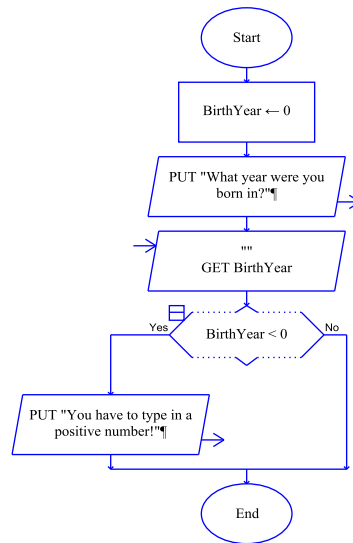


Figure C.68 Slide 37 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 0)
    {
        printf("You have to type in a positive number!");
    }
}
```

Figure C.69 Slide 40 - Text.

4th Program



Figure C.70 Slide 38 – Flow.

4th Program

```
void main(void)
```

Figure C.71 Slide 41 - Text.

4th Program

```
void main(void)
```

```
{
```

Figure C.72 Slide 42 - Text.

4th Program

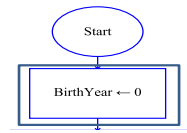


Figure C.73 Slide 39 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
```

Figure C.74 Slide 43 - Text.

4th Program

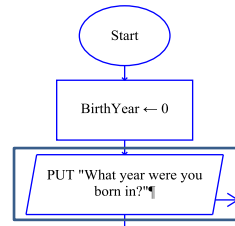


Figure C.75 Slide 40 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
```

Figure C.76 Slide 44 - Text.

4th Program

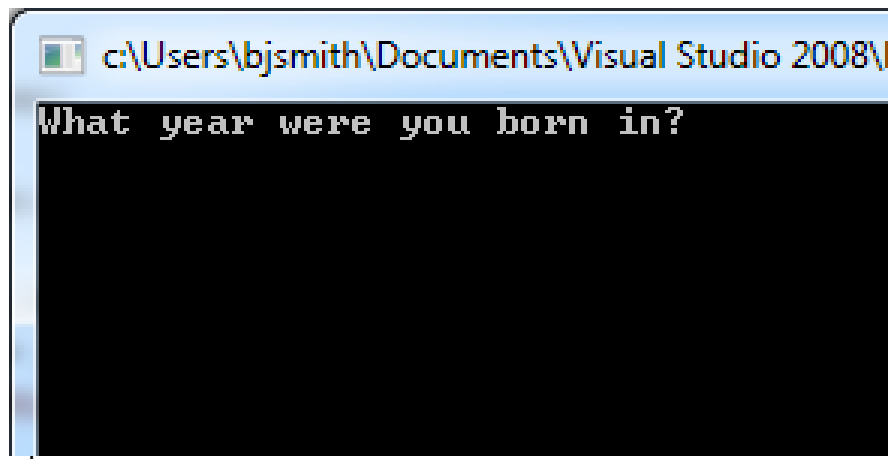


Figure C.77 Slide 41 – Flow and Slide 45 - Text.

4th Program

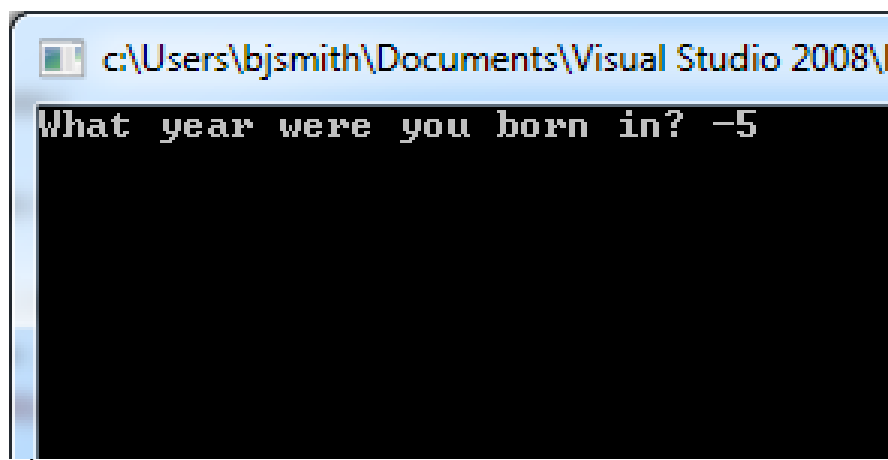


Figure C.78 Slide 42 – Flow and Slide 46 - Text.

4th Program

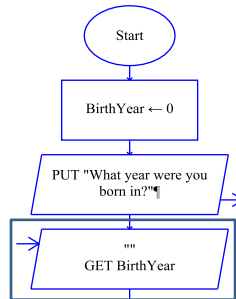


Figure C.79 Slide 43 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
```

Figure C.80 Slide 47 - Text.

4th Program

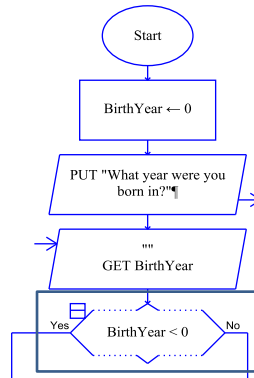


Figure C.81 Slide 44 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 0)
```

Figure C.82 Slide 48 - Text.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 0)
    {
```

Figure C. 83 Slide 49 - Text.

4th Program

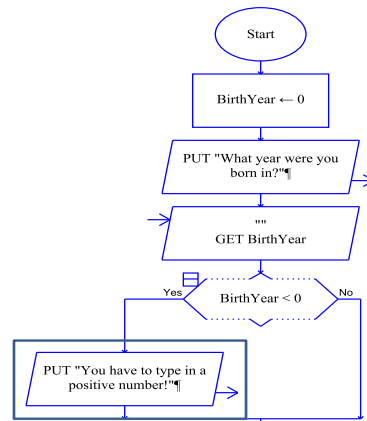


Figure C.84 Slide 45 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 0)
    {
        printf("You have to type in a positive number!");
    }
}
```

Figure C.85 Slide 50 - Text.

4th Program

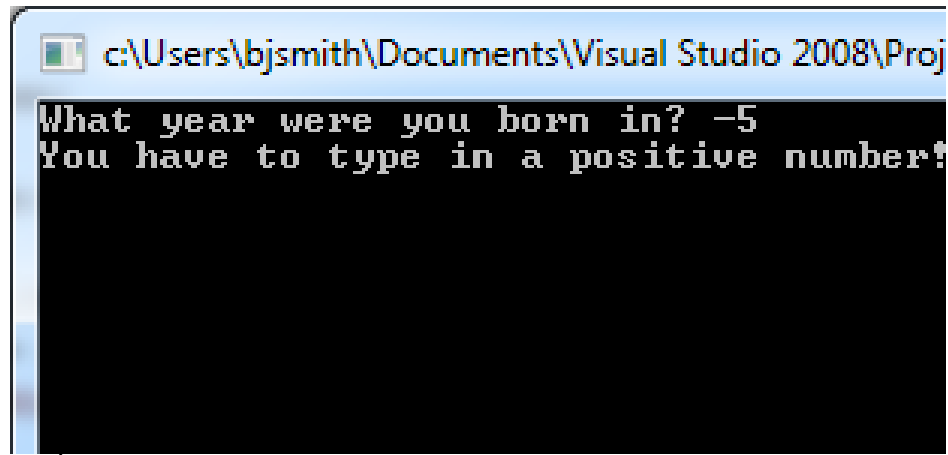


Figure C.86 Slide 46 – Flow and Slide 51 - Text.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 0)
    {
        printf("You have to type in a positive number!");
    }
}
```

Figure C.87 Slide 52 - Text.

4th Program

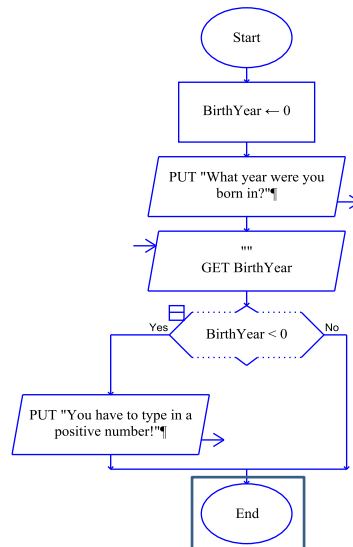


Figure C.88 Slide 47 – Flow.

4th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 0)
    {
        printf("You have to type in a positive number!");
    }
}
```

Figure C.89 Slide 53 - Text.

Alternative, *if else*

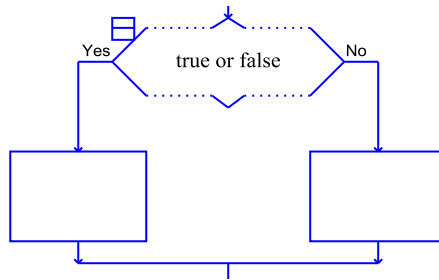


Figure C.90 Slide 48 – Flow.

Alternative, *if else*

```
if(conditional)  
{  
}  
else  
{  
}
```

Figure C.91 Slide 54 - Text.

5th Program

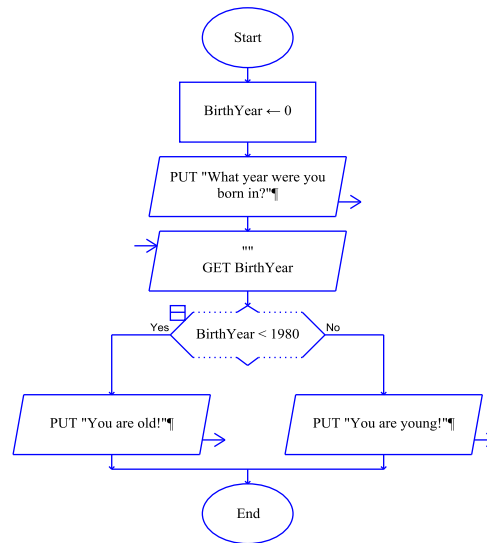


Figure C.92 Slide 49 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {
        printf("You are old!");
    }
    else
    {
        printf("You are young!");
    }
}
```

Figure C.93 Slide 55 - Text.

5th Program

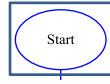


Figure C.94 Slide 50 – Flow.

5th Program

```
void main(void)
```

Figure C.95 Slide 56 - Text.

5th Program

```
void main(void)
```

```
{
```

Figure C.96 Slide 57 - Text.

5th Program



Figure C.97 Slide 51 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
```

Figure C.98 Slide 58 - Text.

5th Program

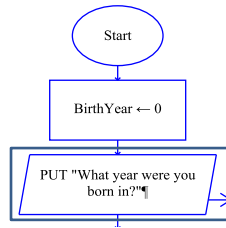


Figure C.99 Slide 52 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
```

Figure C.100 Slide 59 - Text.

5th Program

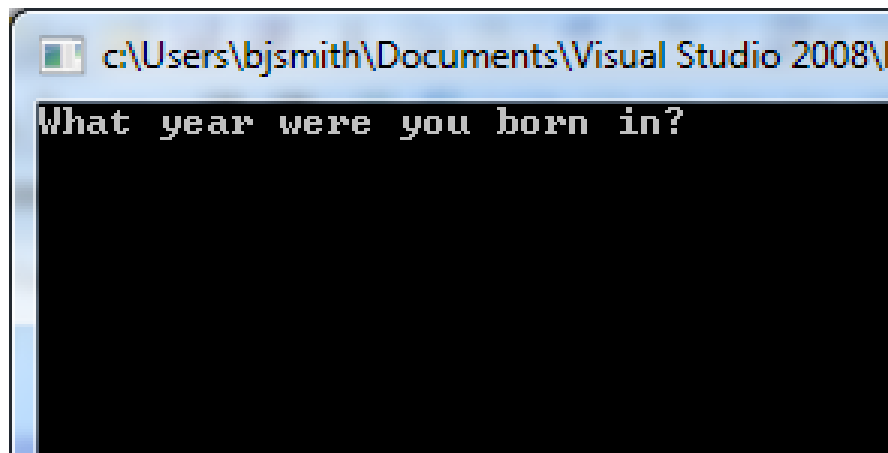


Figure C.101 Slide 53 – Flow and Slide 60 – Text.

5th Program

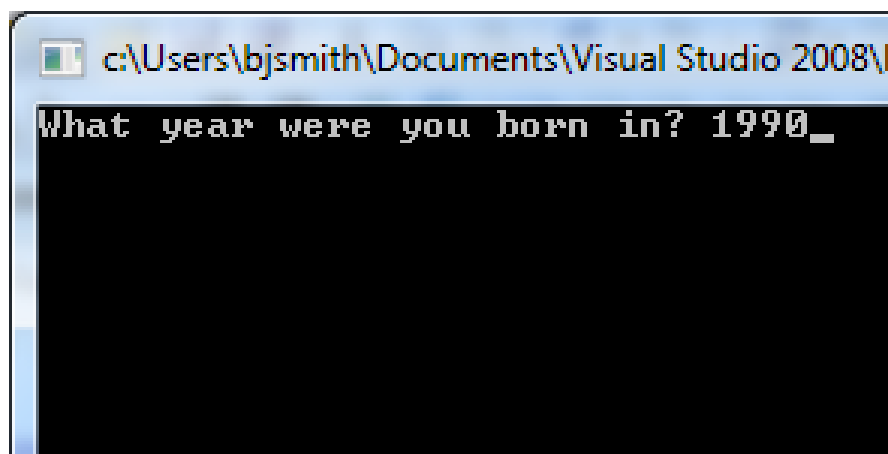


Figure C.102 Slide 54 – Flow and Slide 61 – Text.

5th Program

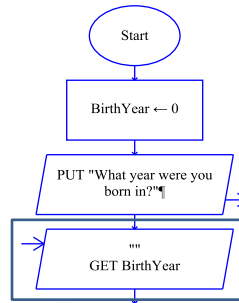


Figure C.103 Slide 55 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
}
```

Figure C.104 Slide 62 - Text.

5th Program

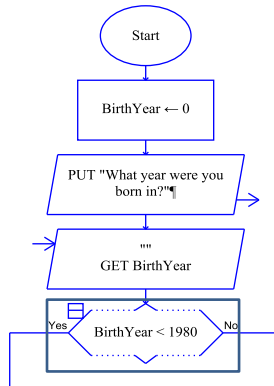


Figure C.105 Slide 56 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
```

Figure C.106 Slide 63 - Text.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {
    }
    else
    {
    }
```

Figure C.107 Slide 64 - Text.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {
    }
    else
    {
    }
```

Figure C.108 Slide 65 - Text.

5th Program

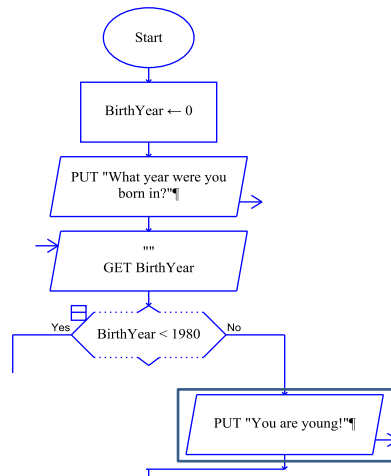


Figure C.109 Slide 57 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {

    }
    else
    {
        printf("You are young!");
    }
}
```

Figure C.110 Slide 66 - Text .

5th Program

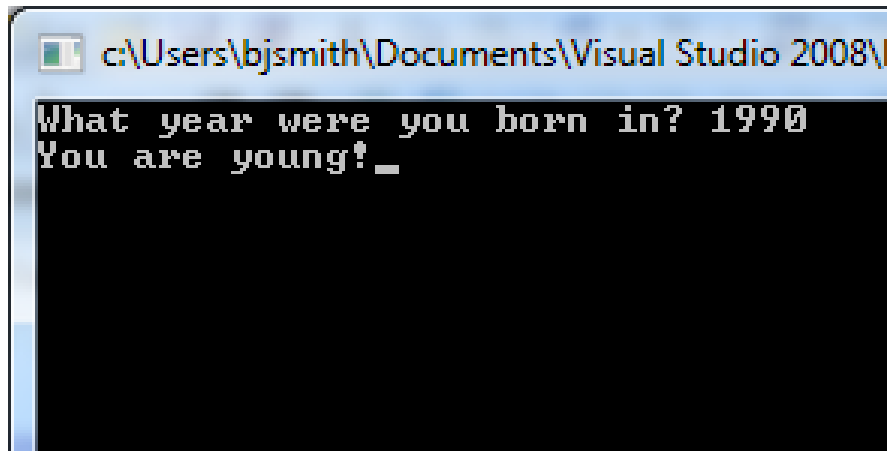


Figure C.111 Slide 58 – Flow and Slide 67 - Text.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {

    }
    else
    {
        printf("You are young!");
    }
}
```

Figure C.112 Slide 68 - Text.

5th Program

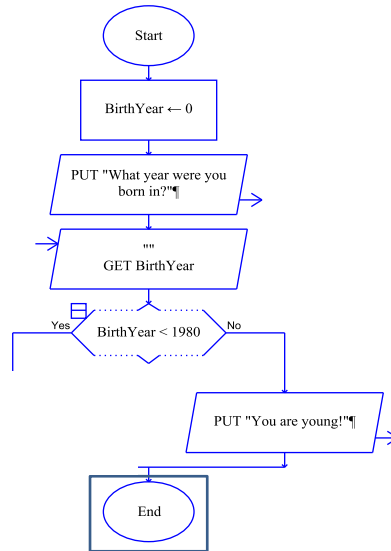


Figure C.113 Slide 59 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {

    }
    else
    {
        printf("You are young!");
    }
}
```

Figure C.114 Slide 69 - Text.

5th Program

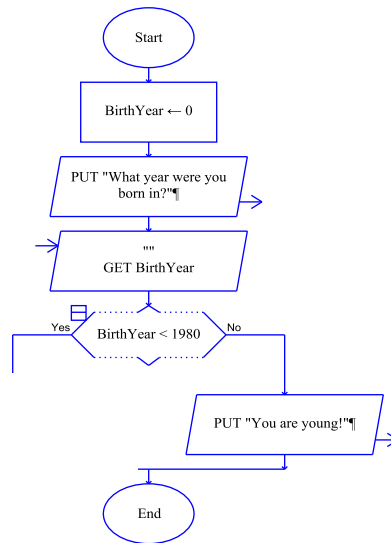


Figure C.115 Slide 60 – Flow.

5th Program

```
void main(void)
{
    int BirthYear;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if(BirthYear < 1980)
    {

    }
    else
    {
        printf("You are young!");
    }
}
```

Figure C.116 Slide 70 - Text.

Iteration, *while*

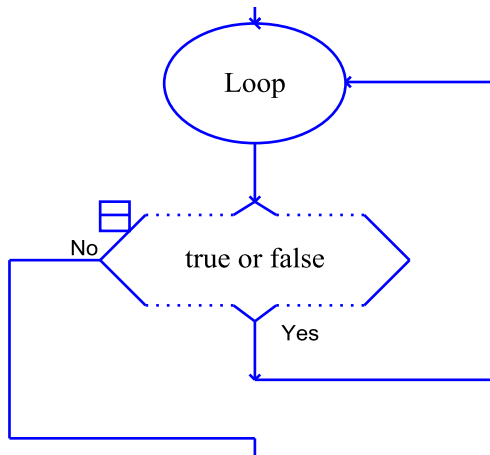


Figure C.117 Slide 61 – Flow.

Iteration, *while*

```
while(conditional)
{
}
```

Figure C.118 Slide 71 - Text.

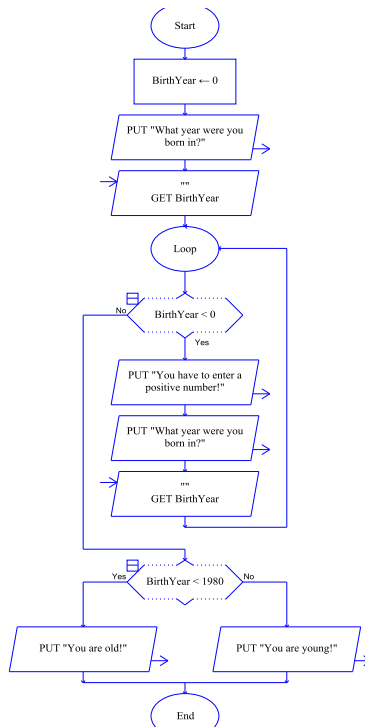


Figure C.119 Slide 62 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }

    if ( BirthYear < 1980 )
    {
        printf ( " You are old! " );
    }
    else
    {
        printf ( " You are young! " );
    }
}
  
```

Figure C.120 Slide 72 - Text.

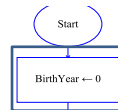


Figure C.121 Slide 63 – Flow.

```
void main(void)
{
    int BirthYear;
```

Figure C.122 Slide 73 - Text.

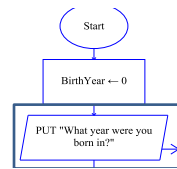


Figure C.123 Slide 64 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
}
  
```

Figure C.124 Slide 74 - Text.

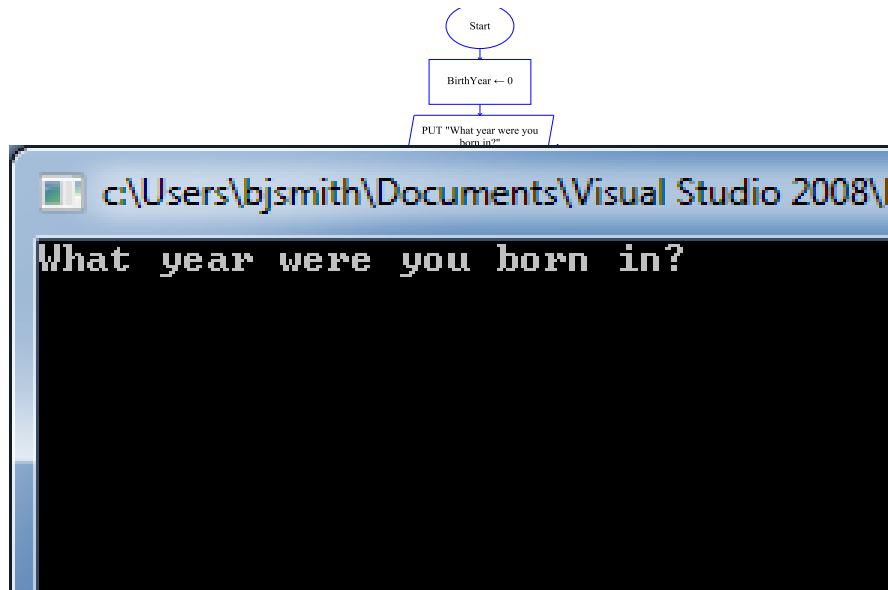


Figure C.125 Slide 65 – Flow and Slide 75 – Text.

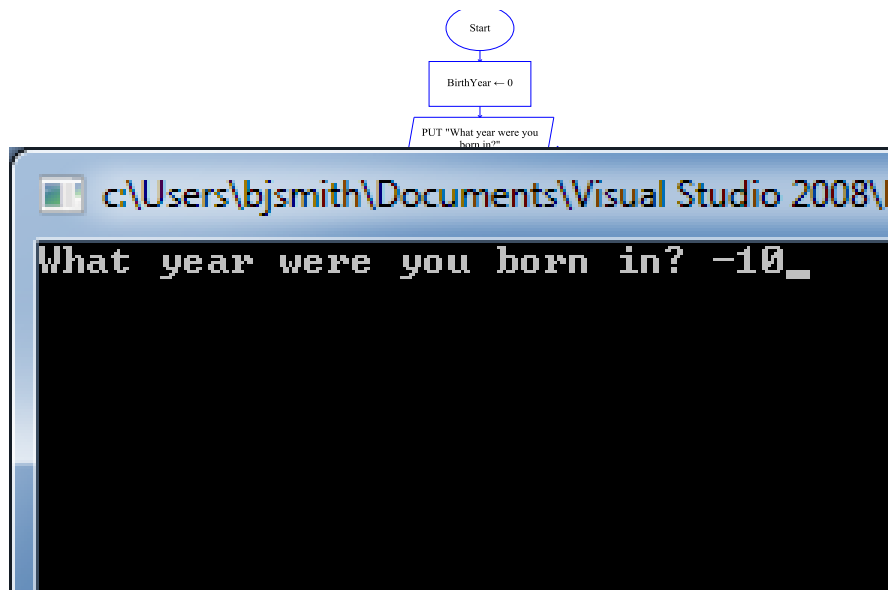


Figure C.126 Slide 66 – Flow and Slide 76 – Text.

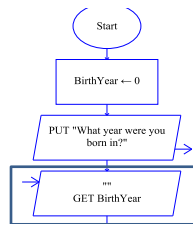


Figure C.127 Slide 67 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);
}
  
```

Figure C.128 Slide 77 - Text.

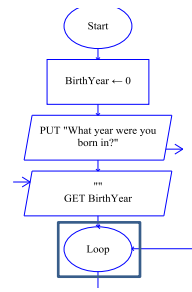


Figure C.129 Slide 68 – Flow.

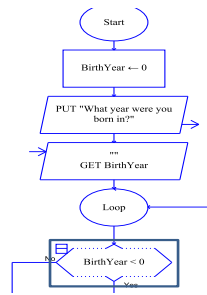


Figure C.130 Slide 69 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )

```

Figure C.131 Slide 78 - Text.

```
void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
```

Figure C.132 Slide 79 - Text.

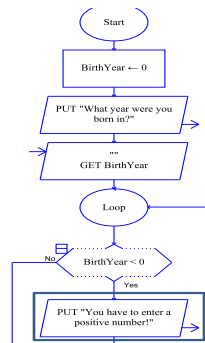


Figure C.133 Slide 70 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
    }
}

```

Figure C.134 Slide 80 - Text.

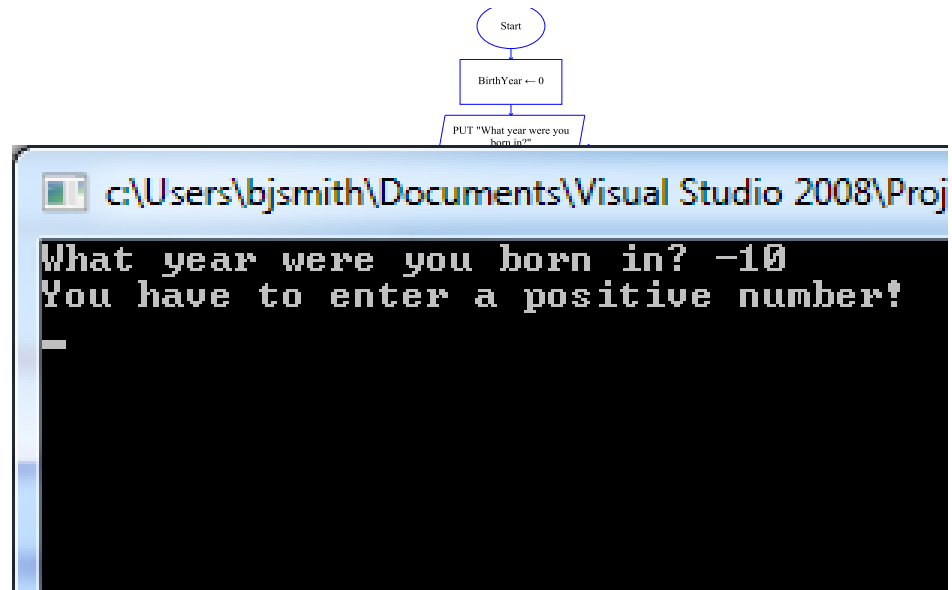


Figure C.135 Slide 71 – Flow and Slide 81 – Text.

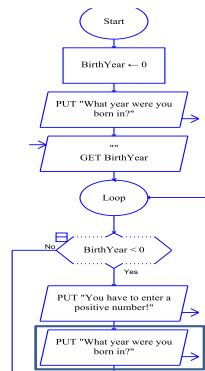


Figure C.136 Slide 72 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
    }
}
  
```

Figure C.137 Slide 82 - Text.

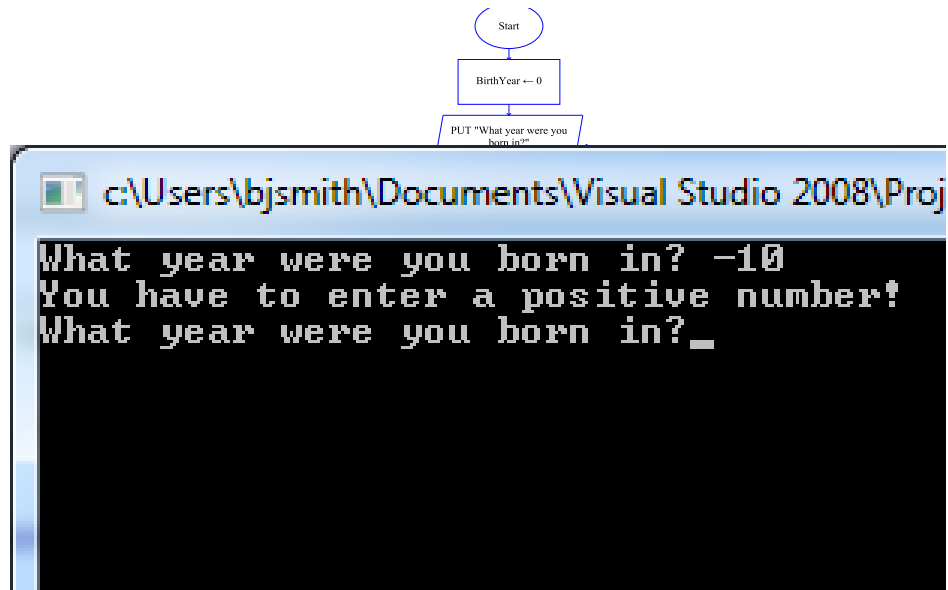


Figure C.138 Slide 73 – Flow and Slide 83 – Text.

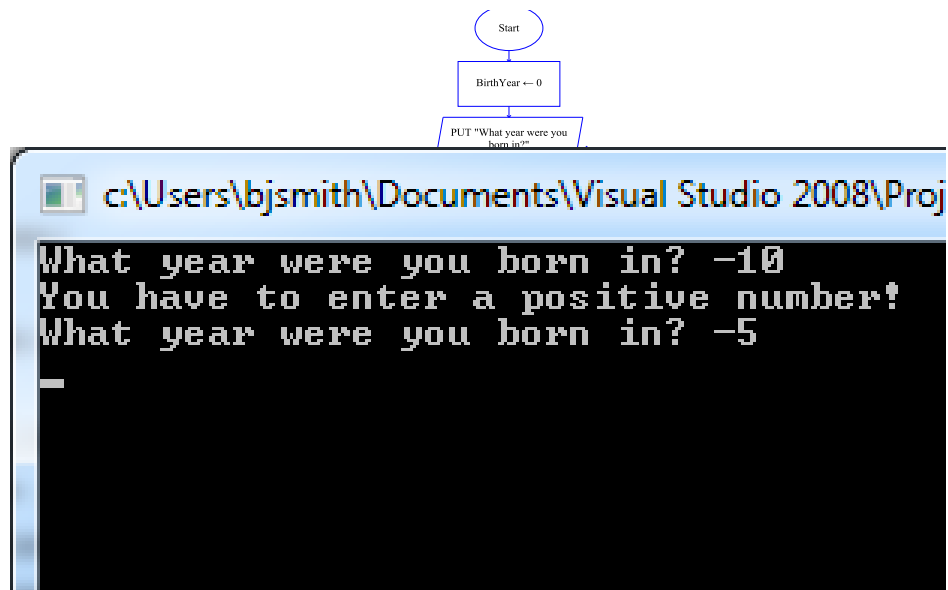


Figure C.139 Slide 74 – Flow and Slide 84 – Text.

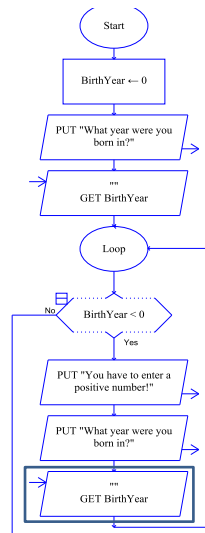


Figure C.140 Slide 75 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }
}
  
```

Figure C.141 Slide 85 - Text.

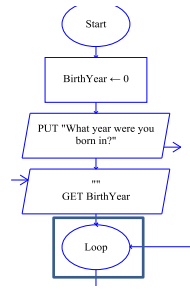


Figure C.142 Slide 76 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }
}

```

Figure C.143 Slide 86 - Text.

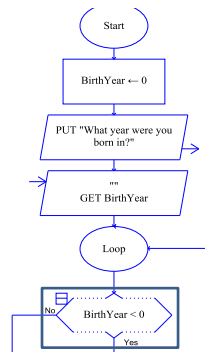


Figure C.144 Slide 77 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )

```

Figure C.145 Slide 87 - Text.

```
void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
```

Figure C.146 Slide 88 - Text.

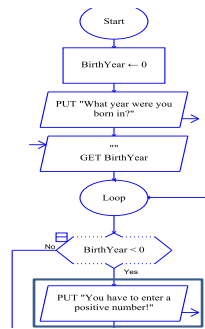


Figure C.147 Slide 78 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
    }
}

```

Figure C.148 Slide 89 - Text.

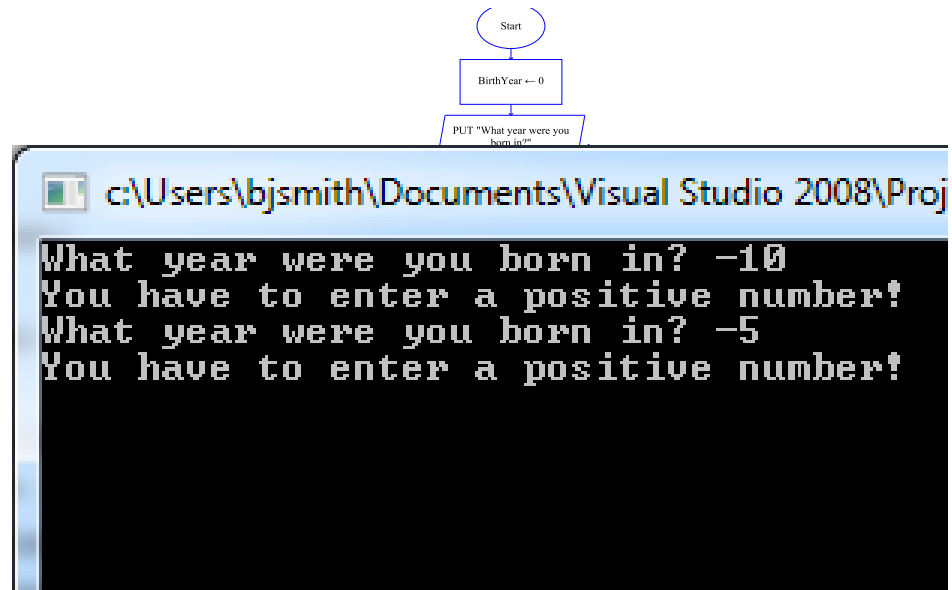


Figure C.149 Slide 79 – Flow and slide 90 – Text.

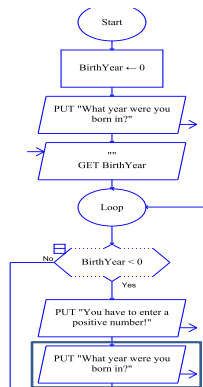


Figure C.150 Slide 80 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
    }
}
  
```

Figure C.151 Slide 91 – Text.

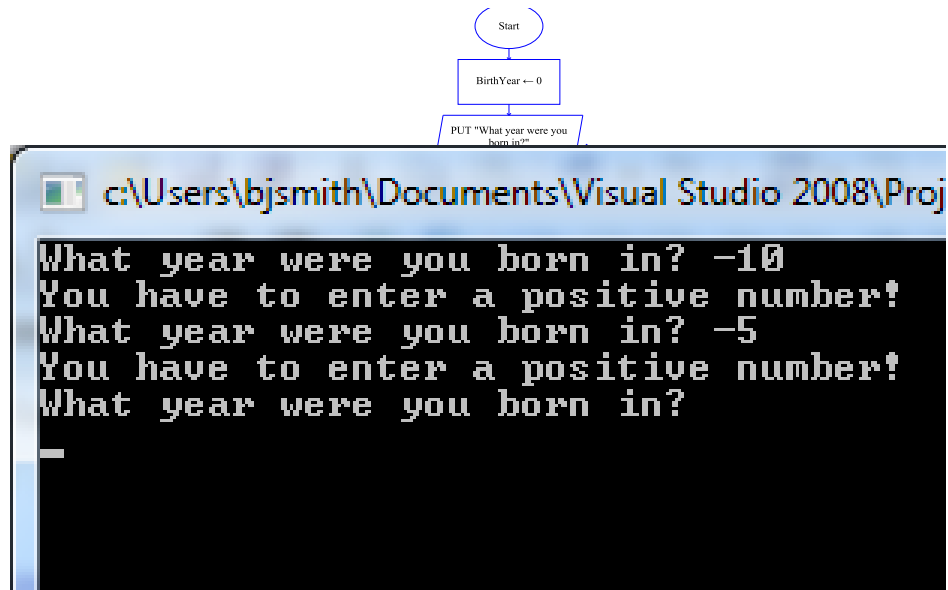


Figure C.152 Slide 81 – Flow and Slide 92 – Text.

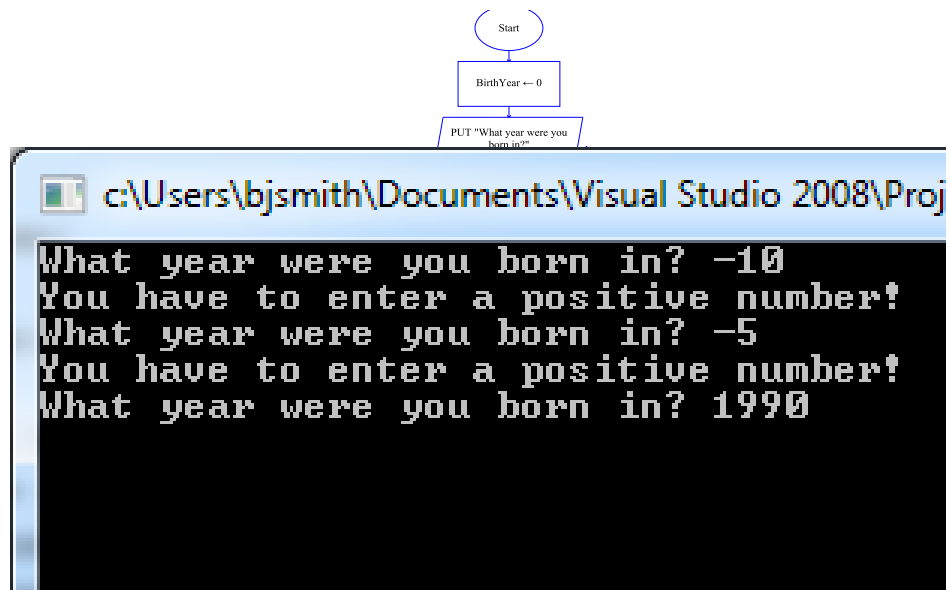


Figure C.153 Slide 82 – Flow and slide 93 – Text.

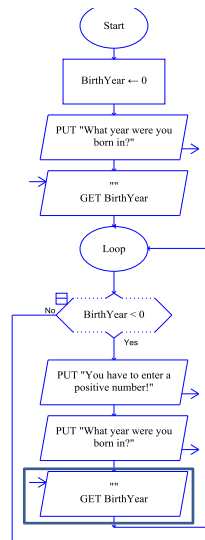


Figure C.154 Slide 83 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }
}
  
```

Figure C.155 Slide 94 - Text.

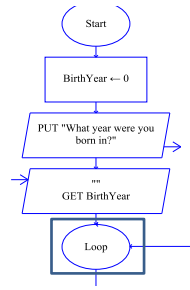


Figure C.156 Slide 84 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }
}
  
```

Figure C.157 Slide 95 - Text.

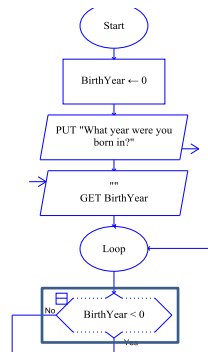


Figure C.158 Slide 85 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )

```

Figure C.159 Slide 96 - Text.

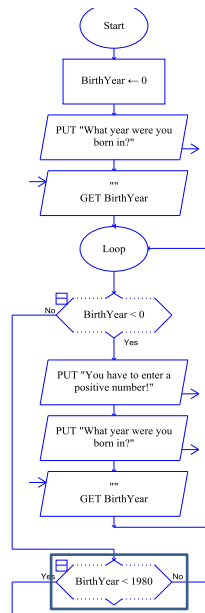


Figure C.160 Slide 86 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {

    }

    if ( BirthYear < 1980 )

```

Figure C.161 Slide 97 - Text.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {

    }

    if ( BirthYear < 1980 )
    {

    }
    else

```

Figure C.162 Slide 98 - Text.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {

    }

    if ( BirthYear < 1980 )
    {

    }
    else
    {

```

Figure C.163 Slide 99 - Text.

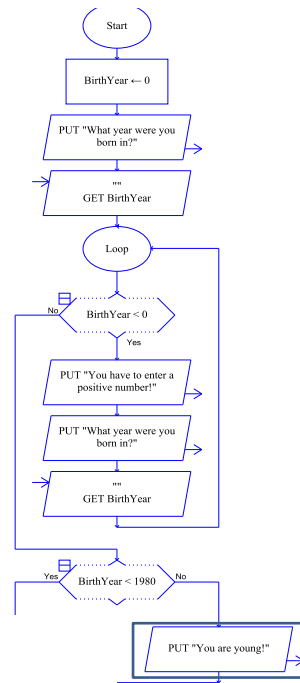


Figure C.164 Slide 87 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {

    }

    if ( BirthYear < 1980 )
    {

    }
    else
    {
        printf ( " You are young! " );
    }
}
  
```

Figure C.165 Slide 100 - Text.

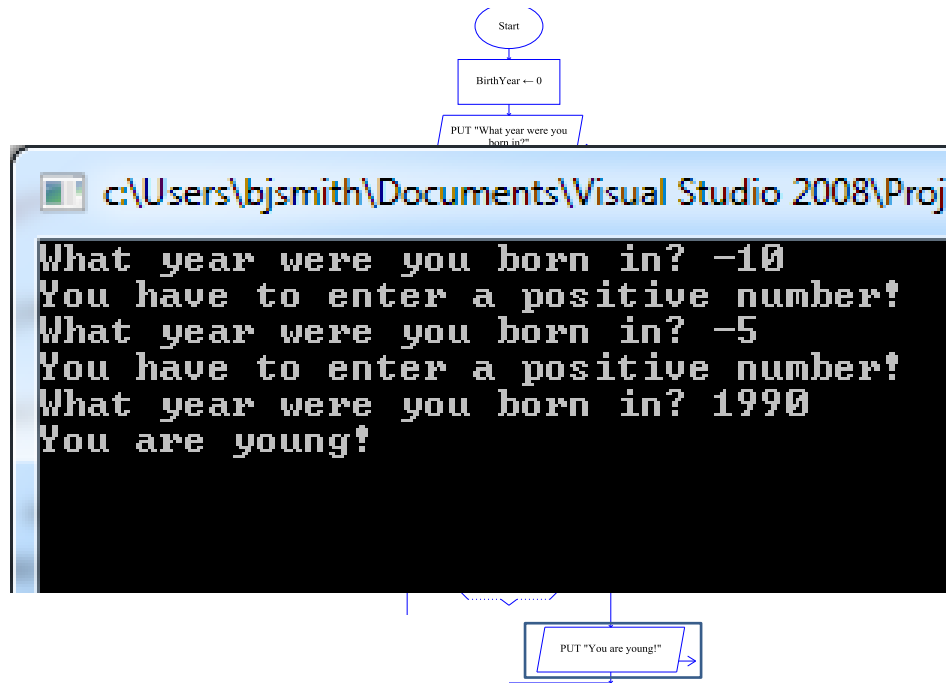


Figure C.166 Slide 88 – Flow and Slide 101 – Text.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }

    if ( BirthYear < 1980 )
    {

    }
    else
    {
        printf ( " You are young! " );
    }
}
  
```

Figure C.167 Slide 102 - Text.

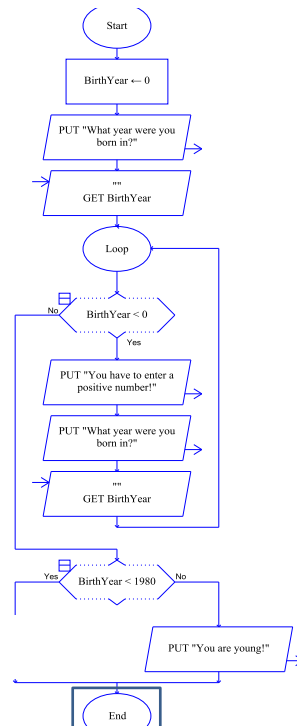


Figure C.168 Slide 89 – Flow.

```

void main(void)
{
    int BirthYear;
    printf ( " What year were you born in? " );
    scanf ( " %d " , &BirthYear);

    while ( BirthYear < 0 )
    {
        printf ( " You have to enter a positive number! " );
        printf ( " What year were you born in? " );
        scanf ( " %d " , &BirthYear);
    }

    if ( BirthYear < 1980 )
    {

    }
    else
    {
        printf ( " You are young! " );
    }
}

```

Figure C.169 Slide 103 - Text.

Appendix D

The Test, Second Pilot

The following figures represent the 10-question test that was given at the end of the second pilot. We have grouped the Flow and Text questions on the same page for easier comparison.

Test Question 1

1. If the user types in *2000* when prompted, what, if anything, will print to the screen?

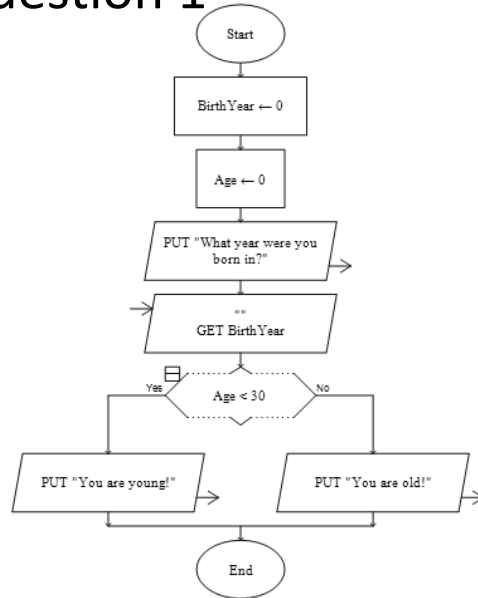


Figure D.1 Test Question 1 – Flow.

Test Question 1

If the user types in *2000* when prompted, what, if anything, will print to the screen?

```

void main(void)
{
    int BirthYear;
    int Age;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if( Age < 1980)
    {
        printf("You are old!");
    }
    else
    {
        printf("You are young!");
    }
}
  
```

Figure D.2 Test Question 1 - Text.

Test Question 2

If the user enters in “4” for height and “6” for width, what number will be displayed on the screen?

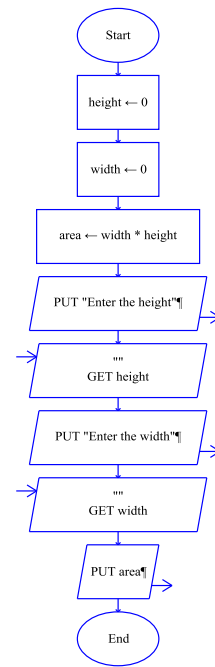


Figure D.3 Test Question 2 – Flow.

Test Question 2

If the user enters in “4” for height and “6” for width, what number will be displayed on the screen?

```
void main ( void )
{
    int height = 0;
    int width = 0;

    area = height * width;
    printf ( "Enter the height" );
    scanf ( "%d", &height );

    printf ( " Enter the width " );
    scanf ( " %d ", &width );

    print ( " %d ", area );
}
```

Figure D.4 Test Question 2 - Text.

Test Question 3

What does the following command do?

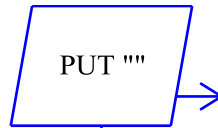


Figure D.5 Test Question 3 – Flow.

Test Question 3

What does the following command do?

```
printf ( "" ) ;
```

Figure D.6 Test Question 3 - Text.

Test Question 4

What does the following command do?

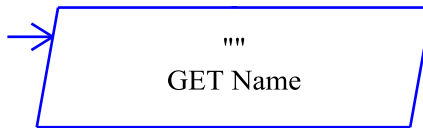


Figure D.7 Test Question 4 - Flow.

Test Question 4

What does the following command do?

```
scanf ( "%d" , &Name ) ;
```

Figure D.8 Test Question 4 - Text.

Test Question 5

How many asterisks (*) will be printed out to the screen?

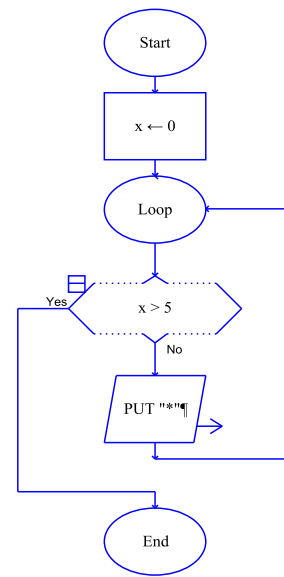


Figure D.9 Test Question 5 - Flow.

Test Question 5

How many asterisks (*) will be printed out to the screen?

```
void main(void)
{
    int x = 0;
    while ( x < 5 )
    {
        printf ( "*" );
    }
}
```

Figure D.10 Test Question 5 - Text.

Test Question 6 and 7

What is a variable?

Create a variable.

Figure D.11 Test Questions 6 and 7 - Flow.

Test Question 6 and 7

What is a variable?

Create a variable.

Figure D.12 Test Questions 6 and 7 - Text.

Test Question 8

Create your own program with an if statement.

Figure D.13 Test Question 8 - Flow.

Test Question 8

Create your own program with an if statement.

Figure D.14 Test Question 8 - Text.

Test Question 9 and 10

If the user types in 35
when prompted, what
will print to the
screen?

If the user types in -5
when prompted, what
will print to the
screen?

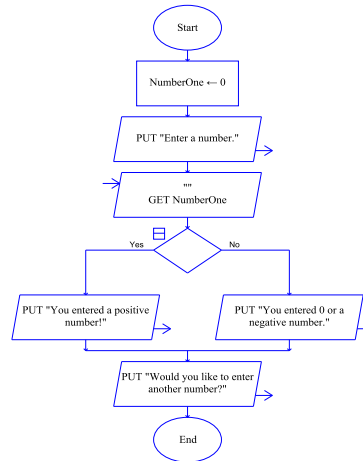


Figure D.15 Test Questions 9 and 10 - Flow.

Test Question 9 and 10

If the user types in 35 when prompted, what will print to the screen?
If the user types in -5 when prompted, what will print to the screen?

```

void main(void)
{
    int NumberOne;
    int NumberTwo;
    printf("Enter a number.");
    scanf("%d", &NumberOne);
    if( NumberOne > 0)
    {
        printf("You entered a positive number!");
    }
    else
    {
        printf("You entered 0 or a negative number.");
    }
    printf("Would you like to enter another number?");
}
  
```

Figure D.16 Test Questions 9 and 10 - Text.

Appendix E

The Test, Experiments

The following figures represent the 20-question test that was given at the end of the full experiment. We have grouped the Flow and Text questions on the same page for easier comparison.

Test Question 1

1. If the user types in *2000* when prompted, what, if anything, will print to the screen?

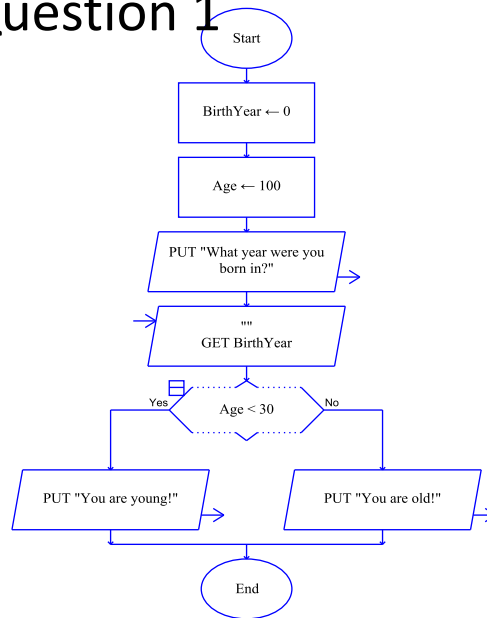


Figure E.1 Test Question 1 - Flow.

Test Question 1

1. If the user types in *2000* when prompted, what, if anything, will print to the screen?

```

void main(void)
{
    int BirthYear=0;
    int Age=100;
    printf("What year were you born in?");
    scanf("%d", &BirthYear);
    if( Age < 30)
    {
        printf("You are old!");
    }
    else
    {
        printf("You are young!");
    }
}
  
```

Figure E.2 Test Question 1 - Text.

Test Question 2

2. If the user enters in "4" for height and "6" for width, what number will be displayed on the screen (PUT area)?

- A. 4
- B. 6
- C. 24
- D. 0

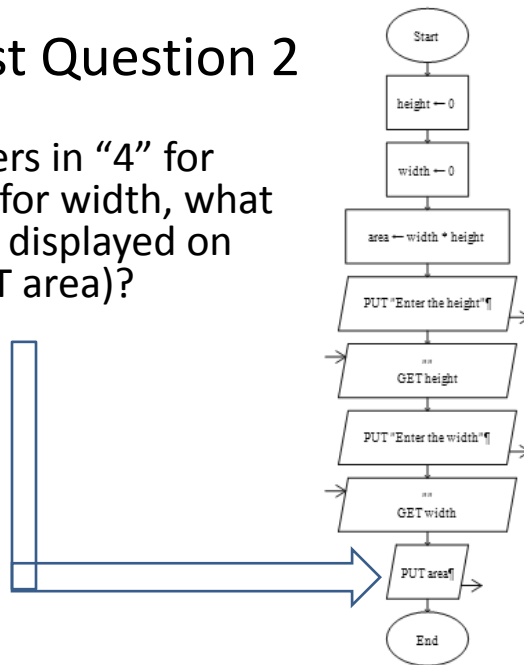


Figure E.3 Test Question 2 - Flow.

Test Question 2

2. If the user enters in "4" for height and "6" for width, what number will be displayed on the screen (printf("%d", area);)?

```

void main ( void )
{
    int height = 0;
    int width = 0;

    area = height * width;
    printf ( "Enter the height" );
    scanf ( "%d", &height );

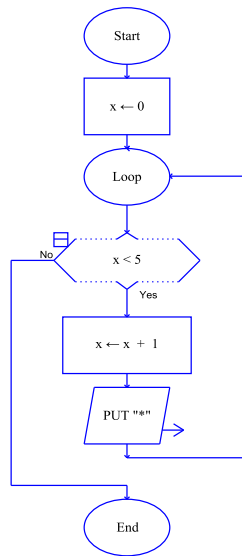
    printf ( " Enter the width " );
    scanf ( " %d ", &width );

    printf ( " %d ", area );
}
  
```

- A. 4
- B. 6
- C. 24
- D. 0

Figure E.4 Test Question 2 - Text.

Test Question 3



3. Circle the choice that represents which program will print 5 stars to the screen.

- A. Left program
- B. Right program
- C. Both programs
- D. Neither program

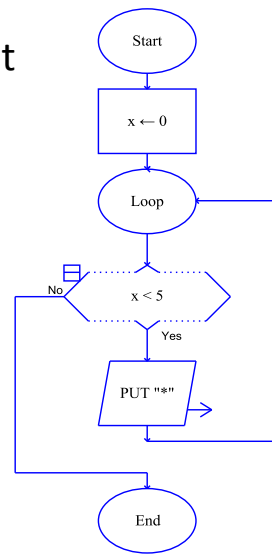


Figure E.5 Test Question 3 - Flow.

Test Question 3

```

void main (void)
{
    int x = 0;
    while(x < 5)
    {
        x = x + 1;
        printf("*");
    }
}
  
```

3. Circle the choice that represents which program will print 5 stars to the screen.

- A. Left program
- B. Right program
- C. Both programs
- D. Neither program

```

void main (void)
{
    int x = 0;
    while(x < 5)
    {
        printf("*");
    }
}
  
```

Figure E.6 Test Question 3 - Text.

Test Questions 4, 5, 6

4. How many variables are in the following program?
5. List the variables.
6. What will be printed out to the screen?

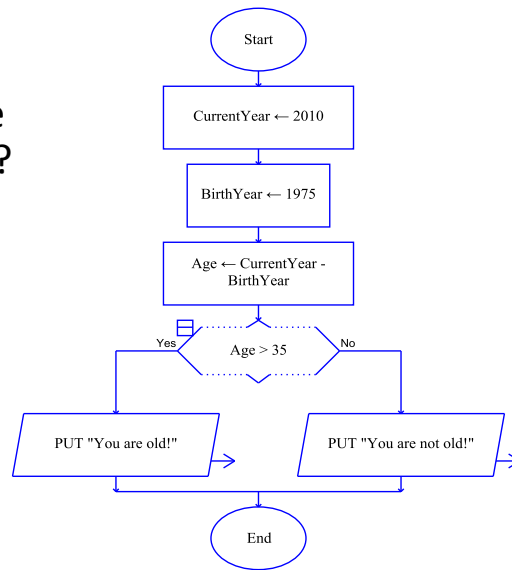


Figure E.7 Test Questions 4, 5, and 6 - Flow.

Test Questions 4, 5, 6

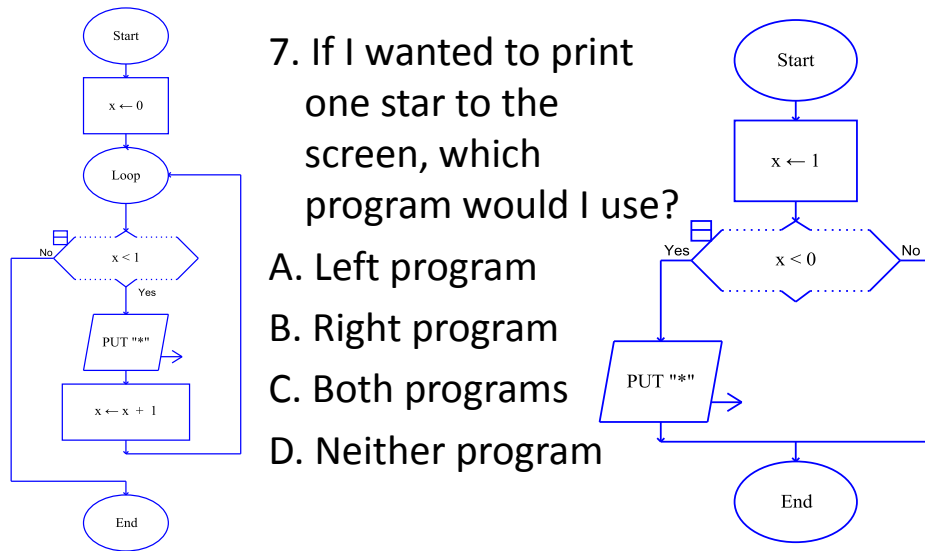
4. How many variables are in the following program?
5. List the variables.
6. What will be printed out to the screen?

```

void main (void)
{
    int CurrentYear = 2010;
    int BirthYear = 1975;
    int Age = CurrentYear - BirthYear;
    if(Age > 35)
    {
        printf("You are old!");
    }
    else
    {
        printf("You are not old!");
    }
}
  
```

Figure E.8 Test Questions 4, 5, and 6 - Text.

Test Question 7



7. If I wanted to print one star to the screen, which program would I use?

- A. Left program
- B. Right program
- C. Both programs
- D. Neither program

Figure E.9 Test Question 7 - Flow.

Test Question 7

7. If I wanted to print one star to the screen, which program would I use?

```

void main (void)
{
    int x = 0;
    while(x < 1)
    {
        printf("*");
        x = x + 1;
    }
}

```

- A. Left program
- B. Right program
- C. Both programs
- D. Neither program

```

void main (void)
{
    int x = 0;
    if(x < 0)
    {
        printf("*");
    }
}

```

Figure E.10 Test Question 7 - Text.

Test Question 8 and 9

8. If the user types in 35 when prompted, what will print to the screen?

9. If the user types in -5 when prompted, what will print to the screen?

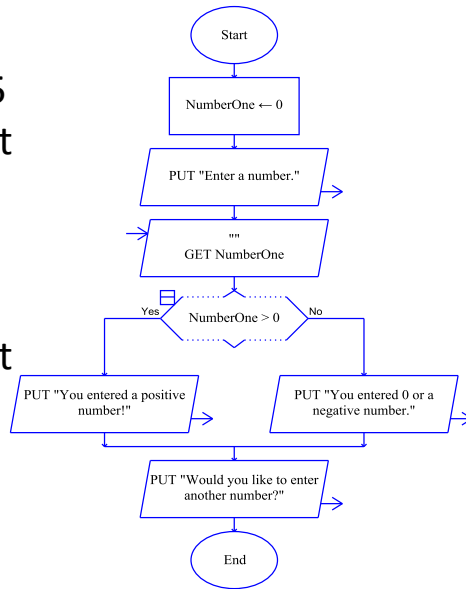


Figure E.11 Test Questions 8 and 9 - Flow.

Test Question 8 and 9

8. If the user types in 35 when prompted, what will print to the screen?

9. If the user types in -5 when prompted, what will print to the screen?

```

void main(void)
{
    int NumberOne;
    printf("Enter a number.");
    scanf("%d", &NumberOne);
    if( NumberOne > 0)
    {
        printf("You entered a positive number!");
    }
    else
    {
        printf("You entered 0 or a negative number.");
    }
    printf("Would you like to enter another number?");
}
  
```

Figure E.12 Test Questions 8 and 9 - Text.

Test Question 10

10. This program is supposed to read in a radius value from the user, calculate the diameter, then print out the diameter. Identify anything in the program that is extra and is not necessary to achieve that goal.

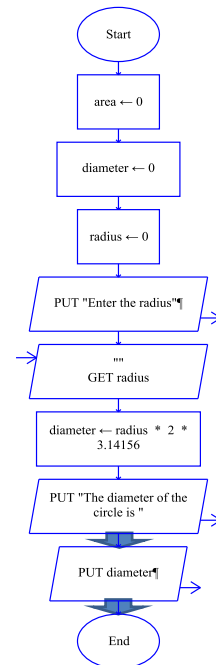


Figure E.13 Test Question 10 - Flow.

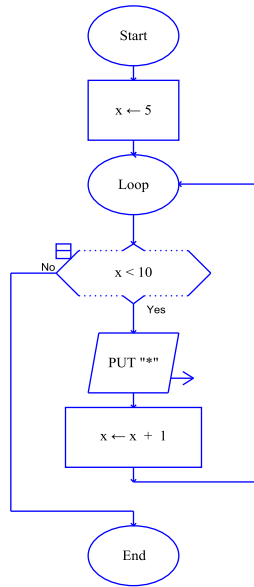
Test Question 10

10. This program is supposed to read in a radius value from the user, calculate the diameter, then print out the diameter. Identify anything in the program that is extra and is not necessary to achieve that goal.

```
void main(void)
{
    int area = 0;
    int diameter = 0;
    int radius = 0;
    printf("Enter the radius.");
    scanf("%d", &radius);
    diameter = radius * 2 * 3.14156;
    printf("The diameter of the circle is");
    printf("%d", diameter);
}
```

Figure E.14 Test Question 10 - Text.

Test Question 11



11. Circle the choice that represents which program will print 5 stars to the screen.

- A. Left program
- B. Right program
- C. Both programs
- D. Neither program

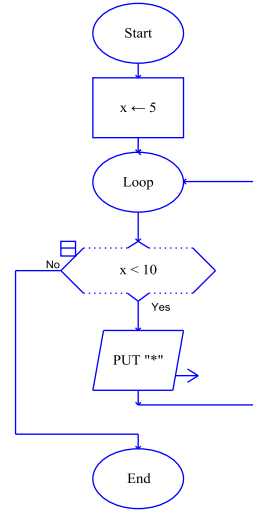


Figure E.15 Test Question 11 - Flow.

Test Question 11

11. Circle the choice that represents which program will print 5 stars to the screen.

```

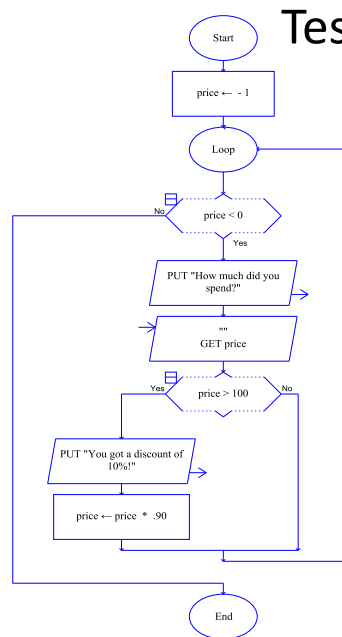
void main (void)
{
    int x = 5;
    while(x < 10)
    {
        x = x + 1;
        printf("*");
    }
}
  
```

- A. Left program
- B. Right program
- C. Both programs
- D. Neither program

```

void main (void)
{
    int x = 5;
    while(x < 10)
    {
        printf("*");
    }
}
  
```

Figure E.16 Test Question 11 - Text.



12. If the user types in 50 when prompted, what is the value of price at the end of the program?
13. If the user types in 200 when prompted, what is the value of price at the end of the program?
14. If the user types in -13 and then 200 when prompted, what is the value of price at the end of the program?

Figure E.17 Test Questions 12, 13, and 14 - Flow.

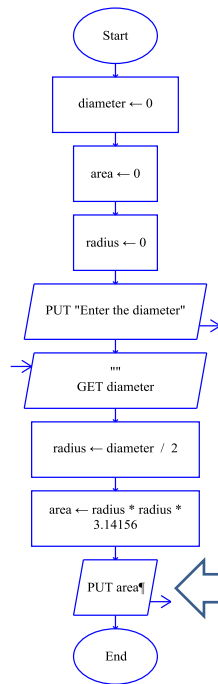
Test Questions 12, 13, 14

```

void main (void)
{
    int price = -1;
    while(price < 0)
    {
        printf("How much did you spend?");
        scanf("%d", &price);
        if(price > 100)
        {
            printf("You got a discount of 10%!");
            price = price * .90;
        }
    }
}
  
```

12. If the user types in 50 when prompted, what is the value of price at the end of the program?
13. If the user types in 200 when prompted, what is the value of price at the end of the program?
14. If the user types in -13 and then 200 when prompted, what is the value of price at the end of the program?

Figure E.18 Test Questions 12, 13, and 14 - Text.



Test Question 15

If the user enters in "10" for diameter, and diameter is equal to twice the radius, and the way to calculate area is radius times radius times pi (which is a value around 3.14156), what is the area of this circle?

- A. $10*10* 3.14156$
- B. $5*10* 3.14156$
- C. $5*5* 3.14156$
- D. Not enough information

Figure E.19 Test Question 15 - Flow.

Test Question 15

```

void main ( void )
{
    int diameter = 0;
    int area = 0;
    int radius = 0;

    printf ( "Enter the diameter" );
    scanf ( "%d", & diameter );

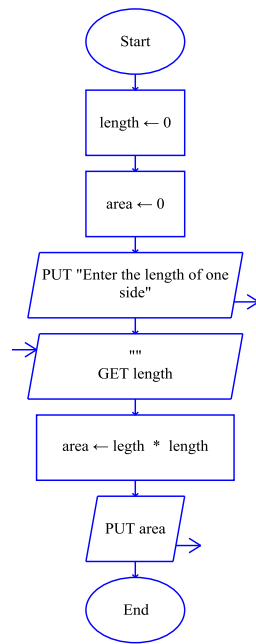
    radius = diameter / 2;
    area = radius * radius * 3.14156;

    printf ( " %d ", area );
}
  
```

If the user enters in "10" for diameter, and diameter is equal to twice the radius, and the way to calculate area is radius times radius times pi (which is a value around 3.14156), what is the area of this circle?

- A. $10*10* 3.14156$
- B. $5*10* 3.14156$
- C. $5*5* 3.14156$
- D. Not enough information

Figure E.20 Test Question 15 - Text.



Test Question 16

If the user enters in "2" for length of the side, and the area of a square is the length times the length, what is the value of area?

- A. 0
- B. 2
- C. 4
- D. Not enough information

Figure E.21 Test Question 16 - Flow.

Test Question 16

```

void main ( void )
{
    int length = 0;
    int area = 0;
  
```

```

    printf ( "Enter the length of one side" );
    scanf ( "%d" , & length );
  
```

```

    area = length * length;
  
```

```

    printf ( " %d ", area );
}
  
```

If the user enters in "2" for length of the side, and the area of a square is the length times the length, what is the value of area?

- A. 0
- B. 2
- C. 4
- D. Not enough information

Figure E.22 Test Question 16 - Text.

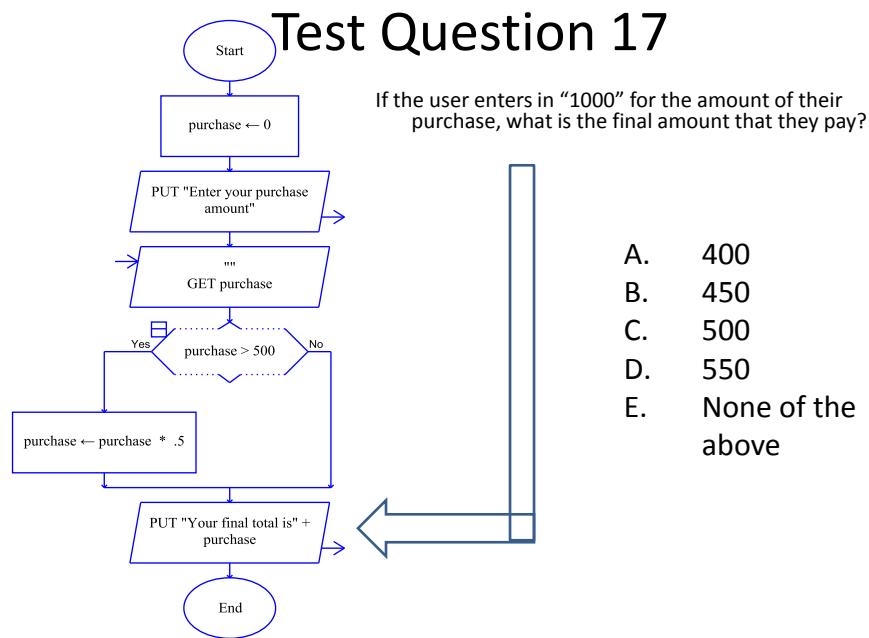


Figure E.23 Test Question 17 - Flow.

Test Question 17

If the user enters in "1000" for the amount of their purchase, what is the final amount that they pay?

```

void main ( void )
{
    int purchase= 0;
    printf("Enter your purchase amount");
    scanf("%d", &purchase);
    if(purchase > 500)
    {
        purchase = purchase * .5;
    }
    printf("Your final total is %d", purchase);
}
  
```

A. 400
B. 450
C. 500
D. 550
E. None of the above

Figure E.24 Test Question 17 - Text.

Test Question 18

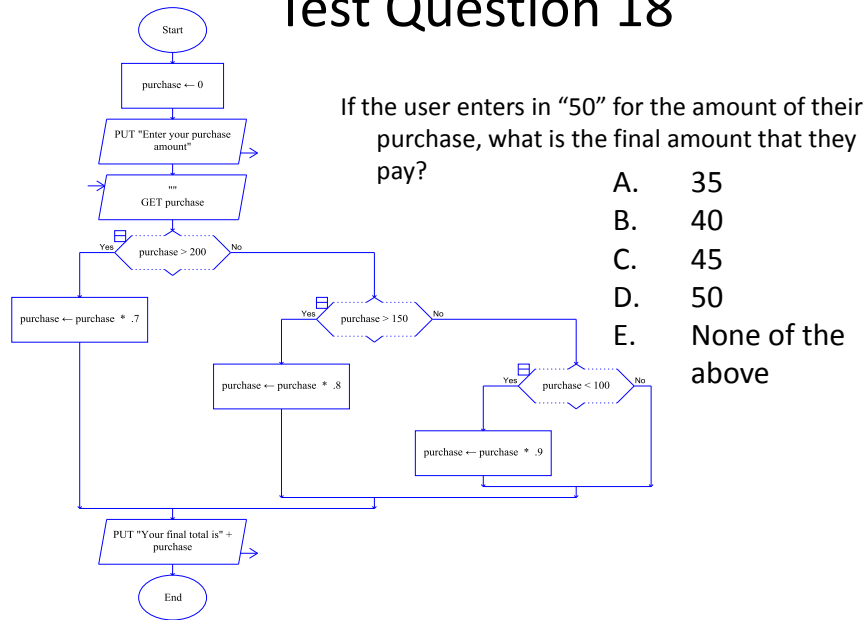


Figure E.25 Test Question 18 - Flow.

Test Question 18

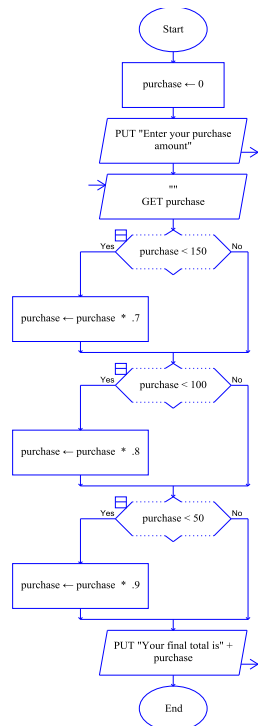
```

void main ( void )
{
    int purchase= 0;
    printf("Enter your purchase amount");
    scanf("%d", &purchase);
    if(purchase > 200)
    {
        purchase = purchase * .7;
    }
    else if (purchase > 150)
    {
        purchase = purchase * .8;
    }
    else if (purchase < 100)
    {
        purchase = purchase * .9;
    }
    printf("Your final total is %d", purchase);
}
  
```

If the user enters in "50" for the amount of their purchase, what is the final amount that they pay?

- A. 35
B. 40
C. 45
D. 50
E. None of the above

Figure E.26 Test Question 18 - Text.



Test Question 19

If the user enters in “200” for the amount of their purchase, what is the final amount that they pay?

- A. 140
- B. 180
- C. 160
- D. 200
- E. None of the above

Figure E.27 Test Question 19 - Flow.

Test Question 19

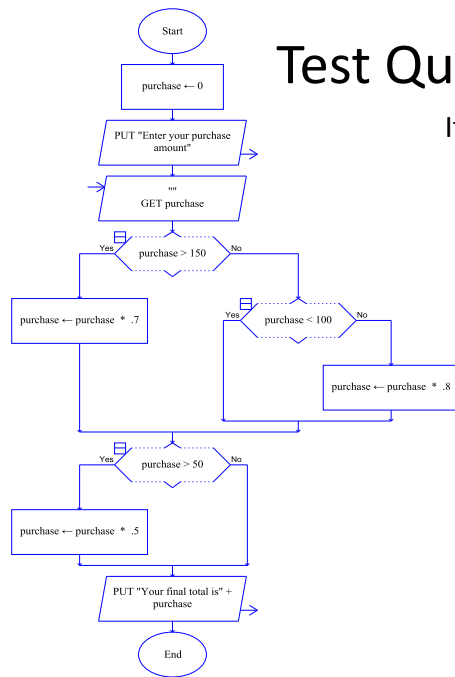
If the user enters in “200” for the amount of their purchase, what is the final amount that they pay?

```

void main ( void )
{
    int purchase= 0;
    printf("Enter your purchase amount");
    scanf("%d", &purchase);
    if(purchase < 150)
    {
        purchase = purchase * .7;
    }
    if (purchase < 100)
    {
        purchase = purchase * .8;
    }
    if (purchase < 50)
    {
        purchase = purchase * .9;
    }
    printf("Your final total is %d", purchase);
}
  
```

- A. 140
- B. 180
- C. 160
- D. 200
- E. None of the above

Figure E.28 Test Question 19 - Text.



Test Question 20

If the user enters in “200” for the amount of their purchase, what is the final amount that they pay?

- A. 200
- B. 140
- C. 160
- D. 100
- E. 80
- F. 70
- G. 50
- H. None of the above

Figure E.29 Test Question 20 - Flow.

Test Question 20

```

void main ( void )
{
    int purchase= 0;
    printf("Enter your purchase amount");
    scanf("%d", &purchase);
    if(purchase > 150)
    {
        purchase = purchase * .7;
    }
    else if (purchase < 100)
    {
        purchase = purchase * .8;
    }
    if (purchase > 50)
    {
        purchase = purchase * .5;
    }
    printf("Your final total is %d", purchase);
}
  
```

If the user enters in “200” for the amount of their purchase, what is the final amount that they pay?

- A. 200
- B. 140
- C. 160
- D. 100
- E. 80
- F. 70
- G. 50
- H. None of the above

Figure E.30 Test Question 20 - Text.

REFERENCES

- [1] Denning, P. and A. McGettrick, *Recentering computer science*. Communications of the ACM, 2005. **48**(11): p. 19.
- [2] Ma, L., et al., *Investigating the viability of mental models held by novice programmers*. ACM SIGCSE Bulletin, 2007. **39**(1): p. 503.
- [3] Bransford, J.D., Brown, A.L., and Cocking, R.R., eds., *How people learn: Brain, mind, experience, and school*. National Research Council, National Academies Press, Washington, DC, 2000.
- [4] Biggs, J., *Teaching for quality learning at university (Buckingham, Society for Research into Higher Education and Open University Press)*. The Higher Education Academy.(2008). Groupwork, Retrieved August 2003. **6**: p. 2008.
- [5] McKeachie, W. and B. Hofer, *McKeachie's Teaching Tips: Strategies, Research, and Theory for College and University Teachers*. 2002: Houghton Mifflin, Boston.
- [6] Ramsden, P., *Learning to Teach in Higher Education, 2nd ed*. London: Taylor and Francis, Inc., 2003.
- [7] Felder, R. and R. Brent, *Understanding student differences*. Journal of Engineering Education, 2005. **94**(1): p. 57-72.
- [8] Kittler, F. and P. Similon, *Gramophone, film, typewriter*. October 1987. **41**: p. 101-118.
- [9] Crapo, A., et al., *Visualization and the process of modeling: A cognitive-theoretic view*. 2000: ACM New York, NY, USA.
- [10] Wade, N. and M. Swanston, *Visual perception: An introduction*. 2001: Psychology Pr.
- [11] Kosslyn, S., K. Sukel, and B. Bly, *Squinting with the mind's eye: Effects of stimulus resolution on imaginal and perceptual comparisons*. Memory & cognition, 1999. **27**(2): p. 276.
- [12] Pinker, S., *How The Mind Works*. 1999: WW Norton & Company.
- [13] Healey, C., K. Booth, and J. Enns, *High-speed visual estimation using preattentive processing*. ACM Transactions on Computer-Human Interaction (TOCHI), 1996. **3**(2): p. 107-135.
- [14] Card, S., J. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. 1999: Morgan Kaufmann.
- [15] Felder, R., *Index of Learning Styles Questionnaire*. [Web Page] [cited 2009; <http://www.engr.ncsu.edu/learningstyles/ilsweb.html>]. Available from: <http://www.engr.ncsu.edu/learningstyles/ilsweb.html>.

- [16] Howard, R.A., C.A. Carver, and W.D. Lane, *Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: Tying it all together in the CS2 course*, in *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*. 1996, ACM: Philadelphia, PA, USA. p. 227-231.
- [17] Hoc, J. and A. Nguyen-Xuan, *Language semantics, mental models and analogy*. Psychology of Programming, 1990: p. 139-156.
- [18] Bransford, J., *Human Cognition*. 1979: Wadsworth Belmont, CA.
- [19] Mayer, R.E., *The psychology of how novices learn computer programming*. ACM Comput. Surv., 1981. **13**(1): p. 121-141.
- [20] Winslow, L., *Programming pedagogy-A psychological overview*. ACM SIGCSE Bulletin, 1996. **28**(3): p. 17-22.
- [21] Dreyfus, H., T. Anthanasiou, and S. Dreyfus, *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. 2000: Simon & Schuster.
- [22] Ragonis, N. and M. Ben-Ari, *On understanding the statics and dynamics of object-oriented programs*. ACM SIGCSE Bulletin, 2005. **37**(1): p. 226-230.
- [23] Lahtinen, E., K. Ala-Mutka, and H. Järvinen, *A study of the difficulties of novice programmers*. 2005: ACM.
- [24] Schulte, C. and J. Bennedsen, *What do teachers teach in introductory programming?* 2006: ACM.
- [25] Pashler, H., et al., *Learning styles: Concepts and evidence*. Psychological Science in the Public Interest, 2008. **9**(3): p. 105-119.
- [26] Coffield, F., et al., *Learning styles and pedagogy in post-16 learning: A systematic and critical review*. 2009.
- [27] Felder, R., *Matters of style*. Asee Prism, 1996. **6**(4): p. 18-23.
- [28] Foundation, T.M.s.a.B., *The 16 MBTI Types*. Available from: <http://www.myersbriggs.org/my-mbti-personality-type/mbti-basics/the-16-mbti-types.asp#INFP>.
- [29] Jung, C., *Psychological Types*. 1971: Routledge, London.
- [30] Briggs-Myers, I., *Myers-Briggs Type Indicator*. 1957: Educational Testing Service, Princeton, NJ.
- [31] Kolb, D., *Learning styles and disciplinary differences*. The modern American college, 1981: p. 232-255.
- [32] Felder, R. and L. Silverman, *Learning and teaching styles in engineering education*. Engineering education, 1988. **78**(7): p. 674-681.
- [33] Harb, J. and R. Terry, *A look at performance tools through the use of the Kolb learning cycle*. 1992.
- [34] Gazzaniga, M., *Forty-five years of split-brain research and still going strong*. Nature Reviews Neuroscience, 2005. **6**(8): p. 653-659.

- [35] Sperry, R., *Hemisphere deconnection and unity in conscious awareness*. American Psychologist, 1968. **23**(10): p. 723-733.
- [36] Taylor, I. and M. Taylor, *Psycholinguistics: Learning and Using Language*. 1990: Prentice Hall.
- [37] Olds, B., B. Moskal, and R. Miller, *Assessment in engineering education: Evolution, approaches and future collaborations*. Journal of Engineering Education, 2005. **94**(1): p. 13-25.
- [38] Carlisle, M.C., et al., *RAPTOR: Introducing programming to non-majors with flowcharts*. J. Comput. Small Coll., 2004. **19**(4): p. 52-60.
- [39] Zywno, M., *A contribution to validation of score meaning for Felder-Soloman's index of learning styles*. 2003.
- [40] Seery, N., W. Gaughran, and T. Waldmann, *Multi-modal learning in engineering education*.
- [41] Livesay, G., et al., *Engineering student learning styles: A statistical analysis using Felder's Index of Learning Styles*. 2002.
- [42] Van Zwanenberg, N., L. Wilkinson, and A. Anderson, *Felder and Silverman's Index of Learning Styles and Honey and Mumford's Learning Styles Questionnaire: How do they compare and do they predict academic performance?* Educational Psychology, 2000. **20**(3): p. 365-380.
- [43] Tuckman, B., *Conducting Educational Research*. 5 (ed.)1999. Harcourt Brace College Publishers, Ft. Worth, TX.
- [44] Litzinger, T., S. Lee, and J. Wise, *A study of the reliability and validity of the Felder-Soloman Index of Learning Styles*©. Education, 2005. **113**: p. 77.
- [45] Felder, R. and J. Spurlin, *Applications, reliability and validity of the Index of Learning Styles*. International Journal of Engineering Education, 2005. **21**(1): p. 103-112.
- [46] Felder, R., *Reaching the second tier: Learning and teaching styles in college science education*. J. Coll. Sci. Teaching, 1993. **23**(5): p. 4.
- [47] Chamillard, A.T. and D. Karolick, *Using learning style data in an introductory computer science course*. SIGCSE Bull., 1999. **31**(1): p. 291-295.
- [48] Naps, T.L., et al., *Exploring the role of visualization and engagement in computer science education*, in *Working group reports from ITiCSE on Innovation and technology in computer science education*. 2002, ACM: Aarhus, Denmark. p. 131-152.
- [49] Chang, B., D. Ungar, and R. Smith, *Getting close to objects*. 1995.
- [50] Lord, H., *Visual programming for visual applications*. Object Magazine, Julho-Agosto, 1994.
- [51] Pinker, S., *Visual cognition: An introduction* I*. Cognition, 1984. **18**(1-3): p. 1-63.
- [52] Burnett, M.B., C. Bohus, P. Carlson, S. Yang & P. van Zee, *"Scaling up visual programming languages"*. IEEE Computer, March, 1995.
- [53] Bragdon, A., et al., *Code bubbles: A working set-based interface for code understanding and maintenance*. 2010: ACM.
- [54] Boshernitsan, M. and M. Downes, *Visual programming languages: A survey*. Computer, 2004.

- [55] Griswold, W., J. Yuan, and Y. Kato, *Exploiting the map metaphor in a tool for software evolution*. 2001: IEEE Computer Society.
- [56] Bransford, J. and M. Johnson, *Contextual prerequisites for understanding: Some investigations of comprehension and recall*. Cognitive Psychology: Key Readings, 2004: p. 431.
- [57] Dooling, D. and R. Lachman, *Effects of comprehension on retention of prose*. Journal of Experimental Psychology, 1971. **88**(2): p. 216-222.
- [58] Dooling, D. and R. Mullet, *Locus of thematic effects in retention of prose*. Journal of Experimental Psychology, 1973. **97**(3): p. 404-406.
- [59] Finke, R., *Creative Imagery: Discoveries and Inventions in Visualization*. 1990: Lawrence Erlbaum.
- [60] Reisberg, D. and R. Logie, *The ins and outs of working memory: Overcoming the limits on learning from imagery*. Advances in psychology, 1993. **98**: p. 39-76.
- [61] Whitley, K., *Visual programming languages and the empirical evidence for and against*. Journal of Visual Languages and Computing, 1997. **8**(1): p. 109-142.
- [62] von Mayrhauser, A. and A. Vans, *Program understanding-A survey*. Colorado State University Computer Science Technical Report CS94-120, 1994.
- [63] Guindon, R., *Knowledge exploited by experts during software system design*. International Journal of Man-Machine Studies, 1990. **33**(3): p. 304.
- [64] Robins, A., J. Rountree, and N. Rountree, *Learning and teaching programming: A review and discussion*. Computer Science Education, 2003. **13**(2): p. 137-172.
- [65] Zualkernan, I., *Using Soloman-Felder learning style index to evaluate pedagogical resources for introductory programming classes*. 2007: IEEE Computer Society.
- [66] Lister, R., *On blooming first year programming, and its blooming assessment*. 2000: ACM.
- [67] C. Hundhausen, S.A.D., and J. T. Stasko, *A meta-study of algorithm visualization effectiveness*. Journal of Visual Languages and Computing, 2002.
- [68] Lave, J. and E. Wender, *Situated Learning: Legitimate Peripheral Participation*. 1991: Cambridge University Press, New York.
- [69] Powers, K., et al., *Tools for teaching introductory programming: What works?* 2006: ACM.
- [70] Sowa, J.F., *Knowledge representation: Logical, philosophical and computational foundations*. 2000: Brooks/Cole Publishing Co., Pacific Grove, CA.
- [71] Mineau, G., *From actors to processes: The representation of dynamic knowledge using conceptual graphs*. Lecture notes in computer science, 1998: p. 65-79.
- [72] Cyre, W., *Executing Conceptual Graphs*. 1998: Springer-Verlag.
- [73] Lukose, D., *Executable Conceptual Structures*. 1993: Springer-Verlag.

- [74] Smith, B.J. and H.S. Delugach, *Another reason why conceptual graphs need actors*. %U <http://dblp.uni-trier.de/db/conf/iccs/iccs2009.html#SmithD09>, in *ICCS* %@ 978-3-642-03078-9, R. Sebastian, D. Frithjof, and O.K. Sergej, Editors. 2009, Springer. p. 293-306.
- [75] Carlisle, M., *RAPTOR - Flowchart interpreter*. 2010 [cited 2010; Available from: <http://raptor.martincarlisle.com/>].
- [76] Conway, M., et al., *Alice: Lessons learned from building a 3D system for novices*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2000, ACM: The Hague, The Netherlands. p. 486-493.
- [77] Rongas, T., et al., *ALOHA: Visual learning tool for comprehending program structure in introductory programming*.
- [78] Carlisle, M.C., *Raptor: A visual programming environment for teaching object-oriented programming*. J. Comput. Small Coll., 2009. **24**(4): p. 275-281.
- [79] Chen, S. and S. Morris, *Iconic programming for flowcharts, java, turing, etc*. 2005: ACM.
- [80] Olivieri, L., *Using Visual Logic© to teach programming logic in an introductory CS course*. Journal of Computing Sciences in Colleges, 2009. **24**(6): p. 146-148.
- [81] Carlisle, M., *RAPTOR Questions*, B. Smith, Editor. 2009.
- [82] DuHadway, L.P., et al., *A concept-first approach for an introductory computer science course*. J. Comput. Small Coll., 2002. **18**(2): p. 6-16.
- [83] Gray, K. and M. Flatt, *ProfessorJ: A gradual introduction to Java through language levels*. 2003: ACM.
- [84] Erwin, B., M. Cyr, and C. Rogers, *Lego engineer and robolab: Teaching engineering with labview from kindergarten to graduate school*. International Journal of Engineering Education, 2000. **16**(3): p. 181-192.
- [85] Storey, M.-A., *Theories, methods and tools in program comprehension: Past, present and future*, in *Proceedings of the 13th International Workshop on Program Comprehension*. 2005, IEEE Computer Society. p. 181-191.
- [86] Mead, J., et al., *A cognitive approach to identifying measurable milestones for programming skill acquisition*. SIGCSE Bull., 2006. **38**(4): p. 182-194.
- [87] Hancock, C.M., *Real-time programming and the big ideas of computational literacy*. 2003, Massachusetts Institute of Technology. p. 1.
- [88] Driscoll, M.P., *Psychology of Learning for instruction*: Allyn & Bacon.
- [89] Allert, J.D., *The effectiveness of innovative approaches to CSI: Comparing opinion to outcome*, in *Proceedings of the 27th Australasian Conference on Computer Science - Volume 26*. 2004, Australian Computer Society, Inc.: Dunedin, New Zealand. p. 151-157.
- [90] Allert, J., *Learning style and factors contributing to success in an introductory computer science course*, in *Proceedings of the IEEE International Conference on Advanced Learning Technologies*. 2004, IEEE Computer Society. p. 385-389.

- [91] Moskal, B., D. Lurie, and S. Cooper, *Evaluating the effectiveness of a new instructional approach*. 2004: ACM.
- [92] Cooper, S., W. Dann, and R. Pausch, *Teaching objects-first in introductory computer science*. 2003: ACM.
- [93] Dann, W., et al., *Objects: Visualization of behavior and state*. 2003: ACM.
- [94] Giordano, J.C. and M. Carlisle, *Toward a more effective visualization tool to teach novice programmers*, in *Proceedings of the 7th Conference on Information Technology Education*. 2006, ACM: Minneapolis, Minnesota, USA. p. 115-122.
- [95] Carlisle, M., et al., *RAPTOR: A visual programming environment for teaching algorithmic problem solving*. ACM SIGCSE Bulletin, 2005. **37**(1): p. 176-180.
- [96] Du Boulay, B., T. O'Shea, and J. Monk, *The black box inside the glass box: Presenting computing concepts to novices*. International Journal of Human-Computer Studies, 1999. **51**(2): p. 265-277.
- [97] Storey, M., F. Fracchia, and H. Müller, *Cognitive design elements to support the construction of a mental model during software exploration*. The Journal of Systems & Software, 1999. **44**(3): p. 171-185.
- [98] Blackwell, A.F., *Metacognitive Theories of Visual Programming: What do we think we are doing?*, in *Proceedings of the 1996 IEEE Symposium on Visual Languages*. 1996, IEEE Computer Society. p. 240.
- [99] Cordy, J., et al., *Source transformation in software engineering using the TXL transformation system*. Information and Software Technology, 2002. **44**(13): p. 827-837.
- [100] Glynn, E., I. Hayes, and A. MacDonald, *Integration of generic program analysis tools into a software development environment*, in *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38*. 2005, Australian Computer Society, Inc.: Newcastle, Australia. p. 249-257.
- [101] Delugach, H.S., *CharGer*. [Software] 2009.
- [102] Cooper, G., *Research into cognitive load theory and instructional design at UNSW*. December 1998.
- [103] Johnson-Laird, P., *Human and Machine Thinking*. 1993: Lawrence Erlbaum.
- [104] Alabastro, M., et al., *The use of visual modeling in designing a manufacturing process for advanced composite structures*. IEEE Transactions on Engineering Management, 1995. **42**(3): p. 233-242.
- [105] Fred Paas, A.R. and John Sweller, *Cognitive load theory and instructional design: Recent developments [introduction to special issue]*. Education Psychologist, 2003. **38**(1).
- [106] Jones, C., *Visualization and Optimization*. 1996: Springer.
- [107] Tegarden, D., *Business information visualization*. Communications of the AIS, 1999. **1**(1es).
- [108] Vessey, I., *Cognitive fit: A theory-based analysis of the graphs versus tables literature*. Decision Sciences, 1991. **22**(2): p. 219-240.

- [109] Bell, J. and H. Fogler. *The Investigation and Application of Virtual Reality as an Educational Tool*. 1995: Citeseer.
- [110] Scott, T., *Bloom's taxonomy applied to testing in computer science classes*. Journal of Computing Sciences in Colleges, 2003. **19**(1): p. 267-274.
- [111] Cook, D., et al., *Lack of interaction between sensing–intuitive learning styles and problem-first versus information-first instruction: A randomized crossover trial*. Advances in Health Sciences Education, 2009. **14**(1): p. 79-90.
- [112] Silverman, L. and D. Publishing, *Upside-down Brilliance*: DeLeon Publishing.
- [113] Mellor, S. and M. Balcer, *Executable UML: A Foundation for Model-driven Architectures*. 2002: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
- [114] Bennett, K. and V. Rajlich, *Software maintenance and evolution: A roadmap*. 2000: ACM.