Coding Strip: A Tool for Supporting Interplay within Abstraction Ladder for Computational Thinking

by

Sangho Suh

A thesis presented to the University of Waterloo in fulfillment of the thesis requirement for the degree of Doctor of Philosophy in Computer Science

© Sangho Suh 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Amy Ko Professor, Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle
Supervisor(s):	Edith Law Associate Professor, David R. Cheriton School of Computer Science, University of Waterloo
Internal Member:	Edward Lank Professor, David R. Cheriton School of Computer Science, University of Waterloo Jian Zhao Assistant Professor, David R. Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Derek Rayside

Associate Professor, Dept. of Electrical and Computer Engineering, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This dissertation includes first-authored peer-reviewed materials that appeared in conference proceedings published by the Association for Computing Machinery (ACM) and IEEE. The ACM's policy on reuse of published materials in a dissertation is as follows 1

"Authors can include partial or complete papers of their own (and no fee is expected) in a dissertaion as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included."

The IEEE's policy on reuse of published materials in a dissertation is as follows 2

"The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant."

Chapter 2

Sangho Suh, Martinet Lee, and Edith Law. How do we design for concreteness fading? survey, general framework, and design dimensions. In Proceedings of the Interaction Design and Children Conference, pp. 581-588. 2020. DOI: 10.1145/3392063.3394413

Chapter 3

Sangho Suh. Using concreteness fading to model & design learning process. In Proceedings of the 2019 ACM Conference on International Computing Education Research, pp. 353-354. 2019. DOI: 10.1145/3291279.3339445

Chapter 4

Sangho Suh, Martinet Lee, Gracie Xia, and Edith Law. Coding strip: A pedagogical tool for teaching and learning programming concepts through comics. In 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 1-10. IEEE, 2020. DOI: 10.1109/VL/HCC50065.2020.9127262

Chapter 6

Sangho Suh, Celine Latulipe, Ken Jen Lee, Bernadette Cheng, and Edith Law. Using comics to introduce and reinforce programming concepts in CS1. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, pp. 369-375. 2021. DOI: 10.1145/3408877.3432465

¹https://authors.acm.org/author-services/author-rights. Accessed January, 2022.

²https://www.ieee.org/content/dam/ieee-org/ieee/web/org/pubs/permissions_faq.pdf. Accessed January, 2022.

Abstract

As technologies advance and play an increasingly larger role in our lives, computational thinking—the ability to understand computing concepts and procedures and their role in the tools we use—has become an important part of our training and education in the 21st century. Thus initiatives to improve people's technical literacy have become a top priority in many countries around the world, with programming classes becoming a mandatory part of many K-12 curricula and increasingly available online.

Unfortunately, many find programming intimidating and difficult to learn because it requires learning abstract concepts, languages, and procedures. Programming concepts and languages employ abstract terms and unfamiliar syntax and conventions; yet how they relate to what we already know (e.g., real-life situations and grounding metaphors) is not always explicated in learning materials or instructions. Learning programming procedures is also difficult due to its abstract nature. For instance, while learners generally need to be explicitly trained on the execution steps in order to be able to trace (or abstract) execution steps, how computer programs are executed and what happens (e.g., in memory) in each step are often omitted or presented as abstractions (e.g., loop), obscuring the process for novice learners before they can master the ability to step through the procedure on their own. As a result, they are often forced to mechanically memorize arbitrary conventions, procedures, and rules in programming without forming any intuition about them.

The reason for these difficulties can be attributed to dead-level abstracting, a phenomenon where information is stuck in certain levels of abstraction. High or low, the lack or absence of interplay between abstraction levels makes it challenging to understand new information in a meaningful, efficient, and effective way. Although the ability to "rapidly change levels of abstraction" has been recognized as a key characteristic of computational thinking and its importance has been stressed countless times, instructions in computing education tend to be mired in abstract levels of abstraction and lack opportunities for students to develop this ability to move up and the ladder of abstraction.

This thesis aims to address this problem by proposing a model where learners can move between levels of abstraction. Specifically, I introduce coding strip, a form of comic strip that has a direct correspondence to code, as a tool for teaching and learning programming concepts, languages, and procedures. By using comics that directly corresponds to code, coding strip is a model instantiated from a framework for computational thinking: learners can move between concrete and abstract levels of abstraction to develop a way of thinking about programming concepts, languages, and procedures in terms of real-life situations and objects. To support its use, this thesis contributes methods, tools, and empirical studies to facilitate the design, creation, and use of coding strips. Specifically, this thesis is structured as follows.

- In Chapter 1, I will explain the motivation, research questions, and scope of this work.
- In Chapter 2, I will review relevant studies, theories, and techniques.
- In Chapter 3, I will introduce a formative study conducted to test the feasibility and usefulness of coding strip when the idea was first conceptualized.
- In Chapter 4, I will present a design workshop study where I investigated the design process and tools for coding strip.
- In Chapter 5, I will present CodeToon, a digital authoring tool, developed to further ease the authoring of coding strips with two novel mechanisms—story ideation and auto comic generation.
- In Chapter 6, I will report an in-class study where I administered and tested four use cases for coding strips to university students in an introductory computer science course.
- In Chapter 7, I will discuss the implications of this work and future research opportunities.
- In Chapter 8, I will end with a brief summary of this work.

Acknowledgements

Without the encouragement, guidance, and help of so many people, this work would not have seen the light of day. At the risk of omitting people who deserve to be mentioned, I will do my best to acknowledge those who contributed to this dissertation. However, note that I intentionally avoid listing all names and details to keep this brief. This does not mean their role was any less significant than those whose names are mentioned here.

First, I would like to thank my supervisor, Edith Law, who believed in this work and encouraged me to build my thesis around it. Without her support, advice, and patience, this dissertation would not have come to fruition. Thank you, Edith, for your friendship and encouraging me to follow my curiosity and passion.

Several mentors and collaborators were instrumental in making this dissertation possible. Ed Lank suggested running a design workshop with teachers attending the annual summer conference at Waterloo for my first study. He also advised me to connect with Celine Latulipe and seek her expertise in CS education. Celine played an important role in the in-class study. Her perceptive feedback significantly improved this work. Dan Vogel suggested teaching an introductory computer science course and testing my work, which led to my in-class study. For the CodeToon study, Jian Zhao recommended the comic evaluation study and helped articulate the contributions of this work. Martinet Lee helped with the planning and running of design workshops. Ken Jen Lee helped run the in-class study. Bernadette Cheng and Gracie Xia assisted in illustrating the comics.

Colleagues and friends at the University of Waterloo also contributed to this dissertation. Alex Williams and Mike Schaekermann were one of the reasons I decided to join Waterloo. When I was deciding, they met with me to share their experience and gave me confidence that I would also be able to thrive at Waterloo. Without them, this dissertation would not be here. Other faculty and members of the Human-Computer Interaction Lab and fellow graduate students at Waterloo have been equally important in making this dissertation come to life. Their friendship helped me cope with the many challenges that graduate students typically face. A special mention goes to my best friend, Henry Chen, who made my life at Waterloo fulfilling and unforgettable.

Finally, I would like to thank my parents, who deserve all the credit for any of my achievements. I would be nothing without their incredible love and sacrifice. I would also like to thank my Heavenly Father, who filled my life with joy, peace, and hope during this journey. He is why I continue to be inspired by the topic of my research. Without Him, this work would not have been possible.

Dedication

This is dedicated to my parents, to whom I owe everything.

Table of Contents

Li	st of	Figures	xiv
Li	st of	Tables	xix
Pr	eface		1
1	Intr	oduction to Coding Strip	2
	1.1	Thesis Statement	12
	1.2	Research Scope	13
	1.3	Thesis Overview	14
2	Rela	ted Work	15
	2.1	Representations	16
		2.1.1 Using Multiple Representations	16
		2.1.2 Moving between Representations	17
		2.1.3 Representations in CS Education	21
	2.2	Comics	21
	2.3	Creativity Support	25
3	Form	native Study for Coding Strip	29
	3.1	Programming Environment	30

4	Des	ign Pr	ocess & Tools for Coding Strip	35
	4.1	Design	Process & Tools	36
		4.1.1	Design Process	36
		4.1.2	Design Tools	37
	4.2	Design	Workshops	38
		4.2.1	Participants	39
		4.2.2	Procedure	40
	4.3	Result	s	41
		4.3.1	Effectiveness of Ideation Cards	41
		4.3.2	Generated Stories & Comics	42
		4.3.3	Ease of Design Process	43
		4.3.4	Perceived Utility for Teaching and Learning	45
	4.4	Discus	sion	45
		4.4.1	Extensions and Limitations	45
		4.4.2	Extensibility of Ideation Cards and Design Process	46
		4.4.3	Design Considerations in Coding Strip	46
	4.5	Conclu	nsion	53
5	Aut	horing	Tool for Coding Strip	54
	5.1	CodeT	Coon: Computational Approach	55
		5.1.1	Design Process and Design Goals	56
		5.1.2	User Interface	57
		5.1.3	Usage Scenarios	60
	5.2	Genera	ating Comics from Code	62
		5.2.1	Story Ideation	62
		5.2.2	Auto Comic Generation	63
	5.3	Evalua	ation	66
		5.3.1	Part 1: User Study	67

		5.3.2	Part 2: Comic Evaluation	68
	5.4	Result	.s	69
		5.4.1	RQ1. Does Code Toon support the authoring of coding strips?	70
		5.4.2	RQ2. Does CodeToon make the process of authoring coding strips more efficient?	75
		5.4.3	RQ3. What is the perceived utility and use cases of CodeToon for teaching and learning programming?	76
		5.4.4	Does CodeToon help generate high-quality comics? (RQ4) \ldots	79
	5.5	Discus	ssion	80
		5.5.1	Implications & Opportunities	80
		5.5.2	Limitations and Future Work	82
	5.6	Conclu	usion	83
6	Use	Cases	for Coding Strip	85
	6.1	Metho	ds	85
		6.1.1	Course & Student Information	86
		6.1.2	Use Cases	86
		6.1.3	Survey	89
	6.2	Result	······································	91
		6.2.1	Demographics	91
		6.2.2	Analysis of Each Use Case	91
		6.2.3	Analysis of Overall Experience	96
		6.2.4	Analysis of Demographics	97
	6.3	Discus	ssion	98
	6.4	A Sun	nmary of Use Cases	99
	6.5		usion	101

7	Dise	cussion	102
	7.1	Summary of Findings	103
		7.1.1 Designing Coding Strip with Ideation Workflow	103
		7.1.2 Creating Coding Strip with CodeToon	104
		7.1.3 Using Coding Strip	105
	7.2	Implications & Opportunities	105
	7.3	Future Directions	107
	7.4	Limitations	112
	7.5	Summary of Contributions	112
8	Con	clusion	114
R	efere	nces	116
A	PPE	NDICES	131
A	Mat	terials for the Formative Study	132
	A.1	Pre-Study Survey	132
	A.2	Task Instruction	133
	A.3	Post-Study Survey	133
в	Mat	terials for the Design Study	135
	B.1	Pre-Study Survey	135
	B.2	Task Intruction	136
		B.2.1 Sketch Warm-Up	137
		B.2.2 Translate Story to Code	137
	B.3	Discussion Questions	137
	B.4	Post-Study Survey	138

\mathbf{C}	Mat	terials for the CodeToon Study	148
	C.1	Part I. User Study	148
		C.1.1 Pre-Study Survey	148
		C.1.2 Post-Study Survey with CSI	152
		C.1.3 System Usability Scale	155
		C.1.4 Paired Factor Comparison	156
	C.2	Part II. Comic Evaluation Study	158
		C.2.1 Survey	158
	C.3	CodeToon Implementation Details	161
D	Mat	terials for the In-Class Study	163
	D.1	Post-Study Survey	163
G	lossa	ry	170
A	bbre	viations	171
N	omer	nclature	172

List of Figures

1	My Ph.D. journey in a nutshell. (Credit: Edith Law & Bernadette Cheng wrote and illustrated this comic, respectively.)	1
1.1	Examples of levels of abstraction in computing	3
1.2	Examples used to introduce abstraction ladder in [73] \ldots \ldots \ldots	4
1.3	Comic using piggy bank as a metaphor for variable	8
1.4	An example of how comics can be used to convey code semantics	9
1.5	Comic examples demonstrating how comics can be used to explain program- ming procedures and highlight different aspects (e.g., procedural, cyclic)	10
1.6	While animation (a) shows one scene at a time, comic (b) shows multiple scenes at once and provides users greater control over the pace at which they process the information [162]. Furthermore, being able to see all the scenes altogether allows one to discover any relationship (e.g., connection between the scenes) more easily.	11
1.7	Coding strip aims to bridge programming language with the visual language of comics. As shown, it provides comic strip and its corresponding code	12
2.1	Bruner's framework for concreteness fading, a method for teaching abstract concept by introducing it in three stages with decreasing levels of concreteness [137].	17
2.2	Tools that allow users to switch between different representations	19
2.3	Representations used to visualize programming concepts and procedures .	22
2.4	Panel placement patterns, which make up the external compositional struc- ture proposed by [105]	23

2.5	An example from [31] showing how the theory of visual narrative grammar can be used to identify the role of each panel in the narrative arc.	24
2.6	Prior work on supporting design process with ideation card-based toolkit .	26
2.7	General framework for auto-generation of comics suggested by Zeeders [166].	27
3.1	Learning progression based on Bruner's framework for concreteness fading which introduces recursion in three stages with representations in decreasing levels of concreteness [137]	30
3.2	Double Recursion: Abstract code	31
3.3	Participants' notes that demonstrate how some learners solved problems differently	33
4.1	Three high-level stages in the design process, with trigger & scenario cards supporting transition from concept to story and design cards from story to coding strip.	36
4.2	Design board with coding strip on recursion made by a teacher group at our workshop	37
4.3	Examples of ideation cards	38
4.4	Example used to introduce the idea of coding strip during the workshops. Participants were asked to guess the underlying concepts/code from the comic strip before seeing its code, to show how comics can be used to intro- duce concepts and start a discussion.	40
4.5	Design workshops with groups of students (left) and high school computer science teachers (middle & right).	43
4.6	Design ideas generated by design patterns and the resulting coding strip on boolean by a student group	44
4.7	Design card types and their examples	48
4.8	Three Code (Execution) \leftrightarrow Comic (Panel) mapping patterns	49
4.9	Three Code (Line) \leftrightarrow Comic (Panel) mapping patterns	50
4.10	Examples of code structures and the layout	51
4.11	Grouping layout. The three panels on the right form their own group both semantically and structurally.	52

4.12	Nonlinear Layout
5.1	CodeToon helps users create stories and comics from code. It uses 1-to-1 mapping to make the connections clear across code, story, and comic. For instance, as indicated by the dotted line, line 1 (code) maps to line 1 (story) and to row 1 (comic)
5.2	System interface: (A) drop-down allows users to check potential code examples for basic programming concepts, (B) code container, (C) button that generates story template from code in code container, (D) story template, (E) drawing canvas for comics, (F) tool palette, (G) style palette, and (H) buttons for changing the interface layout (between code & story, current, or canvas-only layout).
5.3	Users can add code by selecting code example (b) or by typing (c) 5
5.4	Users can add story (to story template) by selecting a list of ideas in drop- down (b) or by typing into input box
5.5	Users can auto generate comic by selecting either of the arrow buttons. If canvas already has some drawing elements, as in this case, they can add the comic to the right (A) or below (B) the existing elements. If canvas is empty. The comic is added to the center of the canvas
5.6	Users can add templates (e.g., comic panels, speech bubbles, and characters) to the canvas using the library
5.7	A simple loop code and (A) auto generated comic. The bottom two rows represent (B) comic updated with different images (phone) and text (e.g., 'battery' for 'x'). Like in this example, CodeToon can update specific rows, giving users fine-grained control over their stories
5.8	Our story ideation consists of generating a story template aligned with the code structure (structure mapping) and allowing users to fill the template with the help of metaphor suggestions (content composition) 65
5.9	Variable assignment: code, template, example
5.10	Conditional expression: code, template, example
5.11	Other visual vocabularies (comic templates, left) and examples (right). The gray boxes are placeholders for text; the dotted squares for objects (e.g., stick figure); the dotted long rectangles for any visual vocabulary, e.g., variable assignment, loop, and so on

5.12	CodeTooon features' usefulness (1: Not at all useful; 5: Extremely useful) .	70
5.13	Baseline vs CodeToon comparison on the (1) usefulness of the tool for the task and (2) the difficulty of the task with the tool. CodeToon users found the tool more useful for creating comics from code and the task less difficult.	71
5.14	Mapping accuracy for auto generated comics. (1: Not at all accurate, 5: Extremely accurate)	72
5.15	Perceived utility of Baseline and CodeToon for teaching and learning pro- gramming	76
5.16	Interest in teaching and learning with Baseline and CodeToon	77
5.17	Comic evaluation results. Statistical significance ($p < 0.05$) is marked with *.	80
5.18	Comparisons of individual pairs according to their concepts. CodeToon comics were rated as being as good or better than Baseline comics in all cases across all measures.	84
6.1	A Spiderman comic used to introduce the idea that values in variables change $(\mathbf{UC1})$. Here, variables are <i>name</i> , <i>mood</i> , <i>age</i> , and <i>hobbies</i>	86
6.2	Sequence of lecture slides used to introduce code with comics $(\mathbf{UC2})$	88
6.3	Examples of reviewing with clicker questions $(\mathbf{UC3})$	89
6.4	Coding strip example on for loop used for one of the use cases: writing code from comics (UC4). Students were asked to translate the comics into code. A sample submission shows how comics can be interpreted in multiple ways.	90
6.5	Students' assessment of four use cases	92
6.6	Students' assessment of comics in general	94
6.7	Two of the three coding strips used for writing code from comics (UC4). One other coding strip is Fig. 6.4	95
7.1	selecting ideation card type (trigger/scenario/design), (C) panel for view- ing and selecting trigger/scenario/design cards, (D) panel for adding ideas related to given concept/topic, (E) panel for writing stories, (F) canvas for creating comics, (G) style palette, (H) tool palette, and (I) button for max-	109
B.1	Sketching activity	137

B.2	Translate story to code	137
B.3	Design board used in the workshops	141
B.4	Design cards for explanation theme. Number of times each card was used in design workshops: Narration-based Explanation: 4/18; Conversation-based Explanation: 4/18; Conversation-based Interaction: 4/18; Character-based Explanation: 5/18; Legend: 1/18	142
B.5	Design cards for sequence theme. Number of times each card was used in design workshops: Assign: 6/19; Passing On: 3/19; Annotated Transition: 4/19; Numbered Transition: 3/19; Highlighted Transition: 3;/19 Point-of-View Switch: 0/19	143
B.6	Design cards for selection theme. Number of times each card was used in design workshops: What-If: $6/16$; Alternatives: $3/16$; Comparison: $2/16$; Is-It: $5/16$	144
B.7	Design cards for repetition theme. Number of times each card was used in design workshops: Conditional Repetition: $5/8$; Counted Repetition: $3/8$.	145
B.8	Design cards $(1/2)$ for presentation theme. Number of times each card was used in design workshops: Moments: $1/27$; Visualization: $3/27$; Concretization: $0/27$; Abstraction: $2/27$; Flowchart: $5/27$; Zoom: $1/27$; Grouping: $0/27$	146
B.9	Design cards $(2/2)$ for presentation theme. Number of times each card was used in design workshops: Multiple Scenarios: $6/27$; Gradual Reveal: $2/27$; Question&Answer: $2/27$; Lens: $0/27$; Break-the-fourth-wall: $0/27$; Visualization: $3/27$	147
C.1	Implementation of computational pipeline for (a) generating story template and (b) generating and updating comic from story.	162

List of Tables

4.1	Concepts used in our study. Bolded are the concepts that workshop partic- ipants chose for their coding strips.	39
4.2	Breakdown of interacting factors that influence the design of coding strips and resulting design variations.	47
5.1	Examples of code and corresponding stories	63
5.2	CSI Results. CodeToon performed better on every factor in creativity support. Statistical significance $(p < 0.05)$ is marked with *	73

Preface



Figure 1: My Ph.D. journey in a nutshell. (Credit: Edith Law & Bernadette Cheng wrote and illustrated this comic, respectively.)

This comic (Fig. 1) summarizes my Ph.D. journey, showing moments from the first time I visited the school to my thesis defense and graduation.

The first four panels illustrate my trip from Seoul, South Korea to Waterloo, Canada to meet my supervisor for the first time during my school visit. The remaining panels show the milestones in my journey: conceiving the idea of coding strip (Chapter 1), running design workshops (Chapter 1), running design thoring tools (Chapter 5), teaching an introductory CS course (Chapter 6), and defending my thesis.

For me, this comic serves as a reminder that small moments can add up to something wonderful someday. I hope this work inspires those who need this reminder and brings as much joy to its readers as it did to me.

Chapter 1

Introduction to Coding Strip

Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.

Jeannete Wing [155]

Levels of Abstraction in Computing

Computing literacy has received enormous attention in the last decade, with countries around the world rushing to make computing education mandatory in schools, to equip students with "the literacy of the 21st century" [65, 67]. This surge of interest created the need to specify what computational thinking is. Although more than a decade has passed since Wing's article on computational thinking [155], there remains much disagreement and confusion over what it encompasses [23, 133].

One thing many agree on, however, is the importance of 'abstraction' and 'decomposition' in computing [91, 75, 110, 135, 68], which Donald Knuth described as transitioning between "levels of abstraction" [70]. Juris Hartmanis and Dijkstra explained that since computer science deals with phenomena at an enormous range of scales (for instance, see Fig. 1.1), computer scientists must reason at varying levels of abstraction [43]. Building on these ideas, many have used the concept of levels of abstraction to model the stages of problem solving [135, 6, 38], teach programming design principles [91, 153, 148],

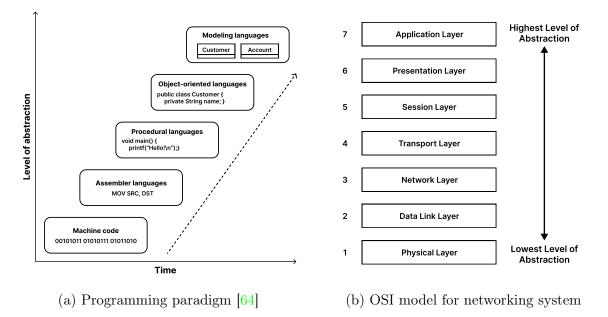


Figure 1.1: Examples of levels of abstraction in computing

introduce computing concepts [9, 147], assess students' learning progress in computer science [135, 74, 46], and describe computational thinking [155, 46, 82].

Although it is undoubtedly a critical aspect in computing, the notion of abstraction levels is not necessarily unique to computer science and engineering [68]; in fact, it has been around longer than the field of computer science itself [61], and studied in a wide range of disciplines, such as philosophy [119, 50], linguistics [42], mathematics [50, 128], history [48], and law [61]; this should not be surprising given how it forms the basis of our communication and concept formations: many scholars have claimed that "all we know are abstractions" [72]. As shown in Fig. 1.2, they specify levels of concreteness (specificity) in our language, as well as complexity of explanations and ideas and knowledge [73]. For instance, *Wired*—a popular magazine focusing on how emerging technologies affect culture, the economy, and politics—produces a show called '5 Levels' featuring episodes titled '_____ Explains ______ in 5 Levels of Difficulty' (e.g., 'Computer Scientist Explains <u>Machine Learning</u> in 5 Levels of Difficulty') where experts from various fields explain a concept in their field to five different demographic groups: child, teen, college student, grad student, and expert [156].

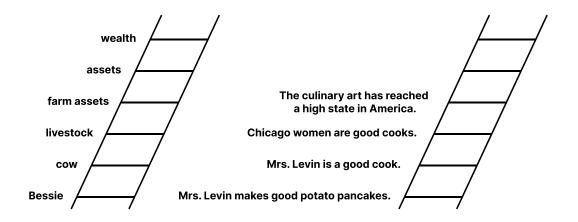


Figure 1.2: Examples used to introduce abstraction ladder in [73]

Series of Abstractions Obscures or Severs Connection to Intuitive Grounding

A problem that emerges in disciplines with many layers of abstraction is that it conceals or severs connection to grounding abstractions (e.g., metaphors) and produces new abstractions [123]. In technical domains, these abstractions often take the form of symbolic representations for efficiency and effectiveness. For instance, computational languages in domains such as math and computing take the form of text-based languages that have language-specific syntax, conventions, and rules. The combination of compact syntax and notation allows them to be effective at communicating instructions unambiguously and delegating computing tasks efficiently. While this requires learners to learn how to command them (e.g., how to use certain constructs in new programming languages), the disconnection makes it challenging for people to develop intuitive and deeper understanding of the computational concepts, languages, and procedures. Abstract terms (jargons) and arbitrary conventions can create an impression to learners that computational ideas and procedures have no apparent relevance to our world. As Giraffa et al. [60] put it: "in learning programming, [students] need to imagine and comprehend many abstract terms that do not have equivalents in real life: how does a variable, a data type, or a memory address relate to a real-life object?" Programming also requires learners to be able to trace a sequence of execution steps, a task that novices find abstract [161] because procedures are often presented as an abstraction (e.g., loop [41]), without delineating steps of the sequence. Consequently, without explicit instructions on how abstract symbols and syntax, as well as computational processes, map to real-life situations and objects, learners can get stuck in the details and struggle to move (reason) beyond the programming level. Thus,

to promote computational thinking, it is important to address this disconnection and the lack of support for interplay between abstraction levels. This thesis aims to contribute to this.

From now on, we will use the term abstraction ladder interchangeably with the term levels of abstraction. Already a popular term in computing [151, 123], it helps our discussion to include the relevant literature that introduces key ideas in this thesis. Introduced by S.I. Hayakawa [73], the term is widely used in various fields ranging from communications to computing. An example that Hayakawa gives for the abstraction ladder is 'Bessie the cow.' As shown on the left of Fig. 1.2, the ladder starts with 'Bessie the cow' at the lowest level of the ladder; it abstracts to 'cow' and eventually to 'wealth.' (The other example on the right of Fig. 1.2 shows the abstracting of an idea.)

Computer Science and Engineering is a field that attracts a different kind of thinker...they are individuals who can rapidly change levels of abstraction, simultaneously seeing things 'in the large' and 'in the small.' Donald Knuth [70]

While Knuth highlighted moving between abstraction levels as a skill specific to computer scientists and engineers, Hayakawa argued that it is an important skill for everyone: "the interesting writer, the informative speaker, the accurate thinker, and the same individual operate on all levels of the abstraction ladder, moving quickly and gracefully and in orderly fashion from higher to lower, from lower to higher" [73]. He reinforced this by specifying the danger of remaining at certain levels of the abstraction ladder—a phenomenon that the linguist Wendell Johnson termed dead-level abstracting [85]. Here is Wendell's explanation [85]:

The **low-level** speaker frustrates you because he leaves you with no directions as to what to do with the basketful of information he has given you. The **high-level** speaker frustrates you because he simply doesn't tell you what he is talking about... there is little for one to do but daydream, doodle, or simply fall asleep.

This notion of dead-level abstracting highlights that the key to effective communication is the dynamic interplay between levels of abstraction. Moving up and down the ladder makes our ideas more accessible and meaningful by helping us develop a deeper understanding of how they connect to specific instances, actions, and procedures. Since the success of teaching and learning depends on effective communication (the major activity of educationists is communication [131]—teaching is communicating and learning processing the communicated knowledge) [117, 118], the proposed research adopts this idea to support the interplay within abstraction ladder in computing education.

Existing Approaches: Storytelling, Animation, and Concrete Representations

Although instructions and activities in computing education tend to be mired in abstract levels of abstraction and lack opportunities for students to reason at multiple levels, there have been various approaches to address this:

- Concept: Storytelling [87], metaphor [152, 35, 49], and analogy have been popular choices for making abstract concepts more intuitive and engaging. CS unplugged is a kinesthetic learning method in which students learn computing concepts through hands-on activities without the use of computers [107]. However, the main limitations of metaphors and analogies are that they may not map perfectly to concepts and lead to confusion.
- Language (Syntax & Semantics): Turning abstract text-based programming into more concrete, graphic-based languages has been a popular approach. Graphic languages such as Scratch [120] (a block-based programming language) provide programming constructs resembling LEGO blocks, making code expressions more accessible to younger audiences. Physical (tangible) programming has also been studied to make programming more approachable, hands-on, and accessible to younger students and the visually impaired [100, 103]. A set of simplified text-based commands specific to a game environment has also been explored [94].
- **Procedure**: Tracing execution steps in a program is a skill novice learners need to master, but the challenge is that in many cases all the steps are not necessarily shown and instead presented as abstractions. For instance, when describing a program, it is common to describe the number of times a program loops but omit what happens and how memory changes in each iteration. While this is to focus on the structural (high-level) aspects of the program, this makes procedures abstract for learners who have not yet mastered the ability to process the steps in their head. To address this, many have researched program visualization and developed new tools; they used animations of visual objects and models (e.g., characters, robot, and animals in Scratch, Gidget, Logo) as proxies to show how the code works; similarly, Storytelling Alice [37] illustrated procedures in terms of characters and stories; others (e.g., Python Tutor) have used interactive visualizations with diagrams, where learners can step through each line of code and observe changes in memory state, to understand the execution sequence and what each line of the code does to memory states [66].

Our Approach: Bridging Abstract with Comics

Motivation. While prior approaches mentioned above helped lower the barrier to entry into programming, learners must eventually transition to text-based programming [101]. Although several transition methods have been proposed [76, 88, 53] and they vary in specifics, researchers agree that explicitly linking the concrete with abstract programming benefits transfer [88, 55]. Thus, I set out to build a model for layered representations where its theoretical framework is abstraction ladder. (The term 'layered' is meant to suggest a tight coupling of these representations.) The reason for using the term layered representations—as opposed to abstraction ladder—is to be precise and clear that our abstraction levels consist of different representations. Note that there can be abstractions of increasing/decreasing concreteness within the same representation (e.g., 'Bessie the cow' example shown in Fig. 1.2). Layered representations makes it explicit that we are using multiple ($n \geq 2$) layers of representations—in our case, comic and code as the layers. (This is not to say that coding strips can consist only of comic and code. As will be shown in Fig. 1.4 and Chapter 3, there can be another representation or the same representation as comic or code but vary in concreteness.)

What is Comics? Comics is a term that refers to a medium that uses images to convey ideas or narrate stories. Often, it takes the form of a sequence of images and uses "a combination of text and other comic elements such as speech balloons and caption" [149]. While some suggest the first time this art form emerged in the human history can be traced all the way back to Lascaux cave paintings [98], the term comics and today's form of comics (with panels and speech balloons) first appeared in the late 19th century. There have been many different forms of comics and debates surrounding its definition. For instance, the early comics ('cartoons') had a single image as opposed to a sequence of images. While some scholars considered them as a form of comics, others excluded them by defining it as "juxtaposed pictorial and other images in deliberate sequence," implying that it needs at least two images to be "juxtaposed" [98]. In this thesis, I do not argue for a particular definition of comic. The comics produced in this thesis generally consist of more than one panel, but it had nothing to do with the definitions of comics. That is, I did not impose any rule to produce comics that abide by any particular definition (e.g., at least two panels).

Why Comics? When justifying the use of comics, many works highlight the power of the medium to engage and offer various cognitive benefits, citing its close ties to theories such as cognitive theory of multimedia learning [98, 97, 154]. While they are part of the reason, I argue for the use of comics based on the medium's (1) expressiveness and (2) unique affordance—both of which make it a powerful medium suited to teaching and learning computational ideas and procedures.

(1) Expressiveness. Below, I explain how the visual language of comics can 'concretize' abstract concepts, languages, and procedures in computing.



Figure 1.3: Comic using piggy bank as a metaphor for variable.

- **Concept**: Comics is a unique medium that can leverage the power of storytelling, metaphors, and analogies. Fig. 1.3 shows Jane putting \$1 to her piggy bank. In here, piggy bank is a metaphor for variable that stores something (\$1 in this case). Jane putting money into it is analogous to a value being assigned to a variable in semantics. On top of storytelling, metaphor, and analogy, comics adds visual representations to the mix. Thus, it can arguably provide (or leverage) the best of both worlds, offering cognitive benefits visual representations possess.
- Language (Syntax & Semantics): Code syntax and semantics can be illustrated and explained by using comics in a creative way. For instance, Fig. 1.4 shows how comics can be used to scaffold a piece of code. Students see a comic (visual language) about the movie *Dr. Strange* where a character uses the Time Stone to start a time loop to repeat the same moment; then students see this expressed in English (natural language), and then in code (programming language). The example shows how comics can be used to teach the language syntax and semantics: the indented structure carries over from comic to English and then to code to teach this syntax; the repetition of the same panels provides visual clarity for users to grasp what the programming construct while does; comics can help users better understand and retain the idea longer by leveraging a familiar story and visuals; the code semantics for 'print()' is also explained with the action of a character speaking, providing intuition by associating it with a familiar abstraction.
- **Procedure**: Comics can leverage its expressiveness to illustrate computational procedures in creative ways. Fig. 1.5 shows two comic examples on loop. The two examples

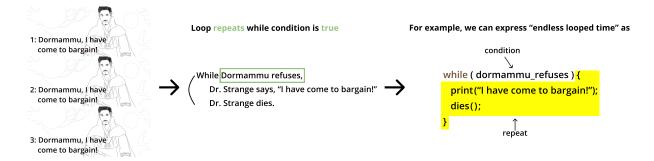


Figure 1.4: An example of how comics can be used to convey code semantics

represent two slightly different ways of viewing (and visualizing) loops. Fig. 1.5a illustrates counted loop (e.g., for), visualizing a loop with fixed number of iterations. Fig. 1.5b illustrates conditional loop (e.g., while), where the number of iterations is not pre-determined. Thus the design for Fig. 1.5a is simply procedural, whereas that of Fig. 1.5b is cyclic since the end of the loop is unspecified.

(2) Affordance. With its unique affordance, comics provides certain cognitive benefits that cannot be found in similar media such as animation. As shown in Fig. 1.6, comics differs from animation in that sequence of events can be juxtaposed and shown all at once using panels—not to mention various ways in which panels can be placed in creative ways to enhance the delivery and reading experience [34, 15]. While the dynamic nature of animation may be more effective in terms of engaging viewers, comics offers unique benefits in some areas, e.g., leveraging multiple panels to allow viewers to compare and contrast events occurring at different times and helping them understand their connections. Also, Yang [162] found that his students who received his math instructions in the form of both comics and video preferred comics: with video, students had to constantly rewind it to re-watch the parts they did not understand; with comics, they could read at "their pace," which empowered them as they do not have to re-wind and be reminded again about the parts they did not understand. Jonathan Hennessey explains the unique affordance of comics this way [96]:

"In one image or composition, the reader can linger over the potential significance of small details without having the sense that the narrative flow has been disrupted. No matter how much time you spend inside one panel, you never feel like the story has stopped or altered tempo. Not the way you would if you pressed 'pause' while watching a video. So it's ideal for students and teachers. It has the vividness of the moving image and the complexity of text."



(a) Comic on counted loop showing itera- (b) Comic on conditional loop using arrows tions to illustrate steps to highlight control flow

Figure 1.5: Comic examples demonstrating how comics can be used to explain programming procedures and highlight different aspects (e.g., procedural, cyclic)

The unique affordance of comics can also be useful in the context of teaching programming. In computing courses, novice learners often vary in their ability to keep up with the teaching. Being able to control the 'tempo' can be useful for students who have difficulties keeping up with the instructions in the class. Also, when an instructor takes a piece of code and steps through the execution sequence, some students can get confused about a certain explanation or step but have difficulty specifying which part or step they were confused about; with comic, students can more easily pinpoint which part they were confused about.

Coding strip. As shown, comics is a medium with a great potential that is currently underexplored. Building on this observation, this dissertation aims to build layered representations with comic and code as the two layers to support the interplay between concrete (comic) and abstract (code); I term this form of comic strip (Fig. 1.7)—one with corresponding code—as coding strip (coined by mixing coding and comic strip), to distinguish it from the form of comics that uses comics primarily to narrate (explain) programming concepts without making any connection to code.

Fig. 1.7 shows a coding strip example on recursion. The labels (i.e., (a), (b), (c)) and boxes surrounding the panels and lines of code specify the mapping between the panels and the lines of code, with the example consisting of three main parts: (a) forward recursion,

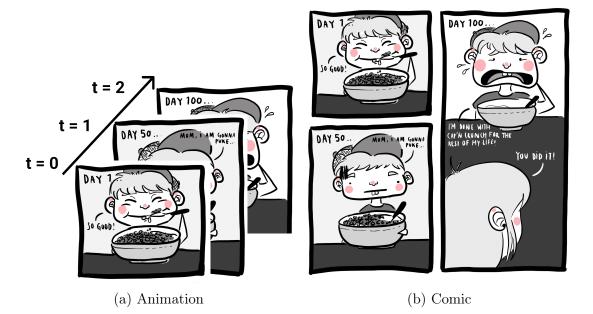


Figure 1.6: While animation (a) shows one scene at a time, comic (b) shows multiple scenes at once and provides users greater control over the pace at which they process the information [162]. Furthermore, being able to see all the scenes altogether allows one to discover any relationship (e.g., connection between the scenes) more easily.

(b) base case, and (c) backward recursion. In Fig. 1.7, the story begins with a girl arriving at a concert. She debates returning home because she suspects the waiting line may be too long. She decides to find out how many are in front of her, so she asks a man in front of her a question (a, forward recursion): "how many are in front of you?" The man in front of her asks the same question to a girl in front of him: "how many are in front of you?" This continues until it finally reaches a girl who does not have anyone in front of her (b, base case). She turns back and tells the boy behind her that there is no one ("zero people") in front of her (c, backward recursion). Then he turns back and tells the girl behind him: "one person," a number derived by adding one to the number he was given by a girl in front of him. The girl behind him does the same. This continues until it reaches the girl who initiated the question. Although not shown here, the story ends with the girl saying, "That's not too bad! I will stay then!"

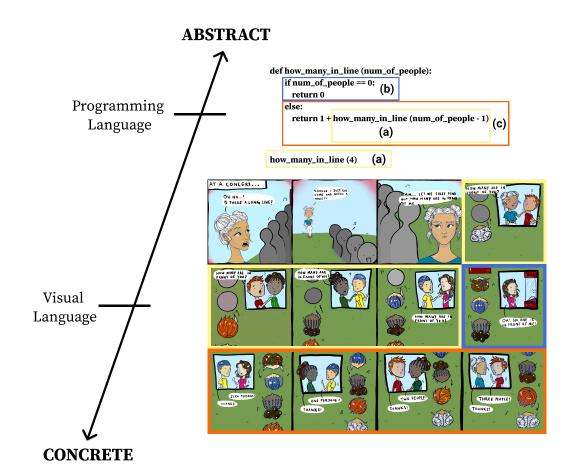


Figure 1.7: Coding strip aims to bridge programming language with the visual language of comics. As shown, it provides comic strip and its corresponding code.

1.1 Thesis Statement

Studying programming is difficult because it requires learning concepts, languages, and procedures that have been abstracted through a series of abstractions. To make them more accessible, many approaches have been explored. One promising medium that has not been examined yet is comics. Because of its sequential nature and ability to present abstract concepts and procedures in a meaningful way through its visual storytelling, comics can be an effective medium. While there have been comic books about programming, they have been used to simply narrate programming concepts. Comics that have a direct correspondence to code (which I term coding strips) can have multiple benefits, such as being able to explain the code expressions and execution steps in a meaningful way by connecting abstract terms to real-life objects and presenting procedures in terms of reallife situations. Since teaching abstract concepts, languages, and procedures is a common challenge in most, if not all, technical disciplines such as engineering and mathematics, the methods for mapping them to real-life situations and objects in the form of comics could inspire ways to address the challenge technical disciplines face. The remainder of this dissertation is structured around the following statement:

In this dissertation, I argue that a form of layered representations that consists of code and comic offers a useful and enjoyable way to learn programming, and that mapping between them can be done efficiently by leveraging the conceptdriven storytelling procedure, combination of structure mapping, metaphor, and visual vocabulary.

To defend this statement, the remainder of this dissertation attends to the following research questions:

RQ1: How do we design coding strips? What is the design process?

RQ2: How do we facilitate the authoring of coding strips?

RQ3: How do we use coding strips for teaching and learning programming?

Next, I describe the scope of this thesis and provide an overview, summarizing research contributions and key ideas presented in each chapter.

1.2 Research Scope

The goal of this thesis is to introduce a new concept called coding strip, outline its design space, and propose methods for visualizing abstract programming concepts, languages, and procedures in the form of comics. While I report perceived (self-reported) benefits of coding strips from teachers with many years of teaching experience and students with experience in programming, the scope of this research does not include the collection of a significant amount of quantitative evidence to demonstrate learning benefits. There is a long history of related work that provides theoretical and empirical support for using visual representations, multiple representations, and comics—all of which coding strips leverage. However, in this thesis, I leave large-scale empirical studies that affirm the learning benefits of coding strips as future work and focus on developing methods and tools to support their design, creation, and use.

1.3 Thesis Overview

This thesis explores coding strip and methods for mapping programming concepts, languages, and procedures to real-life equivalents expressed in the medium of comics. Below is a summary of each chapter.

Chapter 2: The topics relevant to the idea of coding strip and methods for designing and creating them are presented.

Chapter 3: A formative, in-lab study that explored the feasibility of coding strip is presented. This preliminary, qualitative study demonstrates one way coding strips can be used and shows how they can help students learn the concept of recursion.

Chapter 4: I introduce an ideation-based method for creating coding strips. This method contributes a design process and ideation cards for the design of coding strips. Based on the analysis of generated comics, the design space for coding strips and various design considerations are discussed.

Chapter 5: I introduce a computational method for creating coding strips. This method uses two creativity support features, story ideation and auto comic generation, to expedite and automate parts of the design process found in Chapter 4. CodeToon, a system that leverages this computational method, is described. I present evidence that CodeToon speeds up the creation process and produces higher quality coding strips.

Chapter 6: I present an in-class study where I administered and tested four use cases of coding strips that represent common teaching tasks in programming classes. The report contributes perceived benefits and challenges of these use cases.

Chapter 7: I summarize the findings from each study. Based on these findings, I discuss their implications and open questions, and conclude with future research directions.

Chapter 2

Related Work

The most powerful way to gain insight into a system is by moving between levels of abstraction. Many designers do this instinctively. But it's easy to get stuck on the ground, experiencing concrete systems with no higher-level view. It's also easy to get stuck in the clouds, working entirely with abstract equations or aggregate statistics... the ladder of abstraction [is] a technique for thinking explicitly about these levels, so a designer can move among them consciously and confidently...an essential skill of the modern system designer will be using the interactive medium to move fluidly around the ladder of abstraction.

Bret Victor [151]

This thesis builds on three areas of research: (1) **representations** to identify rules and methods for building layered representations, (2) **comics** to understand and leverage its design language and utility, and (3) **creativity support** to develop and evaluate tools for coding strips. In the following, I review the relevant work in each area.

2.1 Representations

Coding strip is a form of layered representations. It consists of multiple representations that share invariant relations. In the following, I review relevant work on using multiple representations and moving between them.

2.1.1 Using Multiple Representations

Our work touches on learning with multiple representations, which has been an active area of research. The field is driven by years of evidence showing benefits of (appropriate) representations to learning. Ainsworth names three functions of multiple representations [3]: (1) computational offloading, (2) re-representation, and (3) graphical constraining. Computational offloading refers to the extent to which "different representations can reduce the amount of cognitive effort required to solve equivalent problems." For instance, compared to text, diagram or flowchart can reduce cognitive effort required to search and recognize the underlying relationship and process by chunking related information (abstracting) and using arrows to make the procedure clear. Re-representation refers to how creating a new representation sharing the same abstract structure as the original representation but differing in the amount of information (e.g., 2D vs 3D shape or diagram vs metaphor) differentially influences learners' ability to solve problems [109]. For instance, Zhang and Norman [167] showed that problem solving improved when the isomorphic version of Tower of Hanoi was re-represented with more external information. Graphical constraining relates to the "range of inferences that can be made about the represented concept." For instance, text (symbolic representation) and graphics (iconic representation) differ in the level of expressiveness and conversely in the amount of ambiguity [136]. Ainsworth notes that the lack of ambiguity in diagrams (in contrast to, for instance, text in word problems which can elicit different images in each learner's mind) makes them more effective for solving determinate problems (e.g., math problems).

The motivation for supporting learning with multiple representations is that representations excel in different ways and each representation is more suited for particular cognitive tasks in the learning process. While leveraging more representations to serve appropriate representations has been considered a sensible approach, researchers found that to harness the power of multiple representations, they need to address the problem of learners struggling to notice the relationship between representations: many studies found that learners struggle to understand how representations are mapped to each other [8, 4], which can end up removing the benefits of multiple representations and even hinder learning [45]. To address this, prior work explicitly explained the connection verbally or illustrated them visually through annotation (e.g., arrows pointing from one representation to another) in many multi-representational systems [66].

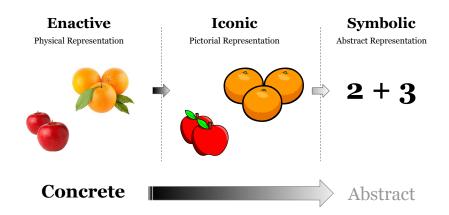


Figure 2.1: Bruner's framework for concreteness fading, a method for teaching abstract concept by introducing it in three stages with decreasing levels of concreteness [137].

2.1.2 Moving between Representations

As we think about using layered representations to move up and down the ladder of abstraction, we need to consider two cases. The first case assumes that we already have layers of representations. In this case, we are interested in how we move between the layers and what benefits or challenges, if any, there are. The second case concerns when we do not yet have layers of representations and need to create them. Here, we face two questions: (1) how do we conceive abstractions at different level(s)? (2) What are the design considerations? To address these questions, concreteness fading, structure mapping theory, and one interactive article on moving up and down the ladder of abstraction are reviewed.

Concreteness Fading

One technique that addresses the above-mentioned challenge of learning with multiple representations is concreteness fading [55]. It is a learning progression, involving two or more stages, in which a concept is illustrated using multiple representations in decreasing levels of concreteness, while its core idea(s) remain intact across the representations in stages [137, 141]. Over the years, it has amassed theoretical and empirical support and is now recognized as an evidence-based instructional technique that allows learners to relate

abstract concepts to real-world objects and scenarios. As shown in Fig. 2.1, representations fade in concreteness—in this case, from concrete, physical representation to abstract, symbolic representation.

The theoretical framework of concreteness fading was first conceptualized by Bruner [24] as an instructional technique to facilitate the learning and transfer of abstract ideas. He believed that new concepts and procedures could be more effectively explained when presented in three progressive forms (Fig. 2.1)—enactive form, a physical or concrete model of the concept; iconic form, a graphic or pictorial model; and symbolic form, an abstract model of the concept. Bruner argued that grounding a concept in a form that is concrete and already familiar to learners can facilitate meaningful learning of foreign concepts. This aligns with a core notion in learning science that highlights the importance of helping learners make connection to what they already know [116, 122, 10]. In mathematics and science education, especially, concreteness fading has shown to be effective in helping students assimilate abstract concepts [54]. Since Bruner, many have extended or modified the 'Enactive-Iconic-Symbolic' model of concreteness fading to formulate new modes of representation. One such adaptation is the 'Concrete-Representational-Abstract' (CRA) sequence, which has been shown to be effective in math education [86, 132]. Recent works have defined concreteness fading more broadly to mean any progression from concrete to abstract, without requiring three distinct stages or a physical object as the initial form [84, 55].

While multiple representations can be introduced sequentially or simultaneously, the sequential, fading progression in concreteness fading reinforces the notion that the representations at each stage are mutual referents possessing the same set of invariant relations [141]. Compared to simultaneous presentation of multiple representations, sequential presentation can reduce the need for people to question how one representation relates to another. Moreover, the fading nature of subsequent representations (e.g., as in Fig. 2.1 where the physical representation fades to the pictorial representation and it is obvious that the faded, pictorial representation references the prior physical representation) can encourage concreteness fading to use representations that have some level of visual resemblance, which in turn help people recognize more easily that they are mutual referents of the same concept. Fyfe suggests that this sequential presentation approach is what differentiates concreteness fading from other approaches that present multiple representations simultaneously and highlight similarities and differences to indicate the mapping [130, 104].

That the core idea (invariant relations) must remain across multiple representations is an idea unique to concreteness fading and not highlighted in other multiple representation techniques. In other areas where multiple representations are simultaneously used, the common usage is dyna-linking, where learners act on one representation and see the results of those actions in another representation to understand various functional aspects of the representation they interact with [151]. Thus, the supplementary representation that acts in response to learners acting on one representation does not have to be its 'concrete' or 'faded' version sharing the same core idea, as in, for instance, physical and pictorial representations in Fig. 2.1. The primary function of the supplementary representation is to provide a proxy, an intermediary object, to reveal the relationship between the user's action on the interacting representation and its meaning. Which proxy (representation) is used or whether it shares the same invariant relationship (abstract structure) is not a concern. On the other hand, this is critical in concreteness fading, as the representations are mutual referents of the same concept. As the goal of this thesis is to develop layered representations, which consists of layers of representations referencing the same idea and abstract structure, the theoretical framework of concreteness fading is used to formulate the idea and design of coding strips.

Although the term layered representations is not a widely used term or concept, interest in what can be described as layered representations has increased in recent years, and researchers have developed innovative tools to harness their power. Nazmus et al. developed Noyon (Fig. 2.2a), a tool that allows users to add drawings and instantaneously switch between drawings and mathematical equations with a slider, allowing learners to interact with the (mathematical) idea at different levels of abstraction (i.e., iconic and symbolic representations). Ye et al. developed Penrose, a system that can convert math notations into diagrams [164]. WritLarge offers 'Representations Menu' (shown in Fig. 2.2b), allowing users to change representations along their structural, semantic, and temporal axes [160].

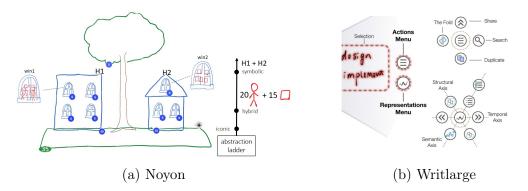


Figure 2.2: Tools that allow users to switch between different representations

Structure Mapping Theory

In Pilosophy in the Flesh: the embodied mind and its challenge to western thought, Lakoff and Johnson argue that the mind is inherently embodied and that abstract concepts are largely metaphorical in nature. That is, our abstract thoughts are grounded in metaphors. Creating a story from code is equivalent to creating a more concrete abstraction of code. Metaphor research can provide useful insights on how to generate abstractions [124, 58], as mapping abstractions has been extensively explored in the field. In our work, we take Gentner's structure mapping theory [57] as a guide. The theory posits that the strength of mapping is determined not by the number of attributes shared between the base and target domains or the specific content of the domains, but by the relations between objects (also referred to as relational structure [57, 11]). That is, while metaphor can be attributional, relational, or both, the theory suggests that relation plays the most important role [58]. An example Gentner provides is: "The hydrogen atom is like our solar system." In this example, what makes hydrogen atom a comparable abstraction is that hydrogen atom and solar system share the same relation (e.g., the electron REVOLVES around the nucleus, like how the planets REVOLVE around the sun), not their object attributes (they do not share the same object attributes, e.g., color, size, as the planets).

Gentner defines abstraction as "a comparison in which the base domain is an abstract relational structure." I use this definition and structure mapping theory to devise a method for generating story from code. Specifically, I posit that we can use code (base domain) as relational structure and allow users to freely choose the content inside this structure, as the theory suggests that specific content of the domain does not determine the strength of mapping. Together with concreteness fading that also highlights the importance of maintaining the relational structure across representations [55, 137, 142], we can infer that the design principle for layers of representations (abstractions) is to maintain the structure across them. These previous work led to a design choice to structure story and comic to align with the code structure. This will be shown in detail later in Chapter 5.

Up and Down the Ladder of Abstraction

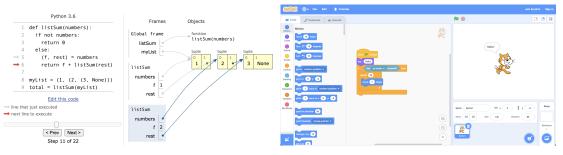
In his interactive article Up and Down the Ladder of Abstraction [151], Bret Victor uses variable as a control for moving up and down the ladder of abstraction. In this article, the variable is 'time,' and a system at a particular 'time' an abstraction; readers use a slider to change the 'time' (e.g., t=1 to t=2) and observe how the system (abstraction)—a car's trajectory—changes. He equates this interaction as moving up and down the ladder of abstraction, explaining that all systems share the same anatomy—an independent variable (e.g., time), structure (the set of rules and what is controlled by the variable), and data (environment)—and suggesting that the process of building the ladder of abstraction (i.e., conceiving abstractions at different levels) consists of identifying what can be parameterized and providing a control to explore the range of abstractions. In Chapter 5, I show a system that, given code input, provides a list of metaphors to help users generate realistic, code-aligned stories, and automatically convert stories into comics. The process suggested in this article informed how we turn code into a story. Specifically, this idea was used to explore which parts of the code can be parameterized and used to develop stories.

2.1.3 Representations in CS Education

One reason novice learners find programming difficult is because of the use of 'text-based' programming languages. Textual representations are called symbolic representations because they consist of symbols. To explain, the text 'apple' itself does not possess any meaning or indicate how it should be pronounced. It is merely a symbol for the actual object it represents. In other words, we have simply learned to associate (map) the text 'apple' with the object and how it should be pronounced. Therefore, with symbolic representations, we have to memorize many arbitrary mappings. On the contrary, with visual representations, no extra mapping needs to be formed, as they could look just like or similar to the actual object. To address the abstract nature of symbolic representations (which stems from the need to form arbitrary mappings in our head), concrete representations have been used. As shown in Fig. 2.3, visual representations for coding exist in many different forms and shapes. Diagram-based tools, such as Python Tutor [66], use abstract shapes (e.g., rectangles) and arrows to show memory changing dynamically as the user executes each line of the code, allowing users to trace code executions and observe what happens with each execution. Storytelling-based tools, such as Storytelling Alice [37] and Scratch [120], use characters and environments as proxies. Users manipulate them with block-based programming or simplified text-based commands [94] and come to understand how programming constructs such as if and loop work by observing how these code expressions manipulate characters and environments.

2.2 Comics

Broadly defined as "sequential art" [33] and "images juxtaposed in deliberate sequence to convey information and/or produce an aesthetic response in the viewer" [98], comics has been returning to spotlight as a literary device and art form and are being used in a number of areas such as in research communication [71], news [27], math [113, 165], literature [150, 21], education [145, 63, 134, 162], and data visualization [15, 12].



grams (e.g., Python Tutor [66])

(a) Visualizing execution steps using dia- (b) Showing effect of programming constructs via animation (e.g., Scratch [120], Logo [18], Storytelling Alice)



(c) Visualizing algorithms using diagrams (d) Tangible programming language (e.g., Visualgo [69]) (e.g., Torino [103])

Figure 2.3: Representations used to visualize programming concepts and procedures

Before this resurgence, comics—for several decades—were dismissed as 'juvenile' and a material 'for kids' due to stereotypes stemming from misunderstanding comics as 'comic books' rather than an art form [98]. Eisner's book *Comics and Sequential Art* and Scott McCloud's book Understanding Comics [98] reignited public interest in comics and made a significant contribution to formal comics studies. Along with Eisner, McCloud helped define comics not merely as a literature 'for kids,' but an elaborate 'visual art form' with unique design language (e.g., gutters and panels) deserving of attention and scrutiny as it is a powerful, expressive medium capable of communicating ideas and stories in an engaging, effective manner. Together with developments like these, interest in studying and adopting the medium has reemerged in the last few decades [13, 15].

Design Language

At a high level, the design of comics can be broken down into its external (i.e., layout of panels) and internal (i.e., content inside a panel) [32, 15]. Cohn offers two frameworks for describing them, External Compositional Structure (ECS) for external and Visual Narrative Grammar (VNG) for describing the internal. As shown in Fig. 2.4, ECS describes how panels are juxtaposed or placed in relation to others. Bach et al. also analyzed comic designs based on layout and content, but more specifically layout and content relations (multiple panels are labeled for content relations, whereas a single panel is labeled in Cohn's VNG) [15]. Based on a matrix created using these two axes, they created multiple design patterns (e.g., Fig. 2.6b) to map the design space for data comics—a form of comics that communicates insights in data with the visual language of comics.

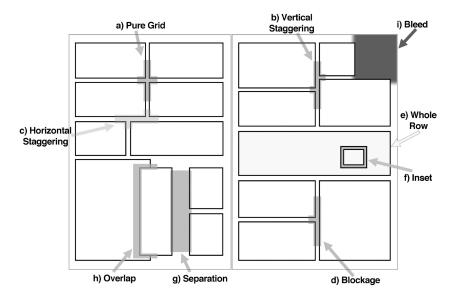


Figure 2.4: Panel placement patterns, which make up the external compositional structure proposed by [105]

While design patterns (e.g., Fig. 2.6b) proposed by Bach et al. [15] specialize in data comics, Cohn's VNG is easier to generalize when it comes to applying to new domains or constructing new sets of comic designs, as it concerns single panels as opposed to multiple panels in design patterns. Concretely, as shown in Fig. 2.5, VNG posits that each panel in the comic can be categorized into any one of the five narrative categories (i.e., stages in a narrative arc) based on its content, which allows for the analysis of the narrative progression as well as the role each panel plays in the narrative. The five suggested categories for VNG are:

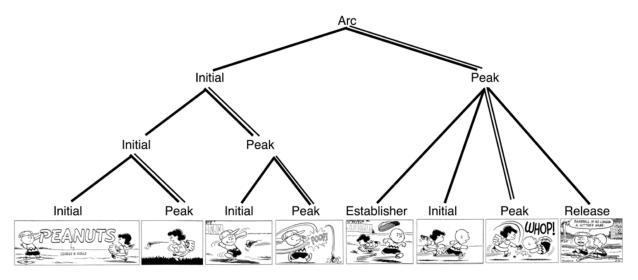


Figure 2.5: An example from [31] showing how the theory of visual narrative grammar can be used to identify the role of each panel in the narrative arc.

- Establisher sets up an interaction without acting upon it
- Initial initiates the tension of the narrative arc
- Prolongation marks a medial state of extension, often the trajectory of a path
- Peak marks the height of narrative tension and point of maximal event structure
- Release releases the tension of the interaction

Utility and Use Cases

Recently, the research communities began to take interest in comics, exposing its design dimensions [33, 12] and the advantages of this medium over other media such as video [162] and infographic [154]. In visualization research communities, data comics is emerging as a genre of its own [14]. A growing number of studies are showing that the expressiveness of the medium allows it to present information in an engaging way, make complicated concepts easier to understand, and help readers retain the information longer than other popular medium such as infographic [154].

There is also a growing interest in using comics in education. Its popularity is evidenced by many studies and online resources that report many ways in which it has been incorporated into teaching practices in various subjects such as math [113], engineering [159], science [79, 2], biology, physics, history, and business [127, 96, 157, 146]. For example, teachers use them to introduce a topic and motivate students; students create comics to illustrate concepts they have learned and use them to teach their peers, which encourages students to reflect on the learned concepts and ground them in familiar situations [113, 114].

A growing number of studies are beginning to elucidate numerous learning benefits that comics offers. Researchers are finding that comics enables teachers to communicate abstract concepts in a more meaningful and engaging manner, promotes discussion, improves students' comprehension of complex concepts, learning experiences, and attitude towards the subject and course [22]. Because comics leverages images and text together, scholars have associated its cognitive benefits with those outlined in the cognitive theory of multimedia learning, which argues that learning with images and text leads to various cognitive benefits when compared to learning with images or text alone [97, 83].

There have been a few attempts to incorporate comics into teaching programming as well. Several comic books—such as *Hello Ruby*, *Lauren Ipsum*, and *Secret Coders*—have been used to introduce computing in classrooms. However, these comics are typically formatted as storybooks without any correspondence to code, which misses out on the opportunity for the learned concepts to transfer. This thesis focuses on coding strip, which I define as a form of comic where the story is accompanied by a piece of code, to enable learners to see how a line of code can map to a meaningful action in real life presented in the medium of comics.

2.3 Creativity Support

Here, I review relevant work that informs the ideation- and computation-based methods for designing and creating coding strips.

Supporting Design Process with Ideation Cards

Ideation cards are design tools that proved useful at supporting the design process [62, 81]. As brainstorming ideas is a first step in every design process [52], many design studies have used ideation cards to support design processes [95, 36, 17, 39]. For instance, Mora et al. [102] developed *Tiles*, a set of ideation cards to help non-experts engage in creating ideas for IoT projects, along with a design board to scaffold the design process. They tested the effectiveness of their tools, ideation cards and design board, by running workshops and asking the participants to assess the perceived usefulness of the tool in supporting their design process. As will be shown in Chapter 4, I also developed ideation cards and design



(a) A design workshop study where researchers tested the effectiveness of their ideation-card based toolkit designed to support brainstorming for IoT projects [102]



(b) An example of design cards provided to participants at data comic workshop to provide design ideas to participants [15]

Figure 2.6: Prior work on supporting design process with ideation card-based toolkit

board as our design supporting tools and tested their effectiveness in the same manner, which is a standard methodology other studies proposing design process and ideation cards also used [62, 95, 36].

Measuring the Effectiveness of Creativity Support

The source of data for measuring the effectiveness of creativity support tools can come from the product (e.g., design ideas) or user. Thus, several metrics have been developed around them. One popular metric has been the number of ideas generated by the end of the session. Another metric is the Creativity Support Index (CSI) questionnaire, which



Figure 2.7: General framework for auto-generation of comics suggested by Zeeders [166].

asks users to assess their perception of how well the tool or system enabled them to be creative [28]. In this thesis, both the total number of ideas generated and the results of CSI were used to measure the effectiveness of creativity support.

Authoring Comics

Many tools and techniques have been proposed to lower the barrier to creating comics. Commercial tools, such as Pixton [1], are widely used by universities and schools to support students engaging in comic creation activities in classrooms and teachers designing learning materials for subjects, such as math, science, business, and history [47, 125, 145, 113, 165]. One step forward from supporting a design process with authoring tools, design patterns, and design guidance is automatically generating comics. There have been quite a few research on automatically generating comics. Cho et al. [29] created AniDiary (Anywhere Diary), which analyzes user's mobile device data and outputs a summary of the user's day in the cartoon-style diary. Kurlander et al. [92] developed Comic Chat, a system that takes chat sessions and visualizes them in the form of comics. Hong et al. [78, 77] proposed a framework called *movie2comics*, which takes a movie and automatically converts it to comics. Their framework contains three components: script-face mapping, key-scene extraction, and cartoonization. Shamir et al. [126] found a process by which their system can transform the interaction of 3D graphics into comics. StripThis!¹ uses scripts with pre-defined commands to generate comics in real time. As a user types the commands into the editor, StripThis! parses the script, imports the stencil images mapped to the commands, and composites them in the comic panel. Machine learning has been used to convert sketches to similar drawings, automatically color the comic, generate layout, and vary the types of speech balloon [59, 25, 163].

¹https://www.kesiev.com/stripthis/

Auto Comic Generation

Zeeders [166]—who analyzed comic creation systems to identify the steps for auto comic generation—suggests that, at a high level, it involves three steps: (1) content creation, (2) translation of content to a (formal) comics description, and (3) graphics creation, as shown in Fig. 2.7. In previous work, the sources of content in the first step have been a multitude of things, e.g., daily activity data [29], chat sessions [92], scripts [?], and movies [78, 163]. (Their work cannot generalize to our content type, code, since it, unlike other data types, lacks contextual information that can be used to form a narrative for comic without user intervening to help define a story.) In the second step, they are formatted into a particular format [7, 166] to provide instructions on how it should be presented graphically. The final step is the graphics creation stage where either the composition or screenshot method is used [166]; the former method composites different images (e.g., character, background, speech bubble) to create a scene for panels, the latter embeds existing scenes (e.g., screenshot of movie scenes) into panels. As will be explained, our work leverages the composition method to automatically generate comics. Overall, this work extends research in this area by demonstrating how we can auto generate comics when working with a new content type, code.

Chapter 3

Formative Study for Coding Strip

Computer Science and Engineering is a field that attracts a different kind of thinker...they are individuals who can rapidly change levels of abstraction, simultaneously seeing things 'in the large' and 'in the small.'

Donald Knuth [70], p. 39

Formative studies are helpful in understanding the potential of a new idea or particular intervention. If the results are clearly negative, it can save researchers' time and resource. This chapter introduces a formative study I conducted in the beginning when the idea of coding strip was first conceptualized. In this section, I describe the details of this study and provide a brief analysis of the results that were used to assess the potential of coding strip.

To understand the feasibility, usefulness of coding strip, and its benefits and challenges, I conducted a preliminary, qualitative study in the context of teaching recursion, an abstract programming concept many learners struggle with, with three coding strips in the form of concreteness fading (i.e., present comic, visualization, and code sequentially as shown in Fig. 3.1).

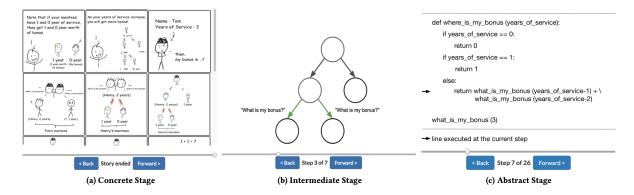


Figure 3.1: Learning progression based on Bruner's framework for concreteness fading which introduces recursion in three stages with representations in decreasing levels of concreteness [137]

3.1 Programming Environment

I developed a programming learning environment that implements concreteness fading. It consists of a simple interface with two panels—the top panel shows a select representation in the concreteness fading sequence, and the bottom panel shows a question that checks the learners' understanding of that representation. Users can step through or review the comic, visualization, and code using the Forward and Back buttons.

The interface helps teach concepts, such as recursion, using four stages instead of Bruner's three stages. This design decision was made during our iterative design study in order to help learners transition from visualization to code more easily. In the following, we describe our choice of representation for each stage.

For the **concrete representation**, we decided not to adhere to the strict definition of concreteness fading where 'concrete' refers to a physical object, and instead chose comics for the concrete stage. We selected comics, since, among many reasons, they are a powerful, interactive medium that delivers a 'concrete' experience for readers [98, 99]. Since our end goal was for learners to be able to write out the correct recursive steps on a piece of paper and follow these steps to calculate the result, our design of the comics involved choosing a story and embedding drawings that could fade into the intermediate representation (i.e., visualization). That is, the story in the comic proceeded in the same order as the corresponding steps in the visualization, in order to facilitate mapping between the concrete and intermediate stage.

For instance, Fig. 3.1 (a) shows a comic about a manager trying to calculate his annual

bonus based on the years of service of his mentees, and his mentees' mentees, etc. In one of the panels, the comic shows a tree diagram illustrating the organizational hierarchy of the manager's mentees. This, we hypothesized, would help participants map the idea of recursive accumulation of bonuses along the organizational hierarchy to the branches of the 'double recursion' function (See Fig. 3.3a).

For the **intermediate representation**, we show a visualization of the drawing in the comic with the narrative details stripped away. For example, as users step through the tree diagram visualization, the phrase 'What is my bonus?' in Fig. 3.1(b) is displayed, referencing the panel in the comic showing the mentees in an organizational hierarchy, and illustrating the recursive process mentioned in the abstract representation.

For the **concrete-code representation**, we show the code where the function and variable names (e.g., what_is_ my_bonus, years_of_service) closely maps to the comic and visualization. At this stage, learners are asked to work with code without the support of the more concrete representations.

For the **abstract-code representation**, as shown in Fig. 3.2, the interface presents code that has the same logic as the concrete code, but with abstract notations and names, such as x and function, for function and variable names, and without any reference to the previous narrative.

```
def function (x):

if x == 0:

return 0

if x == 1:

return 1

else:

return function (x-1) + \

function (x-2)

function (6)
```

Figure 3.2: Double Recursion: Abstract code

Participants. We recruited 29 participants (20 F, 9 M, 0 Non-binary) from a local university with no prior experience in programming. Their interest levels in learning programming were 'Highly interested' (24.1%), 'Interested to a certain degree' (65.5%), and 'Not interested' (10.4%), and their perceived levels of difficulty for learning programming prior to the study were 'manageable' (55.2%) and 'difficult' (44.8%). Participants' age

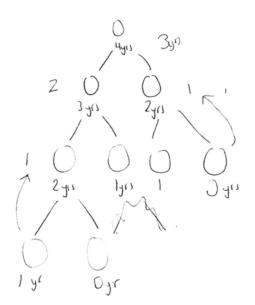
ranged from 18 to 49, with the average age being 21. In terms of background, 19 participants were in Science, Math, and Engineering; 4 in Health and Medicine; 5 in Arts and Humanities; and 1 without post-secondary education.

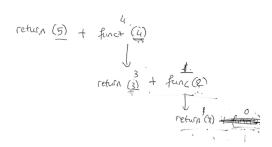
Procedure. Since participants had no experience in programming, they were asked to watch a tutorial video providing the definitions and code examples of recursion, function, and recursive function. Then, they used the system to study three examples of recursive functions (Fig. 3.1 is one example out of the three), and answer questions that asked them to predict the output of programs (variants of the presented examples) written in abstract code. That is, the questions presented code with the same logic as the code in the three examples, but with different inputs or conditions. During the task, participants were asked to think aloud while a researcher observed and recorded their thought process, as well as any interesting behavior or remarks. After the task, participants took part in an interview, where they were asked to elaborate on their responses to questions in the pre- and post-study questionnaires, and answer questions related to their experience with the interface. On average, each participant spent an hour on the study, and half an hour in the interview.

Findings

Utility of Multiple Representations. Through qualitative analysis, we observed benefits of multiple representations Ainsworth [3] summarized, to varying extent: computational offloading, re-representation, and graphical constraining discussed in Chapter 2. Several participants stated that reflecting on the comic or visualization helped reduce the cognitive load (computational offloading) and understand the code better when interpreting it. S14 explained that "since [he] already had the visual pictures in [his] mind, [understanding the code] was much easier to process." Several participants solved problems by mapping comic/visualization to code. S14 tried to find "the logic from the comic first and then apply it to the code" (re-representation). Like many other participants, S27 found the parts in the comic and visualization that demonstrate "branching in two directions," which helped him solve the double recursion problem. One of the comic strip examples (Fig. 1.7) showed someone using a recursive procedure to find out how many people are waiting in line in front of him. For this comic, S25 said that she "imagine[d] herself in the line" and found a solution to the question.

Some participants noted that they tried to map contents of the comic/visualization to specific lines of code in their head. For example, S26 said, "I tried to connect which part of code describes that situation [in the comic] like the 'else' [construct]." Likewise, S10 said that he used comic and visualization because the lines "return 1 and return 1 + function(x-1) are not clear" in the code, but clear in the comic and visualization (re-





(a) An example of a learner who applied the visual representations for solving problems

(b) An example of a learner who did not apply visual representations for solving problems

Figure 3.3: Participants' notes that demonstrate how some learners solved problems differently

representation). Seeing the same information in multiple representations also allowed participants to debug their understanding of the construct. As S15 and S23 said, the transition from comic to visualization helped "simplify" the comic and "organize their thought process" (graphical constraining). This, in turn, helped correct wrong assumptions they may have developed due to content in the comic that is ambiguous or counter-intuitive.

S4, who solved all the problems successfully and at the same time experienced a change in the way he solves problems using visualization, remarked that while he may have been able to solve the problems in the absence of visual aids, without them, he "would have lost a part of the perspective" that enabled him to "learn new ways to solve the problem faster." On the other hand, some participants (e.g., S29; Fig. 3.3b) who preferred to work with abstract code over visuals ended up struggling to solve the double recursion problems, because they did not see how the visual representations depicted the recursive process.

Self-Efficacy. Many participants also felt that the gradual transition from the concrete to the abstract "helped ease in and make [learning programming and recursion] more interesting" (S20). S25 and S27 corroborated this point by supporting that the concreteness

fading approach "demystified the way [programming] works," and added that if they had been introduced to the code first, it would have been "more intimidating." S15 remarked that while his "initial thought was that [programming involves] very abstract thinking [such] that it'll be hard to grasp," this learning experience assured him that learning programming is not as "bad." At the end of the interview, he asked where he can learn programming in the same manner, saying that he was now more interested in learning programming.

Overall, we found that coding strip allowed participants to connect programming concepts to their experiences, boosting their confidence and lowering perceived difficulty of learning programming. Surprisingly, even though she had never learned programming before, after the study, S19 was speaking as if she now has full grasp of what programming is all about, saying: "Programming is not just on a computer. You can apply it to many situations. [Telling] a computer to do something... is like your mom telling you to do chores. It's the same thing." And as S26 said that while she had "a problem with understanding the code first," the gradual scaffolding (from comic to code) gave her "interest" and made her say, "maybe I can approach it this way with coding." Similarly, after using our interface, S10 pointed out with confidence that recursion is "not hard" if one uses the waiting-in-line comic (one of the examples, Fig. 1.7 in Chapter 1).

Conclusion

These preliminary findings demonstrating various benefits of coding strips provided confidence in its potential. At the same time, however, the formative study helped us realize two problems that need to be addressed in order to scale and facilitate its use: how do we design coding strips? And in what other ways can it be used to support teaching and learning of programming concepts? In the next chapter, we present our design study conducted to answer these questions.

Chapter 4

Design Process & Tools for Coding Strip

The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.

Edsger W. Dijkstra [44]

In the previous chapter, I described our formative study. The study helped us realize two problems that need to be addressed in order to scale and facilitate its use: how do we design coding strips? And in what other ways can it be used to support teaching and learning of programming concepts? To answer these questions, I formulated the design process and tools to support the creation of coding strips. Then I conducted two design workshops, one with students and the other with teachers, to evaluate our design process and tools, as well as to understand how students and teachers perceive and see it being used to support learning and teaching of programming. In this chapter, I describe this design study and the results.

At the end of the chapter, we reflect on the design space for coding strips and design considerations gleaned from the generated coding strips and design patterns identified for this design study.

4.1 Design Process & Tools

This section describes how we developed the design process and tools for guiding learners or teachers to create comic strips that illustrate programming concepts.

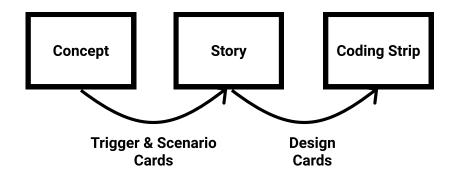


Figure 4.1: Three high-level stages in the design process, with trigger & scenario cards supporting transition from concept to story and design cards from story to coding strip.

4.1.1 Design Process

To formulate the design process, we first identified the main concepts taught in introductory computer science courses in high school and university (Table 4.1) as outlined in curriculum guidelines [108, 51]. Then, the authors followed the common practice [12], designing in parallel at least two comic strips for concepts in each category in Table 4.1 and until all the concepts were used. Through this experience, we identified three high-level stages in our own design process, as shown in Fig. 4.1—concept formulation, where we distill the core ideas behind a given concept, story development, where stories are invented to illustrate these core ideas, and finally comic illustration, where the stories are sketched out in the form of a comic strip.

From this, we created a design board (Fig. 4.2), an empty canvas with 5 sections corresponding to the steps of the ideation process: (1) trigger: reflect on the important properties of a programming concept, (2) scenario: generate many stories to illustrate the concept, (3) story & code: select one of the stories and write the corresponding code, (4) design: select an appropriate set of design patterns for the comic strip, (5) draw: produce the actual coding strip.

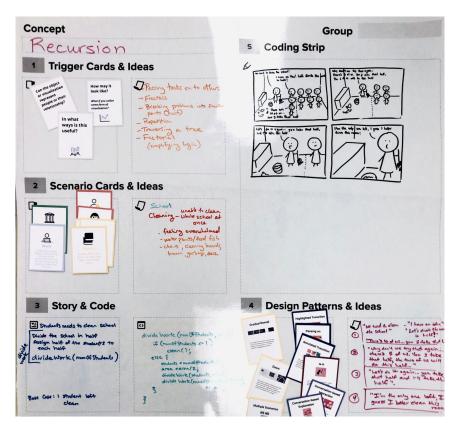


Figure 4.2: Design board with coding strip on recursion made by a teacher group at our workshop

4.1.2 Design Tools

To help designers brainstorm effectively, we created a set of *ideation cards* and introduced them into certain stages of the design process.

We introduced 16 trigger cards (Fig. 4.3a) which pose questions to designers to help them understand the properties of a concept and brainstorm ways to explain it. For example, for any given concept, one can reflect on 'what it is useful for,' 'in which context it is used,' its 'particular strength or weakness,' or 'how it compares with a similar concept' (e.g. linear vs. binary search). Trigger questions were generated by the authors writing down potential explanations for each programming concept in Table 4.1, clustering similar explanations, and then formulating questions that would trigger such explanation. For instance, for the concept of *variable*, the explanations include: 'it is used for storing data' and 'it is like a box that you can put something into'; for conditional, the explanation is 'it is used for decision making.' These explanations were clustered under the theme of 'purpose,' giving us the trigger question 'What is its purpose?'

For brainstorming stories/scenarios, we created 19 scenario cards (Fig. 4.3b) that ask designers to think about the four essential elements of story: setting, character, action, and conflict. *Setting* cards provide ideas for when and where the story takes place; *Character* cards for the protagonist or antagonist in the story; *Action* cards for actions undertaken by characters or objects; and *Conflict* cards for conflicts that drive the plot of the story.



Figure 4.3: Examples of ideation cards

Finally, for creating the comic strip itself, we introduced 30 comic **design cards** (Fig. 4.3c), based on design patterns seen in data comics [15] and comic strips, under five categories: sequence, selection, repetition, explanation, and presentation. *Sequence* patterns help highlight the transition from one step to another in the context of program execution. *Selection* patterns provide ways to introduce conditional and different outcomes. *Repetition* patterns help illustrate conditional and counted repetitions (i.e., while and for loop, respectively). *Explanation* patterns showcase direct and indirect ways to explain a concept. *Presentation* patterns (e.g., Fig. 4.3c) cover various presentation techniques that can make reading more engaging and the delivery of intended message more effective.

4.2 Design Workshops

We conducted two three-hour workshop studies: one (W1) with a group of undergraduate and graduate students, and the other (W2) with high school computer science teachers, to ensure a balance in perspective and feedback.

4.2.1 Participants

For W1, we recruited 13 university students (5M, 8F; age: 17-29, mean 21), with basic knowledge in programming concepts and from a variety of programs (4 Environment, 3 Science, 1 Computer Science, 1 Engineering, 3 Mathematics, 1 Arts). Most mentioned that they have no experience (8) or very little experience (3) teaching computer science (e.g., programming), and similarly in terms of teaching experience in other areas. Participants were split in terms of their level of confidence in drawing (4 Not confident; 5 Somewhat confident; 4 Confident), and confidence in designing comics for teaching coding concepts (5 Not confident; 7 Somewhat confident; 1 Confident). Participants were recruited via posters and SONA (a research participant recruitment platform), and provided \$60 for participating in the workshop.

For W2, we had 6 participants (2M, 4F). Among these participants, only three teachers (age: all 45) answered the pre-study survey questions. The three teachers specified that they had 3-5 years, 5+ years, 1-3 years of teaching experience, respectively. As for teaching experience in computer science (e.g., programming), one teacher had no experience, while two teachers had 5+ years and 3-5 years of experience. The three teachers rated their own teaching ability as 4 out of 5. All three participants were not confident about their ability to design comics for teaching coding concepts. W2 was conducted at a computer science educator conference hosted by the university; as such, participants were not paid.

Constructs	Data Structures	Algorithms	Problem Solving Techniques
Variable	Array	Selection Sort	Greedy
Boolean	Linked List	Insertion Sort	Divide & Conquer
Condition	Queue	Merge Sort	Recursion
Counted Loop	Stack	Bubble Sort	
Conditional Loop	Tree	Linear Search	
Function	Graph	Binary Search	
	Dictionary		

Table 4.1: Concepts used in our study. Bolded are the concepts that workshop participants chose for their coding strips.

4.2.2 Procedure

Both workshops consisted of two sessions (90 minutes each), with a 15-min break in between. After the consent forms and pre-study questionnaire (10 min), we introduced the idea of coding strip using an example, shown in Fig. 4.4 (10 min).

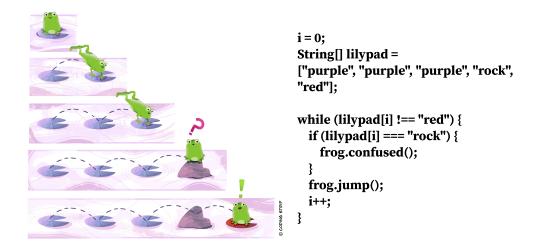


Figure 4.4: Example used to introduce the idea of coding strip during the workshops. Participants were asked to guess the underlying concepts/code from the comic strip before seeing its code, to show how comics can be used to introduce concepts and start a discussion.

Then we engaged in two warm-up activities (10 min each), which involved (1) several iterations of sketching simple stick figures and (2) translating story to code. After the warm-up, each group of two to four participants selected a programming construct from a list (Table 4.1), and proceeded to design a coding strip about the concept using the design board and ideation cards. In the second design session (50 min), groups were free to choose any programming concepts, including problem solving ones. The workshop ended with a discussion (35 min) and post-study survey comprised of short answer, multiple-choice, and five-point Likert scale questions (5 min).

For both workshops, we had two artists assist participants with sketching. They did not engage in the design activity but intervened when asked to help. We video-recorded both workshops with the consent of participants for accurate transcription of their responses and analysis of their interaction. The authors also made observations throughout the workshops.

4.3 Results

This section presents the analysis of the effectiveness of our design process and tools, and reports how students and teachers perceive and see coding strips being used to support learning and teaching. To ensure anonymity, we refer to our student participants from W1 as S113, and teacher participants from W2 as T16.

4.3.1 Effectiveness of Ideation Cards

Ideation cards were central to facilitating groups in generating coding strips. Thus, to understand their effectiveness, we examined (1) how many ideas each card was able to help generate, (2) how they were used, and (3) what ideas were generated from them.

Trigger Cards

They helped generate more than one idea for each card (# of cards: M=4.2, SD=1.7; # of ideas: M=6.7, SD=2.9). Across both workshops, the most frequently used trigger cards were 'How would you explain this to a five-year-old?' and 'When should you use this?' The 'Can you act it out?' card was never used possibly because this question asks a person to embody a concept, and such an approach is used mainly when teaching data structure and algorithm concepts (e.g., sorting). The 'How would you explain this to a five-year-old?' card was used by groups to generate stories containing objects and context familiar to children. For instance, a group working on the concept boolean used candy in their story, while another group working on conditional loop came up with an idea of asking a child to "check series of toys and report whether the color is red."

Scenario Cards

Each scenario card also generated more than one idea on average (# of cards: M=3.6, SD=1.7; # of ideas: M=5, SD=1.7). The groups usually had five story ideas to choose from, suggesting that scenario cards were quite effective at supporting ideation. Our analysis also reveals that the groups had different preferences on how to use scenario cards. While participants were encouraged to generate as many story ideas as possible during the ideation stages, only half of the groups followed this advice while the other half used scenarios cards (i.e., *Setting, Character, Action, Conflict*) to add details to the story. For instance, one group generated "playing sport" as an idea for Context, "soccer ball" for Object, "pass, shoot, defend" for Action, "between teammates & competitors" and "unable to

decide what action to take" for Conflict. The group used these to form a story about when to pass, shoot, and defend in the soccer field, in order to illustrate the concept conditional. The most frequently used scenario cards were Unable To and Institution. With the Unable To card, a student group working on recursion came up with the conflict: "unable to graduate due to a shortage of completed terms." One teacher group working on conditional used the Unable To card to generate the conflict idea "unable to remember," then used the Work card to come up with the action "deliver," and subsequently produced scenario: "Delivery man is unable to remember the house number he is delivering food to, so he just stands outside in the rain with his soggy pizza."

Design Cards

Several design cards (M=5.2, SD=2.4) were used each time to support design. For design ideas, since it required the participants to sketch, which they had varying levels of confidence on, we saw different results. Some put down sketch ideas, as shown in Fig. 4.6; some wrote script (e.g., dialogue) for each panel, as shown in Fig. 4.2; some did not sketch and went straight into drawing the actual coding strip. The most commonly used design cards were What-If (*Selection*), Multiple Scenarios (*Selection*), and Assign (*Sequence*). Fig. 4.6 shows the comic produced by one group working on the concept boolean; they used the Comparison, Multiple Scenarios and Before/After cards to generate a story comparing two possible scenarios for a fictional character Bob Lean (a play on word boolean): "Bob Lean is at home. It is 2:02 am, and he's playing games instead of sleeping. Next day, he can't go to class because he's sleep-deprived. Bob Lean is at home. It's 9:30 pm. Bob is well asleep. The next day he wakes up and goes to class/study."

4.3.2 Generated Stories & Comics

Participants, in general, were able to generate interesting coding strips using ideation cards. For example, one student group working on the concept conditional used scenario cards—such as Food, Unable To, Planets—to generate different stories and corresponding code, such as "is this pizza? yes/no," "if sleep == false: cannot study, else: study" and "if pluto != planet specs; return false." As another example, one teacher group working on recursion used 3 trigger cards: 'Can the object or visualization represent people or their relationship?', 'How may it look like?', and 'In what ways is this useful?' From these triggers, they generated ideas about the properties of recursion, such as "passing tasks on to others," "breaking problems into smaller points (half)," "repetition," and "traversing a tree." Then, they selected 7 scenario cards, and used the Unable To card to generate the



Figure 4.5: Design workshops with groups of students (left) and high school computer science teachers (middle & right).

"unable to clean" conflict, the Institution card to select "school" as the setting of their story, and produced the scenario "unable to clean the whole school at once." After writing down the story and its code, they selected 9 design cards and without sketching them out, wrote a script for each panel and asked one of the artists to draw the coding strip for them. The final product is shown in Fig. 4.2.

4.3.3 Ease of Design Process

Contrary to our initial skepticism, participants did not require much help in producing the comics. Every student group in W1 designed their coding strip without help from the artists; only once did one group request an artist to help them sketch a minor part of their design. On the contrary, most teacher groups asked the artists to help sketch their coding strips. However, their request for help seems to stem from the desire for a more polished final product. In general, participants found the design process and tools useful (M=4.1/5, SD=0.8 for W1, and M=4.2/5, SD=0.8 for W2), the instructions relatively easy to follow (M=3.8/5, SD=0.5 for W1 and M=3.6/5, SD=0.9 for W2), and the process of designing comics very engaging (M=4.2/5, SD=1 for W1, M=4.3/5, SD=1 for W2).

Their confidence in their own ability to produce coding strips also improved. Eleven students indicated that they feel more confident after this design session, with 2 participants, who were confident and somewhat confident prior to the session, remaining the same

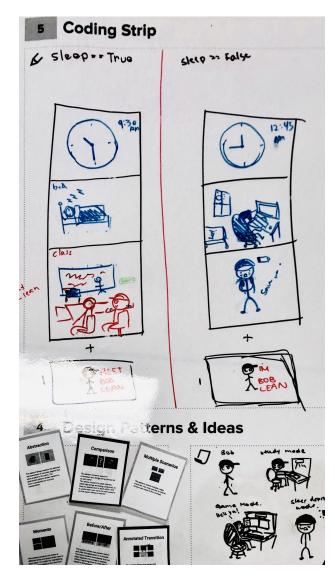


Figure 4.6: Design ideas generated by design patterns and the resulting coding strip on boolean by a student group

as before. At W2, 5 teachers said that they feel more confident after the session, with 1 participant who was not confident feeling the same as before.

4.3.4 Perceived Utility for Teaching and Learning

In the post-study survey, we asked—"How useful do you think comics is for teaching/learning compared to before the design session?"—to assess whether our design process changed their view of coding strips. Eight student participants said that coding strip seems more useful after the design session, with 5 students, who rated the usefulness highly (M=4, SD=1) before the session, feeling the same as before. At W2, 4 teachers stated that it seems more useful, with 2 participants perceiving usefulness the same as before. A similar trend was observed for the perceived utility of coding strip for learning.

When probed about the reasons for their desires to learn with coding strip, participants said that it is "fun," "engaging," "easier to understand as it relates to everyday life," and "offers visual representation." One student who said "no" explained that he thinks "it makes concepts harder to understand." A teacher participant who said "not sure" explained that she is not familiar yet with the design process; she may have misunderstood our question as she was enthusiastic about coding strip: she asked us to come to her school and run the same workshop with her students. Two other teachers also expressed a desire to access and use our design board and ideation cards in their classrooms. As for teaching with coding strip, teachers (5 'yes'; 1 'not sure') liked the idea, while students (6 'yes'; 7 'not sure') were divided.

4.4 Discussion

4.4.1 Extensions and Limitations

Several suggestions emerged from the workshops. One recommendation is to include a "peer feedback" section on the design board to allow groups to interact with each other by leaving feedback on each other's work. Another suggestion is to develop "a unified comic language" which would allow people to see "similar design everywhere." Two participants (S1, T1) voiced the desire to have a system with pre-defined templates that enable users to create coding strips without having to manually draw them. One participant (S12) explained that in order to "help maximize learning," the design activity should help students "[focus more] on creating the ideas for the coding strip [than] on drawing." One participant (T5) suggested labeling ideation cards with level of difficulty to ease the identification of relevant cards.

Participants also noted several limitations in coding strip. One participant was skeptical of the ability of coding strip to portray hard concepts: As S5 said, "I believe that this

should only be used for beginner computer science concepts, as it would be more difficult to express more advanced and abstract concepts through this medium." This is probably because he did not get to see, or design, coding strips for advanced concepts, such as merge sort, like the authors did while creating coding strips for every concept in Table 4.1. Some participants (S3, S5) expressed uncertainty about engaging novice learners in the design activity, as the activity might rely too much on the novice learners' "capability to make the correct representation of a concept." Two student participants noted (S3, S4) that coding strips might be more effective for students with a certain learning style, e.g., those who "learn better with visual representations." While this concern is valid, the variety of use case scenarios, such as using the design activity to teach collaborative skills, suggests that there are ways for coding strip to benefit various types of students. One student participant also mentioned that in order for students to participate in the design activity, they must have an adequate level of understanding of the concepts. This is aligned with another teacher's feedback that our design process would work well for her 12th grade students, but not for the 11th grade students as they have not learned all the concepts yet.

4.4.2 Extensibility of Ideation Cards and Design Process

Golembewski et al. and Chung et al. noted that another use of ideation cards is defining the design space [62, 30]. While this was not the focus of our study, our design cards can certainly help inform the design space for programming concepts—like how the Bach et al. [15] used design patterns to chart the design space for data comics. Some of the design cards used in this study are shown below (Fig. 4.7). (All the ideation cards and design board can be found in Appendix B.) Additionally, our design process in which we use trigger and scenario cards to generate stories for a concept may be useful for any domain that wants to use stories as analogies to explain any of its abstract, complicated concepts. Several teachers at the workshop noted the broad applicability of trigger and scenario cards, saying that they would like to use them when teaching other subjects, such as math and science, for brainstorming ways to explain complicated concepts.

4.4.3 Design Considerations in Coding Strip

Here, we reflect on various design considerations for coding strips. We first review the design patterns (cards) used in the workshop and then the interacting factors (see Table 4.2 that influence the coding strip design. We describe any meaningful observations for further exploration in the future. Note that the design patterns or layout below are not new discoveries; they have been explored in other studies on comics [34, 15] and in comics (e.g.,

Code	Execution	Line	Structure
External Design (Panel Layout)	1-to-1 1-to-many many-to-1	1-to-1 1-to-many many-to-1	Grid Grouping Network Flowchart
Internal Design (Panel Content)	Scene / Setting / Action		

Table 4.2: Breakdown of interacting factors that influence the design of coding strips and resulting design variations.

PhD Comics [26])—they inspired some of our design patterns. The discussion on design patterns is intended to shed light on design considerations specific to coding strips.

Design Patterns & Choices. As mentioned, our design cards had five categories: sequence, selection, repetition, explanation, and presentation. Fig. 4.7 shows an example card for each category. One thing to note here is that two categories—presentation and explanation—represent general design patterns, whereas the remaining categories—repetition, sequence, and selection—are specific to programming and code constructs. For instance, design patterns under *Repetition* would be used if the code snippet contained loop; those under *Selection* would be used if the code snippet contained code related to control flow such as conditional statement (e.g., if x == 0:). This suggests that certain design patterns would be more relevant and that we can expect to see or use certain patterns more depending on the code snippet. General design patterns (i.e., those under *Presentation* and *Explanation*) is less predictable, as these can depend on stories and individuals' aesthetic, presentation preferences or needs (e.g., what they desire or need to teach).

Related point that plays a role in this dynamic is the programming language or paradigm. In the workshop, participants were free to select their programming language when writing code. As a result, we saw a range of choices, from pseudocode to Python and Java. We did not see functional language such as Scala. But our preliminary analysis of produced coding strips suggests that which programming paradigm (e.g., procedural, object-oriented, functional) is used can influence design choices. For instance, comics for functional paradigms may leverage arrows and nonlinear layout more than linear layout as many function calls can be recursive. On the other hand, design choices for object-oriented programming comics may focus more on grouping layout (see Fig. 4.11) as it needs to highlight different objects present in the code.



Figure 4.7: Design card types and their examples

Code (Execution) \leftrightarrow **Comic (Panel).** As shown in Fig. 4.9, we find that there can be **1-to-1**, **1-to-many**, and **many-to-1** patterns in terms of how panels are mapped to execution steps. In **1-to-1** mapping, the most frequently used pattern by the participants, each panel describes an action or values (state) at each execution step (e.g., Fig. 4.9a, 4.4). Examples of comics that used **1-to-many** mapping include flowcharts that represent multiple execution steps within a loop, or a set of panels that illustrate only the beginning, middle, and end of a loop. Finally, we observed the use of **many-to-1** mapping in a story about counted loop where a child colors every object in the household. As shown in Fig. 4.9c, the group allocated three panels for a single step, e.g., each panel showing more areas of the sofa being gradually covered in crayon marks. In general, **1-to-1** mapping may benefit novice learners' procedural understanding by spelling out the action step by step; **1-to-many** mapping may help learners better grasp high-level concepts by creating



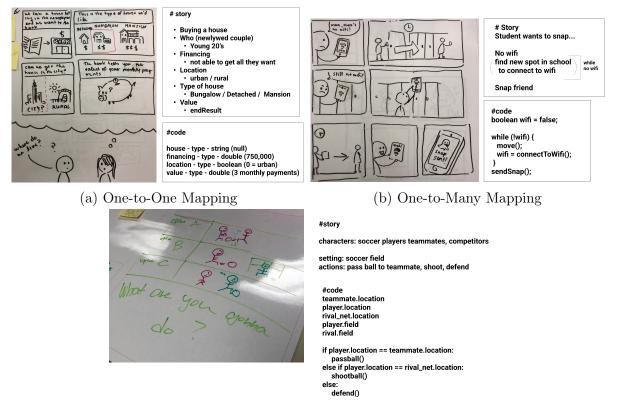
(a) One-to-One Pattern (b) One-to-Many Pattern (c) Many-to-One Pattern

Figure 4.8: Three Code (Execution) \leftrightarrow Comic (Panel) mapping patterns

meaningful, comprehensible abstractions for execution steps; finally, **many-to-1** mapping may improve conceptual understanding by providing more details about a concept's specific properties by leveraging multiple panels to explain a single execution step. These observations can be related back to the type of designs to use depending on the point of emphasis.

Code (Line) \leftrightarrow **Comic (Panel).** Another interesting observation is the relationship between panel and code. This relationship is not equivalent to the relationship between panel and execution steps. To demonstrate, consider for loop that repeats 100 times. While it executes 100 times, it can be fewer than 4 lines of code. The number of execution steps and number of lines in the code are not necessarily the same. In some cases, the numbers can match if the code does not have loop and uses simple constructs that execute only once (i.e., 1 line \leftrightarrow 1 execution step). But as the example shows, this is not always the case and thus panel-to-code mapping can be another consideration when designing coding strips.

Code (Structure) \leftrightarrow Comic (Panel Layout). Code structure refers to the organization that the lines of code has according to their purposes. Fig. 4.10, for instance, shows two simple structures: one with setting and action, the other with just the action. If the code contains variable assignments in the beginning, for instance, we may regard it as a block that sets the story setting and then assign control flow related code (e.g., while, if, for) as action block. But we do not need to mandate that certain constructs only serve specific roles. We can apply the same principle that Cohn applied for visual narrative

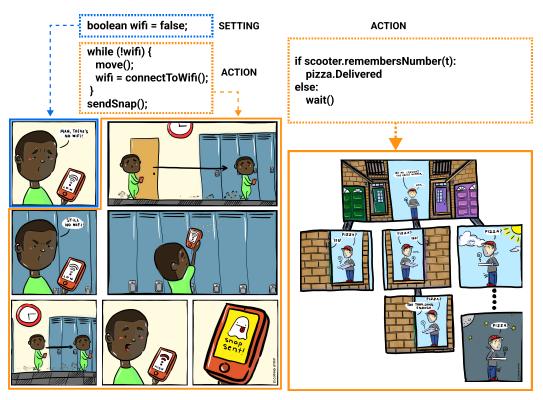


(c) Many-to-One Mapping

Figure 4.9: Three Code (Line) \leftrightarrow Comic (Panel) mapping patterns

grammar (cf. Chapter 2). That is, what role to assign the code block can depend not on the code itself but on the role it serves in relation to the code next to it or more precisely its role in the entire narrative. What panel layout to use can be decided once the story has been formed.

Panel layout is tightly coupled to the code structure, because of its role in effectively guiding users' gaze which can follow the computational procedure present in the code. We observed four common panel layouts: Grid, Grouping, Network, Flowchart. Grid layout is a simple pattern that directs users to read in a single direction horizontally or vertically, e.g., left-to-right, right-to-left, or top-to-bottom. This layout uses panels of similar width and height without horizontal or vertical staggering (cf. Chapter 2), which signals that each panel is of the same importance (cf. Fig. 4.9a). Thus it can be useful for listing the actions in the iterations of the loop. This layout can also be used to chunk related panels—for example, in the same column to make it clear the role of the panels



(a) Structure consisting of setting and ac- (b) Structure consisting only of action tion

Figure 4.10: Examples of code structures and the layout

(cf. Fig. 4.6). Figures 4.9a, 4.9c represent comics with grid layout where panels are read left-to-right. **Grouping** layout helps partition comic panels into relevant groups. (Cohn uses the term *staggering* but we will use *grouping* as it aligns better with the semantics assigned here.) For instance, Fig. 4.11 shows panels that are grouped into two; the first panel introduces the story context, the following three panels in each row represent one iteration in a loop. How about conditional loop where the number of iterations are not pre-defined? How can we illustrate that? For this, two nonlinear layouts—network and flowchart layouts—can be used. **Network layout** is a layout that has a cyclic flow, as shown by Fig. 4.12. In the case of **Flowchart** layout, panels were used as a rectangle that represents 'process' in flowchart, with arrows representing the 'flowline.'



Figure 4.11: Grouping layout. The three panels on the right form their own group both semantically and structurally.

Code (Execution/Line/Structure) \leftrightarrow Comic (Content). We identified three content types: scene, setting, and action. The action contents are those that visualize the execution steps and code expressions, whereas the scene and setting contents are those that provide context for the story. For instance, the first panel on the left in Fig. 4.11 illustrates a story setting (context) not tied to code expression or execution, while the three panels on the right illustrate looping execution steps (i.e., Day 1 ... Day 50 ... Day 100) and the code expression (e.g., day 1). (A similar pattern can be observed in Fig. 4.8c: the first panel at the top opens up by introducing the context of the story and then proceeds to illustrating the execution steps.) But here, there can be several questions. What if the first few lines of the code concern variable assignments and are followed by loop? Story-wise, if the variable assignments represent the context (setting) of the story and the loop is where the action (i.e., climax in the narrative arc) occurs, should we regard the part of comics illustrating variable assignment code (e.g., x = 5) as setting? Or should it be regarded as action content because it illustrates code executions and expressions? Or should we restrict action content to those that involve only control flow (e.g., loop and conditionals) so that we can regard variable assignments as setting? What labeling scheme is appropriate can be up for debate and which to use may depend on the context. Since the scheme is not the goal of this study, we leave this as future work.

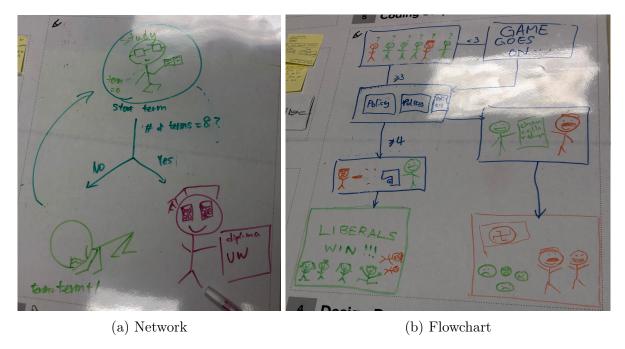


Figure 4.12: Nonlinear Layout

4.5 Conclusion

In this chapter, we presented a design workshop study and design considerations for coding strips. Although not exhaustive, the set of design considerations demonstrates various ways coding strips can be designed to illustrate abstract concepts, languages, and procedures in an effective manner. We explored how to design coding strip, comic strip for teaching and learning programming concepts. Findings from our two design workshops show that both students and teachers are enthusiastic about the prospect of applying coding strip to the current pedagogical practices, and our design process and ideation cards are able to support the design process, even without the support of artists. In addition to the design process and supporting tools—which are available online ¹, our work contributes new understandings concerning what design considerations (e.g., $code_{execution} \leftrightarrow comic_{layout}$ mapping) exist and how the visual language of comics can be used to support the teaching and learning in programming.

¹https://codingstrip.github.io/

Chapter 5

Authoring Tool for Coding Strip

Designing a computational language is to create a bridge between two domains: the *abstract* world of what is possible to do, and the '*mental*' world of what people understand and are interested in doing. The challenge is [...] to give people a way to describe these [abstractions].

Stephen Wolfram [158, 123]

In the previous chapter, a design study and tools for supporting the design of coding strips were presented. While they worked well, they are not a *scalable* solution: the manual authoring can be time-consuming and laborious. Some participants asked for a digital authoring tool that can make the process easier and less burdensome. This begs the question of if we can automate parts of the process to ease the process of creating coding strips. Building on the fact that the process requires mapping (1) code to story and (2) then story to code, can we computationally support the mapping to make the authoring process easy and quick? In this chapter, I introduce CodeToon ¹, an authoring tool that explored this question. I first discuss how this process slightly differs from the design process in Chapter 4. Then I present its user interface, workflow, design process and goals used to iteratively develop the system, and two-part evaluation studies.

¹https://codetoon-research.github.io/

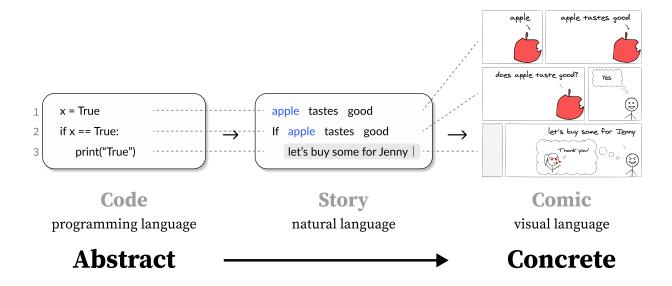


Figure 5.1: CodeToon helps users create stories and comics from code. It uses 1-to-1 mapping to make the connections clear across code, story, and comic. For instance, as indicated by the dotted line, line 1 (code) maps to line 1 (story) and to row 1 (comic).

5.1 CodeToon: Computational Approach

When the coding strip idea was first conceptualized, one of the ideas that fueled this research was building a system that can take code as input and output comics—i.e., visualize program in terms of comics—so that students can learn to code using comics. Just like program visualization tools that had been developed—but with comics as opposed to with diagrams [66] and animated characters [120].

Because comic is manually created in the ideation method (Chapter 4), its design process began with programming concept instead of with code. This was because the scenario used to formalize the design process was one where teachers are teaching (or students are learning) a certain programming *concept* and are tasked with creating comics to explain/illustrate it. In other words, they begin with *concept*, not with *code*. Because there are many things to say about a concept and ways to explain it (e.g., what it is useful for, when you need this), the design process in the ideation method began with brainstorming what we can say about the concept. As we search for a way to automatically generate comics from code, the starting point—as is commonly the case with program visualization tools—is *code*, not *concept*. A piece of code can be associated with a certain concept, and as will be shown below, CodeToon does provide a list of code snippets and associate them with programming concepts. However, the reason for pointing this out is to highlight the differences in the user interaction between the two methods. CodeToon users do not need to actively brainstorm ideas about the concept as was necessary for the ideation-based method. In the next section, we introduce CodeToon and its interface.

5.1.1 Design Process and Design Goals

To develop CodeToon, we first conducted a pilot study with 12 participants, where we investigated whether our preliminary designs help users be creative and collected suggestions from users to improve the authoring experience. Over the course of the pilot sessions, we improved, added, and tested new features and workflows of CodeToon (see Sections 5.1.2 and 5.2 for details) until we observed no major changes are needed to enable creative authoring experience.

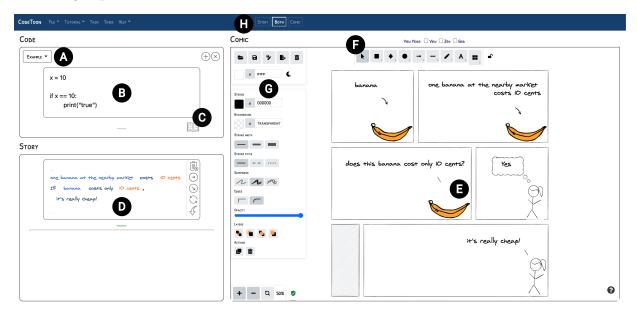


Figure 5.2: System interface: (A) drop-down allows users to check potential code examples for basic programming concepts, (B) code container, (C) button that generates story template from code in code container, (D) story template, (E) drawing canvas for comics, (F) tool palette, (G) style palette, and (H) buttons for changing the interface layout (between code & story, current, or canvas-only layout).

To guide the development of CodeToon, we derived a set of design goals based on literature on multiple representational systems [3, 19, 142] and creativity support for comics [143], as well as the pilot study. The final design goals are as follows.

D1. Allow users to iterate on their code, story, and comic. From our pilot study, we found that the authoring process may not be linear. While creating comics, users can be inspired by their comic and form additional ideas to add to their story. While working on the story template, they could also think of a better story and desire to edit code (e.g., change value assigned to variable). Thus, the tool should make this interaction easy for its users.

D2. Augment, not constrain, users' creativity with story ideation and automatic comic generation. Our pilot study revealed that providing story ideas and comic templates can accelerate the authoring process. But it also showed that some users can already have some ideas on what they want to create and how to design their comics. Thus, the tool should not limit users to use only story ideas and comic template provided by the tool.

D3. Make mapping clear across code, story, and comic. Research suggests that making correspondence explicit and consistent is important when presenting multiple representations [142, 140]. Otherwise, they do more harm than good because it only confuses people. For us, this means that mapping between code, story, and comic should be clear. Previous research on coding strip also suggested that the mapping between code and comic needs to be clear for it to be effective and useful [140].

D4. Use simple, scalable visual vocabulary. Scalability relies on having a set of basic building blocks that can be combined to construct anything of varying complexity (see composition method in Section 2.3). The building blocks in visualizations are called visual vocabulary. To generate comics that can scale to any code input, establishing a set of simple, scalable visual vocabulary that can express any set of code is necessary.

As a whole, our design goals aimed to create a "low floor, high ceiling" system for generating coding strips. That is, a system that makes the process of creating coding strips simple, effortless, and easy, while providing "high ceiling" for creative exploration. In the next section, we describe the CodeToon's user interface and usage scenarios.

5.1.2 User Interface

CodeToon consists of three panels: code (Fig. 5.2B), story (Fig. 5.2D), and comic (Fig. 5.2E). At any point, a user can select any of the layout buttons (Fig. 5.2H) to change which panels are shown. The default layout ('Both') shows all three panels, the 'Story' layout only code and story panels, and the 'Comic' layout only the drawing canvas. The layout buttons can be used when a user needs more space to author code, story, or comic.

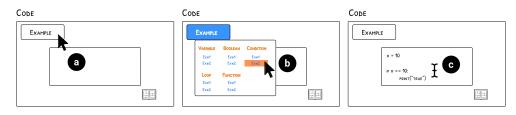


Figure 5.3: Users can add code by selecting code example (b) or by typing (c).

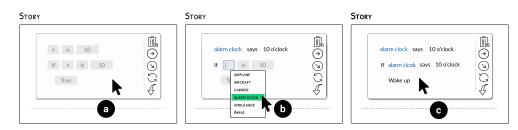


Figure 5.4: Users can add story (to story template) by selecting a list of ideas in dropdown (b) or by typing into input box.

The three panels represent stages in the design process for creating a coding strip [143]. Concretely, the basic workflow consists of users (1) adding code in the code panel, (2) generating a story template (from the code) and writing a story in the story panel, and (3) using the auto comic generation feature to instantly generate comic in the comic panel. Below, we describe each panel and how they facilitate this workflow.

Code. In the code panel (Fig. 5.2B), users can add any number of programs, each within a different 'code container.' As shown in Fig. 5.4, users can add the code into the container by using the code example repository (Fig. 5.2A) or manually typing into the code container. There are two buttons on the top right corner of the code panel for adding (\oplus) and deleting (\bigotimes) code containers. The ability to add additional code containers were added during the pilot phase to make it easy for users to iterate on their code, story, and comic (**D1**). After the user adds code, they can press a button (Fig. 5.2C, EE) to generate a story template (see Fig. 5.2D) from code.

Story. When a user generates a story template, it is added to the story panel, which is initially an empty panel. Fig. 5.2D shows what the user would see when a story template has been added. Story templates are linguistic representation of the code, with input boxes where users can add real-life equivalents for the code expressions. For instance, the code expression, 'x = 10' generates \times is 10 (see Fig. 5.4(a-b)) as its story template. Now, a user can add 'alarm clock' to \times , 'says' to \times , and '10 o'clock' to \times . As shown in Fig. 5.4(b), CodeToon provides a dropdown containing a list of metaphors to help users

brainstorm story ideas. The dropdown does not appear for every input box, however. At the time of testing, it appeared only on input boxes mapped to variables (e.g., 'x') and variable assignments ('='). A dropdown for the former showed a list of 345 categories (e.g., apple, car) and that of verbs synonymous with or semantically close or related with the semantics of '=' (e.g., 'assign,' 'has'). Although dropdown appears to help users with story ideation, they do not have to form their stories around these suggestions; they can type in any text (**D2**).

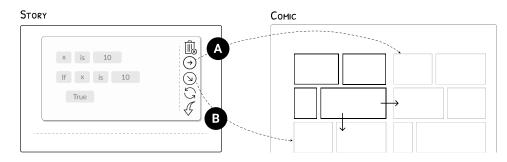


Figure 5.5: Users can auto generate comic by selecting either of the arrow buttons. If canvas already has some drawing elements, as in this case, they can add the comic to the right (A) or below (B) the existing elements. If canvas is empty. The comic is added to the center of the canvas.

Comic. As shown in Fig. 5.5, a user can instantly generate comic (Fig. 5.2E) by selecting any one of the two arrow icons below the trash can icon (\rightarrow) 阆 and (\mathbf{A})). The reason for two arrows is to allow users to expand on their existing comic by adding newly generated comics to the right or below the existing drawings. If users change the story, they can press the update icon \Im (below the arrow icons) to instantly update the content of the auto generated comic to reflect these changes, making iterative design of their story and comic frictionless (D1). While users can use the auto comic generation feature to instantly generate comics from the story template, they can also use the tool palette (Fig. 5.2F), style palette (Fig. 5.2G), and stencil library (Fig. 5.6) to manually create comic or edit/expand the auto generated comic. The view mode above the palette (Fig. 5.2F) provides three checkboxes for turning on/off different view modes: 'grid' makes grid appear in the canvas for precise alignment and measurement, 'zen' removes style palette, and 'view' removes both the tool and style palette and makes the canvas view-only.

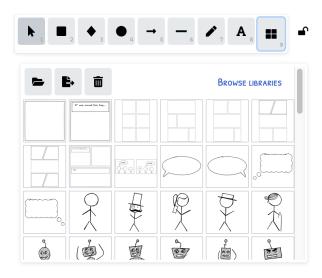


Figure 5.6: Users can add templates (e.g., comic panels, speech bubbles, and characters) to the canvas using the library.

5.1.3 Usage Scenarios

To further clarify the user interface and the workflow, we present here two usage scenarios to demonstrate how users may use CodeToon.

Teacher. Amanda is a high school teacher. She is teaching a programming class to 10th-grade students who are learning programming for the first time. She wants her students to discover that programming is not just about memorizing rules, syntax, and expressions. She wants her students to realize that computational ideas can be found in our every day life as well. When the class starts, she explains this as her goal for her students. To show how they can think of computing in terms of real-life objects and situations, she opens up CodeToon and shows a sequence of code-story-comic example shown in Fig. 5.1. She explains that while code expressions may appear scary for now, they can think of code as being no different from words we use to communicate. To direct their attention to the logic (i.e., *if-then* structure) and not the content, she switches the object in the story from 'apple' to 'banana' and updates the comic (by pressing the update icon \mathbb{G}). To make the stories more engaging and help students connect with them, she also adds contextual details to the expression 'banana,' editing it to 'one banana at the nearby market,' generating a sentence 'one banana at the nearby market costs 10 cents' (cf. Fig. 5.2). She invites her students to suggest a story and produces comics for them on the fly. Students suggest diverse stories based on their personal experience and learn that

programming is not as difficult as they thought it would be.

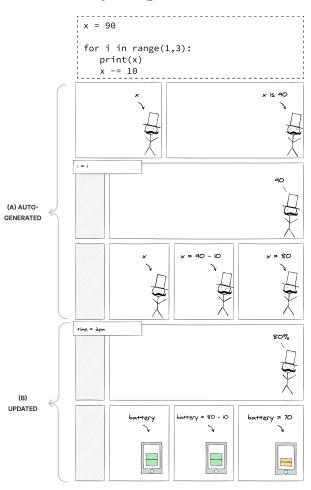


Figure 5.7: A simple loop code and (A) auto generated comic. The bottom two rows represent (B) comic updated with different images (phone) and text (e.g., 'battery' for 'x'). Like in this example, CodeToon can update specific rows, giving users fine-grained control over their stories.

Student. Ken is a graduate student working as a teaching assistant in the introductory CS course. While overseeing a lab, one student asks for help, saying that her code is not working the way she wants it to. He takes a look at the code and feels that a piece of code involving loop may be the culprit. He tells the student but realizes that she does not have a good grip on the concept of loop and how to debug the code. He opens up CodeToon and adds the code in her assignment and generates the comics to show her how loop works. Fig. 5.7 shows an example of a code with loop and its corresponding comic. Ken explains

to the student how the first row of the comic maps to the first line of the code (x = 5) and that the next two rows represents the two lines in the loop; he notes that the program then moves to the next iteration and points to i = 1 as an indicator of which iteration the program is running at. To help the student connect the concept to a real-life situation, Ken adds a story to the story template: he replaces 'x' with 'my phone's battery,' '=' with 'is at,' '90' with '90 percent,' and 'i' with 'time' and presses the update icon to update the comic, which shows a comic with a story that shows the phone having a battery initially at 90% and decreasing by 10% every hour, as shown in Fig. 5.7. Ken creates another code container and shows the student another loop example to help her master the concept. After the student is done receiving help from Ken, she asks Ken for the URL of CodeToon so that she can use it to review and assist her studies in the class.

5.2 Generating Comics from Code

CodeToon supports the generation of comic from code using two mechanisms, story ideation and automatic comic generation. Here, we describe these two mechanisms in detail.

5.2.1 Story Ideation

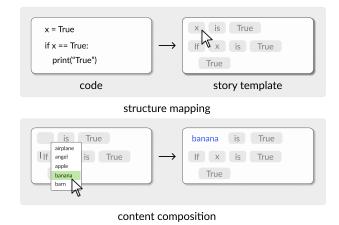


Figure 5.8: Our story ideation consists of generating a story template aligned with the code structure (structure mapping) and allowing users to fill the template with the help of metaphor suggestions (content composition).

CodeToon aims to make the generation of comics from code more efficient, and the first step in that process is rapid story ideation. To understand how we can turn code into a story, we first went through code expressions and turned them into a story. Table 5.1 shows some examples. We found that any code expression can be parameterized (cf. Section 2.1.2) and replaced with metaphors of corresponding form. For instance, variables (e.g., x) can be metaphors in noun (e.g., wallet) or phrases (e.g., message in my email); assignment operator (=) can be 'be verbs' (e.g., 'am,' 'is,' 'are') and 'transitive & intransitive verbs' (e.g., wallet 'has' 5 parking coins, my dog 'feels' sick); values can be contextualized (e.g., $5 \rightarrow 5$ o'clock, True/False \rightarrow on/off, 'hello'' \rightarrow "hello, John"); keywords (e.g., def) and (built-in/user-defined) functions (e.g., print()) can be replaced with semantically related verbs/phrases. For instance, print can be 'say.' Thus, in the story template, we (1) provided a list of metaphors they can choose from and (2) converted code expressions into text fields, to allow users to be creative with the story authoring (D2).

code	hybrid (code & story)	story
x = 5	$time = 5 \ wallet = 5 \ student = 5$	time is 5 o'clock wallet has 5 parking coins student received 5 dollars
x = True	$switch = on \ my_schedule = busy \ this = True$	switch is on my schedule is busy this is expensive
x = "hello"	message = "hello"	message reads, "hello"
<pre>print("Even")</pre>	<pre>print("it's even")</pre>	say, "It's even!"

Table 5.1: Examples of code and corresponding stories

5.2.2 Auto Comic Generation

Auto comic generation requires the conversion of different types of code expressions into a visual panel arrangement (e.g., panels, characters, speech bubbles). We approach this problem by defining in advance a specific design template for each code expression (e.g., variable assignment, loop). Specifically, we leveraged the theory of VNG [31], which suggests that each panel of the comic can be categorized into one of the five phases of a narrative: (1) *Establisher*, which sets up a scene; (2) *Initial*, which depicts the start of an action; (3) *Prolongation*, which shows moments between the start of an action and its peak; (4) *Peak*, which marks the point in the action when the tension reaches the peak; (5) *Release*, which shows moments after the action has ended. Below, we provide two examples of code expression template, and the thought process that went into designing these templates using VNG.

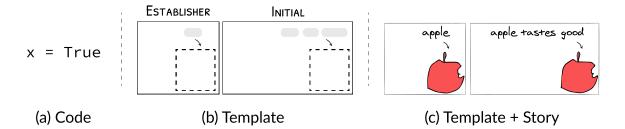


Figure 5.9: Variable assignment: code, template, example

Fig. 5.9 shows the variable assignment 'x = True' (Fig. 5.9(a)), its template (Fig. 5.9(b)), and the first row of the comic in Fig. 5.1 (Fig. 5.9(c)) that uses this template. Variable assignment can be defined by two operations: first, computers allocate a memory space for the variable x; then they assign value to the variable. The first operation is (semantically speaking) analogous to 'Establisher' in that it 'sets up a scene (for assigning value).' The second step can be regarded as 'Initial' as it 'initiates' the action of assigning value to this variable. Hence we use two panels, Establisher and Initial, as shown in Fig. 5.9. While it can also be valid to have just one panel (e.g., 'Initial' instead of 'Establisher + Initial'), another justification for the 'Establisher + Initial' combination is a cleaner design. Typical computer programs will contain multiple variable assignments, which means that the auto generated comic will have several templates like this, stacked on top of each other. Having two or more panels, aligned and stacked on top of each other, can make the final comic design more structured. This can also be more appropriate for teaching variable assignment at the memory level, by illustrating space allocation and value assignment separately.

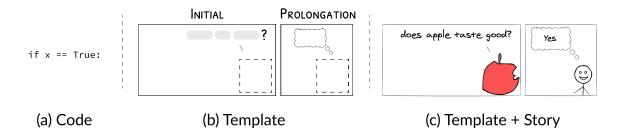


Figure 5.10: Conditional expression: code, template, example

As another example, Fig. 5.10 shows the conditional expression 'if x = True:' (Fig. 5.10(a)), its template (Fig. 5.10(b)), and the second row of the comic in Fig. 5.1 (Fig. 5.9(c)) that

uses this template. A similar thought process went into designing this template: the conditional expression first checks whether the statement is true or false; as this (checking) is an action, the first panel is an 'Initial' panel and has a placeholder for text that ends with a question mark to indicate the 'checking' action. The following panel is used to report (hence the speech bubble) whether the expression evaluates to true or false; this panel is a 'Prolongation' panel, as it sits between the 'Initial' and 'Peak' (code wrapped around the conditional expression that would run if the expression evaluates to true).

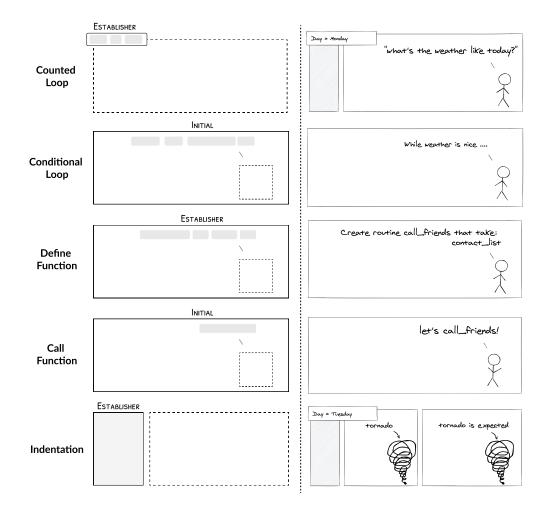


Figure 5.11: Other visual vocabularies (comic templates, left) and examples (right). The gray boxes are placeholders for text; the dotted squares for objects (e.g., stick figure); the dotted long rectangles for any visual vocabulary, e.g., variable assignment, loop, and so on.

There can be more than one way to define these templates for coding strips depending on the goal(s), programming language and paradigm. In our case, the goals were: ensuring scalability, conveying semantics and executions, and offering design variations (e.g., if possible, avoid using only a single panel within a row); also, our visual vocabulary was created using Python (code expressions, syntax, and conventions). Understanding various nuances required to construct visual vocabularies for various programming languages is not within the scope of this research; we leave that as future work. (Further implementation details are provided in Appendix.)

A key challenge in generating comics from code is maintaining a clear mapping between code, story, and comic (D3), so that learners can tell which line of code maps to which line of the story and which panel of the comic. We achieved this design goal (D3) by generating story template and comics that map 1-to-1 to lines of the code, as shown in Figures 5.1 and 5.8. In addition to 1-to-1 mapping, visual cues were also used to make the correspondence easily perceptible. For instance, if there was an indentation in the code, this was carried over to the story template. For comic, a gray empty panel was used to indicate indentation, as shown in Fig. 5.11. Note that, as shown in Fig. 5.7, when code has loop, 1-to-1 mapping cannot be maintained as comic needs to visualize the iterations.

5.3 Evaluation

To test whether CodeToon successfully supports the creative design process and generates quality comics from code, we conducted a two-part evaluation: a user study and a comic evaluation survey.

In the user study, we aimed to answer: (**RQ1**) Does CodeToon support the authoring of coding strip, in terms of story ideation and comic creation; (**RQ2**) Does CodeToon make the process of authoring coding strip more efficient; and (**RQ3**) What is the perceived utility and use cases of CodeToon for teaching and learning programming? Notice that RQ3 does not focus on how well CodeToon supports the authoring of coding strip but on the perceived pedagogical value of CodeToon. While the learning benefits of engaging in the process of creating coding strips as well as their use cases have been examined in prior studies [143, 140], prior works did not explore this with CodeToon. This motivated us to explore RQ3, as it can guide future work.

Further, based on the created comics from the user study, in our comic evaluation study, we aimed to investigate: (**RQ4**) Does CodeToon help generate high-quality comics, and how consistent is the quality?

5.3.1 Part 1: User Study

Our user study employed a between-subjects design with two conditions: Baseline (B) and CodeToon (C). Baseline was the same as CodeToon, but without the story ideation and auto comic generation features.

Participants

We recruited all 24 participants (12 for each condition) from a local R1 university's study participant recruitment platform. Participants were required to have (1) basic programming knowledge, (2) a mouse, and (3) Chrome browser on their device.

Most Baseline users (age: M=23.3, SD=2.5; gender: 8F, 4M, 0 Non-binary) had decent programming experience (6 Some Experience, 6 Much Experience), some experience with digital drawing tools (2 No Experience, 9 Some Experience, 1 Much Experience), and mostly positive perception towards the comics' usefulness as a tool for teaching and learning programming (1 Slightly, 1 Moderately, 6 Very, 4 Extremely Useful). Many were in the middle in terms of their confidence in drawing (4 Not Confident, 5 Somewhat Confident, 3 Confident) and creating comics for teaching programming concepts (3 Not Confident, 8 Somewhat Confident, 1 Confident). They had varied experience with teaching programming (6 No experience, 2 0 - 1 year, 4 1-3 years).

CodeToon users (age: M=27.3, SD=4.8; gender: 2F, 10M, 0 Non-binary) also had decent programming experience (9 Some Experience, 3 Much Experience), some experience with digital drawing tools (3 No Experience, 7 Some Experience, 2 Much Experience), and similar perception towards comics' usefulness (1 Slightly, 2 Moderately, 4 Very, 5 Extremely Useful). Many of them were not confident in drawing (7 Not confident, 4 Somewhat Confident, 1 Confident) and creating comics for teaching programming concepts (8 Not Confident, 3 Somewhat Confident, 1 Confident, 1 Confident). They had little to some experience teaching programming (3 No experience, 5 0-1 year, 2 1-3 years, 1 3-5 years, 1 5+ years).

Procedure

Baseline and CodeToon users followed the same study procedure. The study was conducted remotely via a video conferencing software. Participants first completed a pre-study survey. The survey included questions about their demographic information. Then, they went through tutorial videos that explained the interface and how to use the tool, after which participants conducted practice tasks that were replication tasks by following the videos. CodeToon users spent more time (at least 8 min) in the tutorial phase as they needed to learn and try story ideation and auto comic generation features.

Next, participants entered the task phase. The high-level task was to create a comic about a programming concept of their choice. They were told to keep in mind that the end goal is to have code and corresponding comic that can be used together to teach students about the concept. Time limit was not imposed to investigate how users perform tasks in a 'natural' setting. Through our pilot studies, we confirmed that participants generally had enough time to finish the tasks.

After the task phase, we administered three surveys: (1) post-study survey with CSI [28], (2) Paired Factor Comparison (PFC) [28], and (3) System Usability Scale (SUS) [16]. We also conducted a short interview to ask participants to elaborate on their survey response to get a better understanding of what worked and what did not. The study lasted between 1.5-2 hours, and participants received \$30 Amazon gift card for their participation.

5.3.2 Part 2: Comic Evaluation

Our comic evaluation study followed the same procedure as in prior work [106], comparing the comics resulting from two different conditions (Baseline and CodeToon) to understand whether CodeToon helped generate comics of better quality.

Study Dataset

From the results of our user study, our comic dataset ² consisted of 24 comics (B: Baseline; C: CodeToon) including variable: 1 B; 0 C; condition: 7 B, 7 C; loop: 2 B, 2 C; function: 2 B, 3 C. We selected a subset instead of all comics for our survey for two reasons. First, it was not realistic to ask participants to rate all coding strips as that can take approximately 2 hours (i.e., 5 minutes per comic). We did not want to risk a survey receiving poor responses due to its length [56, 89]. Second, the quality of Baseline comics varied greatly. Since the amount of effort participants exerted ranged from very little to very high, we chose to select quality comics that participants put effort into (e.g., those who indicated they were satisfied with their work). In this way, we could minimize variability and keep the survey at a reasonable length.

Thus, we formed pair of comics (Baseline v.s. CodeToon) for the concepts by filtering for comics where participants answered in the post-study survey that they were 'highly

²Downloadable at https://codetoon-research.github.io/download/

satisfied' with their comics (9 or 10 out of 10 on 'I was satisfied with what I got out of the tool'), because participants' level of satisfaction varied (B: M=8.4, SD=1.6, Range=[6,10]; C: M=8.9, SD=1.3, Range=[6,10]). Then, we grouped them into sets based on their concepts. This resulted in 2 sets for condition, 1 set for loop, and 1 set for function, in total 8 comics to rate for a 30 to 55-minute survey.

Participants

We recruited 20 participants (age: M=24, SD=3.9) who can read and understand basic Python code through R1 university's study participant recruitment platform. Participants were mostly proficient in coding (1 Beginner, 1 Semi-Amateur, 4 Amateur, 8 Semi-pro, 6 Pro) and had moderate attitude towards comics' usefulness as a tool for teaching (2 Not at all, 3 Slightly, 8 Moderately, 4 Very, 1 Extremely useful) and learning (3 Not at all, 1 Slightly, 9 Moderately, 4 Very, 2 Extremely useful).

Procedure

After signing up to participate, participants received the Qualtrics survey URL. After demographic questions, the survey presented 8 comics in a random order. Each page started with the (programming) concept the comic is based on, the comic, the code the comic is based on, and then a set of evaluation questions related to (1) how well the comic maps to code, (2) how well the comic illustrates code and concept, and (3) how useful they think the comic is for teaching and learning the concept it is based on. Participants received \$ 10 Amazon gift card for their participation.

5.4 Results

Not surprisingly, Baseline comics were generally inconsistent in terms of their design language compared to CodeToon comics as all CodeToon users incorporated auto generated comics. This resulted in CodeToon comics being generally longer and more structured than Baseline comics. As for usability, both the Baseline and CodeToon users found the tool highly usable (SUS_B : 78.5 (SD=20.5) < SUS_C : 81.0 (SD=9.6)), giving usability scores well past the cutoff score (68) for production. To ensure anonymity, we use B112 and C112 to refer to Baseline and CodeToon users, respectively.

5.4.1 RQ1. Does CodeToon support the authoring of coding strips?

We analyze a mix of responses from survey, interview, and CSI results. We review the effectiveness of the proposed two features—story ideation and auto comic generation—and whether participants found it helped them be creative (CSI). Fig. 5.12 summarizes participants' ratings on the usefulness of the two novel features. We also provide a summary of Baseline users' responses for context—that is, what users with no creativity support experience.

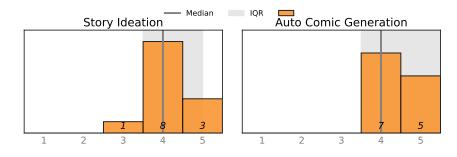


Figure 5.12: CodeTooon features' usefulness (1: Not at all useful; 5: Extremely useful)

Feature 1. Story Ideation

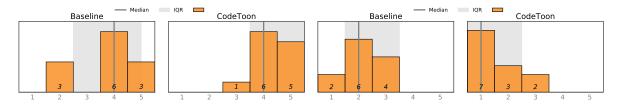
CodeToon users liked how the story ideation helped "spark creativity" (C5) and "think of creative ideas" (C3). C6 noted that "the options [in the dropdown]" provided "initial push of ideas" to get started on figuring out what kinds of "objects or things" could be thought of. C8 was quite surprised at how seeing the list of ideas could inspire him to create "much more content beyond this object." C2 appreciated the fact that even though they had "the extensive list of the objects that [they] could choose" from, they could also write anything inside the input boxes (suggesting CodeToon satisfied one of our design goals, **D2**).

Feature 2. Auto Comic Generation

CodeToon users liked the auto comic generation feature, because it saves "a lot of time creating the layout" (C12). C3 complimented CodeToon as a system that—even when compared to other commercial comic drawing tools—is more useful, because it "automatically generates the comics," decreasing the "workload by a big factor." Several participants identified the auto comic generation as a novel approach, asking whether there has been work like this before.

Impact of Two Features

Fig. 5.13 provides a further evidence that the two features facilitated the authoring of coding strips. Through the comparison, where the difference between CodeToon and Baseline is the presence of the two features, we see that they increased the perceived usefulness of the tool and decreased the perceived difficulty of the task.



(a) Usefulness (1: Not at all useful, 5: Ex- (b) Difficulty (1: Not at all difficult, 5: Extremely useful) tremely difficult)

Figure 5.13: Baseline vs CodeToon comparison on the (1) usefulness of the tool for the task and (2) the difficulty of the task with the tool. CodeToon users found the tool more useful for creating comics from code and the task less difficult.

Code-to-Comic Mapping Accuracy

As shown in Fig. 5.14, participants mostly found the overall accuracy and the accuracy with the code's execution and semantic very accurate. Many participants supported our design decision to map each line of the code to each row of the comic. C5 explained that she gave a high score to these accuracy questions because comic is mapped to code line by line. C3 explained that this is why, compared to before the task, he feels comic is more useful. C3 said, "I never thought [you] can teach the if conditions and conditional loops and everything to a student in the form of comics [in this manner]." C5 who prior to using the system thought about an alternative "event-by-event approach" acknowledged that this design choice is better, saying: "... it is very helpful to have the comics that go along with every single line just so that they can see what each line is actually doing. It's kind of like walking through the code step by step in a visual way... I think it's better that it's line by line, because [it] makes it a lot easier for learners to make that connection." C8 made an interesting remark that accurate mapping between the comic and code can help address concerns about metaphors sometimes failing to communicate the ideas accurately and that if one were to use comics to teach programming, it has to be mapped to code like we did:

I find that this tool maps the comic to the code very well. It's more accurate than I have ever expected...this balances the tradeoff between the barrier of learning the code and the inaccuracy of the metaphor. If any teachers can tell the student that this frame is mapped to which part of the code...It has to be like this. Strictly mapped to the structure [of the code]...this changed my thought on how helpful comic is.

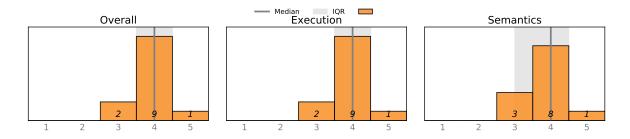


Figure 5.14: Mapping accuracy for auto generated comics. (1: Not at all accurate, 5: Extremely accurate)

Creativity Support

As shown in Table 5.2, CodeToon scored high and better than Baseline in all factors of the CSI. There was statistically significant difference in enjoyment, suggesting participants highly enjoyed using CodeToon. In the interview, many CodeToon users mentioned how fun the tool was. C1 said, "This is a really interesting work. I really enjoyed playing with it." Moreover, all the CodeToon users except for one (11 More Confident, 1 Same as Before) who was already "Somewhat Confident" in creating comics about programming concepts before the task answered that they feel "More confident" about creating comics about programming concepts after this task.

Authoring without Story Ideation and Auto Comic Generation: Baseline Users' Experience

Here, we describe the authoring process, experience, and challenges Baseline users encountered to contextualize CodeToon users' assessment.

Authoring Process & Experience. As Baseline participants described their authoring process, their stories signaled that our creativity support features and design process

	Baseline	CodeToon	Sig.
Factor	Score (SD)	Score (SD)	p
Enjoyment*	15.9(1.56)	17.6 (2.0)	0.01
Expressiveness	16(3.1)	16.8 (1.5)	0.21
Exploration	16(3.1)	16.4 (1.6)	0.39
Immersion	15.8(4.1)	16.8 (2.6)	0.24
Results Worth Effort	16.6(2.4)	17.8(2.1)	0.10
Overall CSI Score	80 (14.6)	85.2 (7.5)	0.14

Table 5.2: CSI Results. CodeToon performed better on every factor in creativity support. Statistical significance (p < 0.05) is marked with *.

aligned well with theirs. For instance, B9 described how he tried to "imagine a story" from a piece of code with numbers such as 100 and 70 as arguments in the if conditions. He explained that the number 100 made him think about "percentage." After thinking about "what things are 90% and 70%," he thought of "vaccine rate" and the "grade in a course." Several other participants (B5, B12) described the same mental exercise. B12 suggested that she "kept on drawing analogies in [her] story...comparing print function as printing a newspaper."

Participants complimented how the tool had distinct design stages and provided guidance through this interface. B12 said she "really liked the process of creating a story first, of creating five stories first, because it got [her] creative juices flowing and [she was] encouraged to think more creatively." Finding the overall design process to be "a very enjoyable process," B12 said, "believe it or not, I did not feel like it's work at all because the tool is very easy to use, and somehow creating stories and drawing connections is very enjoyable for me." B10 expressed the same sentiment, saying:

I think the tool had a good design. I like the idea that you had the story part separate from the visual part. That was something that would not have come to my mind being a beginner with creating comics. So I think that would not have occurred to me if it wasn't enforced by the tool, making me get the story ready before moving on to the visual part, so that was really interesting.

After the study, a researcher explained to B12 that a system in the other condition (i.e., CodeToon) automatically generated comics for users. Interestingly, B12 suggested that while "that would be helpful in teaching [learners] understand abstract programming

expressions[,] [she] still like[s] the idea of giving [learners] blank canvases so they can draw and create their own stories to understand code because it sparks imagination and makes programming more enjoyable rather than [feel like] a chore." She further added, saying: "when I was learning programming, I thought it was more like chore-like. I feel like allowing students to create their own stories and draw their own meaning from programming expression can make them less antagonistic towards programming. I mean maybe they like it even more and it would spark more of their interest."

Challenges. On the other hand, several Baseline users found the process of generating stories and comics challenging, as shown in Fig. 5.13b. They cited several reasons. One was that the ability to relate the abstract programming ideas to something concrete has atrophied. B4 noted that "we are used to programming so much now that connecting it with real life story is a big task [and can be challenging when asked to frame a story on programming all of a sudden]." B7 explained that it "seemed so hard to think of a story" but then seeing "drawings in the gallery like robots and the person, the laptop" helped him think of stories.

Another challenge they experienced was figuring out (1) whether they need to make a particular mapping when generating stories from code or comics from stories and (2) if yes, how to make that mapping. For instance, B6 said he was "confused" as to whether his "story also have to be a little more complicated [like his code]." Similarly, B13 also admitted that she "wasn't sure how many panels [she] needed." So she explained that she "went line by line [in the code] and drew a simple drawing." These demonstrate how participants felt lost on how to convert the abstract code to story or story into comic.

For others like B14, they found the task challenging because they felt it also requires one to have additional qualities such as creativity and visual communication skills. B10 said he was "trying to keep [story] relevant [by making explicit] link to the code...while also trying to keep [story] creative...so that it's immersive as well." B7 suggested that to use this tool, a user needs to be creative, to begin with. He said that even for teachers, they would struggle with creating stories and comics from code. He said, "not every teacher [can] explain programming concept in the form of comic... or come up with a story," because "not everyone is creative." She also noted that it can be challenging to find those who are both creative and proficient in programming, saying: "people can be creative, but then they they are not good at programming... [someone that is good at both] can be difficult to find." B8 suggested that the task has nothing to do with "whether [one] knows programming" but instead "depends on individuals' ability to visualize concepts." B11 noted that while it is easy to "see something in real life and say it could be [analogous to] a [computer] program," going from code to story (i.e., from abstract to concrete) is difficult, saying that looking at "X and 10 and true...didn't spark any stories off the bat." B5 also expressed doubt over "how [teachers] can be creative with [the tool]," as the instructors he met for his CS classes were not even effective at verbal explanations, which B5 felt is an easier task than visual explanations.

Another challenge included the time-consuming nature of the task. B7 indicated that time can be an issue: they "might have to spend some time brainstorming" and that "it can be difficult if [they're] trying to come up with something creative for every programming concepts." When asked to elaborate on his rating (6 out of 10) on the agreement question ("What I was able to produce was worth the effort I had to exert to produce it."), he explained that this is because "[he] spent like an hour on [the task] in total... but [ended up with] just stick figures." He said he "felt like [the output is] not really good considering the time [he] put in."

5.4.2 RQ2. Does CodeToon make the process of authoring coding strips more efficient?

Our analysis showed that CodeToon users were able to save more than 6 minutes on average for the comic authoring (T-test: p=0.08; Baseline: M=18:35, SD=11:17; CodeToon: M=11:45, SD=6:20) and the overall authoring time (Baseline: M=24:47, SD=13:43; CodeToon: M=18:27, SD=9:16). The average time spent on creating a story, on the other hand, was almost the same (T-test: p=0.7; Baseline: M=6:11, SD=3:13; CodeToon: M=6:41, SD=3:38), which can be attributed to the relatively less time-consuming and less challenging nature of creating stories when compared to creating a comic. While CodeToon saved more than 6 minutes, the overall time difference was not statistically significant (T-test: p=0.19).

This result is not surprising given that CodeToon can instantly produce comics. Baseline users needed to spend time brainstorming design ideas and creating them. CodeToon users, on the other hand, generated comics and spent remaining time simply adding details to them. CodeToon users seemed generally satisfied with the layout (i.e., layout of the panels): they added additional panels before or after but did not change the layout. C3 who uses a PowerPoint in the workplace to explain code flow to coworkers was impressed with how well CodeToon automatically generates a nice visualization to illustrate the code flow, saying "[after CodeToon] automatically generates [comics,] the only thing [left to do] is the finishing touch."

5.4.3 RQ3. What is the perceived utility and use cases of CodeToon for teaching and learning programming?

Perceived Utility: CodeToon Users' Perspective

As shown in Fig. 5.15, most CodeToon users perceived CodeToon as a very and extremely useful tool for teaching, learning, and novice learners. Most of them (8/12) said they would like to use the tool for teaching; the others (4/12) who answered 'Maybe' explained that this is because whether CodeToon is the right tool can depend on "age," "programming level," or "learning styles." ³ Likewise, there were variations in the perceived utility of CodeToon. While several participants mentioned that the tool would be especially useful for "younger students" (C9), others suggested that it can be useful for older students (C8), such as undergraduate students (C5), and even for experienced programmers, e.g., for "debugging" (C12). One thing everyone seemed to agree was that CodeToon provides a fun, creative way to learn programming.

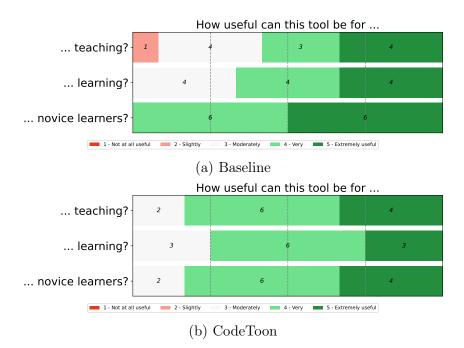


Figure 5.15: Perceived utility of Baseline and CodeToon for teaching and learning programming

³The author is not acknowledging the false notion that learning with one's "learning style" helps one to learn more effectively. The author is transcribing the phrase to avoid any loss of information or connotation.

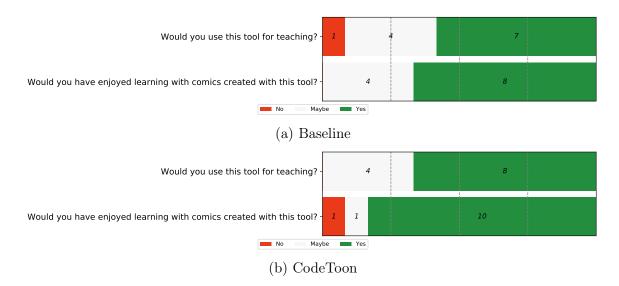


Figure 5.16: Interest in teaching and learning with Baseline and CodeToon

Participants provided several reasons for positively rating its utility. They thought that CodeToon can help students master "important programming concepts," namely loop and function (C12). C12 said, "especially in loop where [automatically generated comics] show i=0, i=1,... comics gives a visual explanation very well." C2 suggested that CodeToon would be useful for teaching data types for beginner students as it can show various examples (e.g., "apple costs 10 dollars") to help them develop intuitive understanding. C6 and C10 stated that the "scaffolding" in CodeToon would help students develop computational thinking, enabling them to see real-life situations in terms of code and vice versa (i.e., moving between different abstraction levels, as shown in Fig. 5.1). C6 pointed out that learning with CodeToon can show how "computer science can be engaged beyond looking at codes on a screen... [and that comic] is a very good lens to use."

Most CodeToon participants (9 More Useful, 3 Same as Before) stated that the experience of using CodeToon made them realize that teaching/learning with comics (created from this tool) can be more useful than they had thought. Several participants explained that they had no idea that comics could be used in such a meaningful way. They explained that they thought that comic was going to be used to merely narrate programming concepts. But seeing CodeToon generate comics that map to code line by line made them realize that comic can be "very useful" (C3).

Use Cases: CodeToon Users' Perspective

Participants provided several ways the tool can be used to teach and learn programming. One group of related ideas was using the task as an exercise in "classroom" or "tutoring settings" as well as a group or individual assignment, after which they can "present to class [their] story and understanding of the code." Another ideas were holding a "special programming class" where students learn to read code after learning their corresponding comic expressions and then using it for exams or quiz where "[students, instead of] writ[ing] paragraphs to explain what [a] piece of code [presented in the exam] means, they can make comics [to explain what they mean]." Participants also made several suggestions pertaining to how to use the tool. B6 suggested "start[ing] with the comic first and then go[ing] to the story [and then] to the actual code," saying that "this would help generalize the topic very easily." Several participants suggested using it to "visualize" basic programming concepts including "data types" as well as complicated concepts such as "pointer" (B12). Explaining how teachers utilize funny comic strips or memes in their teaching slides, participants also suggested that teachers can use the tool to quickly create and add visuals (comic) to their slides. C3 noted that the story ideation feature can also help teachers come up with appropriate examples and metaphors to explain the code and programming concept.

Perceived Utility & Use Cases: Baseline Users' Perspective

Even though some Baseline users found it difficult to manually create stories and comics and expressed concerns over whether teachers can use the tool due to such difficulties, many participants still praised the tool's usefulness for teaching, learning, and novice learners, as Figures 5.15 and 5.16 show. Most Baseline users said that they think teaching/learning with comics is more useful compared to before the task (9 More Useful, 3 Same as Before). Participants explained their support by acknowledging how it can help teachers effectively illustrate basic programming constructs (B9) as well as complex to intermediate level concepts (B12), adding that relating and portraying abstract code (e.g., x) as a "person" (B10) and "everyday situations" (B7) makes it easier to understand and accessible for students, especially "artistic and creative" students (B7) and "kids" (B7). B4 even suggested that the tool can be a "game changer" as "we are always able to understand the concept if it's picturized rather than if it's in theory."

Participants saw several learning benefits. One was that it encourages learners to think at a different level of abstraction and draw many connections. B12 who was already an experienced programmer pointed out that this activity enabled her to see programming in a different light. She said: It's very useful because I also need to think about what this function means on a different level and it allows me to draw as many connections as I can...it forces me to think about what the different use cases are for this function. What different ways are there to use it and such...while creating comic I felt like 'Oh my God, I just understand a lot more about the functionalities.' I felt like I get to see programming languages in a different lens, so I find it very useful for learning using this [tool].

Importantly, B12 and other participants (e.g., B6) enjoyed the process of creating stories and comic. B12 said:

I wish my professors could teach me these concepts in class in this way. It'll be a lot more fun. I had a lot of problem paying attention in programming class because it was so boring. My professors were basically saying, you assign this value and then you will get this and this. In programming classes, we are taught by reading lines and lines of code. It's very easy to get lost and distracted, but if I was being taught with such a visual representation, it would be a lot more engaging.

B4 elaborated on how the tool's ability to help learners "picturize" programming can yield learning benefits, saying: "if from the start you are making a person picturize things more clearly in their head, they can understand programming from the start and they won't feel it's a difficult thing, or that they should give up." But even as many Baseline users found the tool useful, they recognized several barriers to adoption. For instance, when asked why she answered differently for the tool's usefulness for teaching and learning (she answered "Extremely Useful" for teaching and "Very Useful" for learning), B4 explained that it will "take a little bit of time" for students with no knowledge of programming to first learn how to write code and be able to create a story and comic.

5.4.4 Does CodeToon help generate high-quality comics? (RQ4)

Fig. 5.17 shows how Baseline and CodeToon comics compare across the measures we investigated. As shown, there were statistically significant differences (Exact Wilcoxon-Mann-Whitney Test) between Baseline and CodeToon comics across all measures (accuracy, illustration, usefulness). In all measures, CodeToon comics were rated better than Baseline comics. That is, CodeToon comics were perceived as more accurate, illustrative, and useful

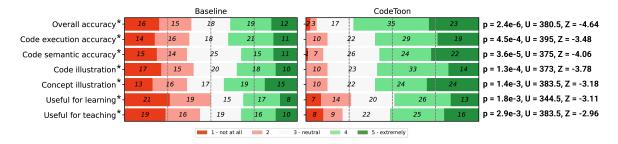


Figure 5.17: Comic evaluation results. Statistical significance (p < 0.05) is marked with *.

for teaching and learning. That CodeToon comics are rated more positively was also true in individual pair comparisons (see Appendix).

Another important metric is whether CodeToon consistently produces high-quality comics. Along with the general observation that Baseline comics vary in quality while CodeToon provides comics of consistent quality (cf. Fig. 5.18), responses to two agree/disagree statements from the CSI survey provide support. The two statements are: (1) 'What I was able to produce was worth the effort I had to exert to produce it'; (2) 'I was satisfied with what I got out of the tool.' In both statements (1 as Highly Disagree, 10 as Highly Agree), on average, CodeToon users ((1) 8.83; (2) 8.92) rated high and higher than Baseline users ((1) 8.16; (2) 8.42), suggesting that CodeToon users were able to consistently produce comics of the quality they are satisfied with.

5.5 Discussion

5.5.1 Implications & Opportunities

Code-Driven Storytelling. The storytelling process in CodeToon differs from prior storytelling processes involving programming as the storytelling is driven by code. Whereas the role of code has been creating animations and interactive stories in prior work (e.g., Scratch [120], Alice [37]), in code-driven storytelling, code functions as a blueprint for the stories: the logic or plot embedded in the code is translated into natural language and then into the visual language of comics. What is interesting about this approach is that it preserves the structure (logic) of the code throughout these abstractions, making it easier for learners to associate the code expressions and logic to natural language and reallife situations. One potential strength of this approach is in helping learners learn code expressions, syntax, and conventions by making associations with familiar abstractions (e.g., English, comic) while engaging in the creative process of storytelling. As the general ideas in this approach will likely work with any programming language, this work opens up exciting opportunities to explore how we can leverage story ideation and auto comic generation in a similar manner for other programming and even computational languages (e.g., math).

Storytelling with Text-based Programming. CodeToon enables a way to learn programming through storytelling but with the use of text-based programming languages. Learning programming through storytelling has traditionally been done using block-based programming languages and environments [37, 120]. Our computational pipeline for transforming text-based code to story and story to comic opens up a whole new direction to explore. For example, this approach could potentially enable us to teach young students who, because of their age, are first introduced to coding via block-based programming. We could test whether they can learn with text-based programming languages using this approach or whether they can learn comic expressions and then use them to learn corresponding text-based programming need to re-learn text-based programming when they grow older, this approach could potentially provide a shortcut to text-based programming.

Comics for Computational Languages. The feasibility of our approach means that we can explore how other programming languages and programming paradigms can also leverage comics. For instance, object-oriented programming is an area where its concepts and code are often presented in terms of real-life equivalents [144]. Comics could benefit students learning object-oriented programming. Other programming paradigms such as functional programming may require different abstractions (e.g., pointers) and help expand the set of visual vocabularies and advance our understanding of mental models for programming.

Design Implications. The findings from our study suggest that the line-by-line mapping to connect code and comics is an effective, useful design. This confirms the suggestion in the literature on multiple representational systems, which recommends making the mapping between multiple representations clear. More importantly, it shows that in the case of using code and comics as a corresponding representation, line-by-line mapping is recommended.

Visual Programming Environment. While CodeToon is developed as an authoring tool for coding strip, it is also a novel visual programming environment. Since using comics to teach and learn programming is a new approach, there is much work to be done. We need a curriculum containing a set of learning activities and guidelines to ease its adoption. Moreover, understanding the nature and impact of this approach needs to be investigated. For instance, what separates CodeToon from other visual programming environments like Scratch is that it can host artistic activities such as drawing. How can we incorporate this into teaching and learning programming? How can/does this impact learning in programming?

5.5.2 Limitations and Future Work

Limited Set. CodeToon has several limitations that need to be addressed. First is the limited set of categories for users. For the drawings in the auto generated comics, we used the dataset from Google's Quick, Draw! game [93], which offers sketches of 345 categories. But since they are quick, effortless sketches people drew under a short time frame, the quality was sub par in some cases. For a better quality, we chose only the AI correctly-guessed drawings. However, many of them were still not well-drawn. We saw several participants switching to different objects if the quality of these sketches was not satisfactory. We suspect these sketches gave some of our participants an impression that the tool is not for professional audience but for young audience. In the future, machine learning-based solutions may be able to help address this limitation. For instance, use machine learning to convert existing pictures into comic-style sketches and then use them in the auto generated comics. Or use machine learning to enhance users' drawings on the canvas. Methods like these can help CodeToon users not be limited by the drawings present in the system.

Lack of Options. The second is a lack of support for various design patterns and choices. For instance, because we aimed to show semantics for every line in the code, auto generated comics also illustrated code that is not executed (e.g., code inside if where the expression was False). Some participants changed the opacity of these panels, whereas some suggested not showing them. While some were surprised they were shown (hence 4/5 instead of 5/5 for some participants when rating mapping accuracy for semantics, shown in Fig. 5.14), they suggested that this design would be helpful for teaching. One participant (C12) also asked for a real-time rendering option. That is, when a user adds story to the story template, the comic is immediately rendered (or updated). Another participant asked to include the code expression in the auto generated comic. Suggestions like these indicate that there can be many different ways the comic can be designed and enhance the workflow in CodeToon. Therefore, an option panel where a user can toggle such options would help CodeToon accommodate varying needs.

Generative Models. An exciting avenue to explore next is further enhancing the two main features in CodeToon with generative deep learning models. Recently, large language models such as GPT-3 and Codex have shown impressive performance at language-related tasks, including generating code and translating languages, e.g., natural language into programming language and vice versa. These generative models can be an exciting addition to CodeToon. On user's request, generative conversational AI can generate or fill the gap in stories or code. Based on the analysis of the drawings on the canvas, it can also provide design guidance or suggestions on comic as well. For instance, if a user modifies the comic and it loses the relational structure carried over from code, it can provide suggestions or ask if if that is the intention, making the authoring process and experience more interactive, creative, and collaborative.

5.6 Conclusion

In this work, we introduce CodeToon, a comic authoring tool for facilitating the creation of coding strips by supporting story ideation and enabling auto generation of comics from code. To support this code-story-comic mapping, we used structure mapping theory and developed visual vocabularies for coding strips. Then we tested whether CodeToon successfully supports the authoring of coding strips and helps generate high-quality comics through two-part user study. The results of our user and comic evaluation study shows CodeToon not only supports the authoring of coding strips well but also reduces the comic authoring time by a significant amount while producing accurate, informative, and useful coding strips. In addition to contributing CodeToon and methods for generating story and comic from code, our work contributes an analysis of structure mapping theory and the visual narrative grammar to show how we can leverage them to generate story and comic from code.

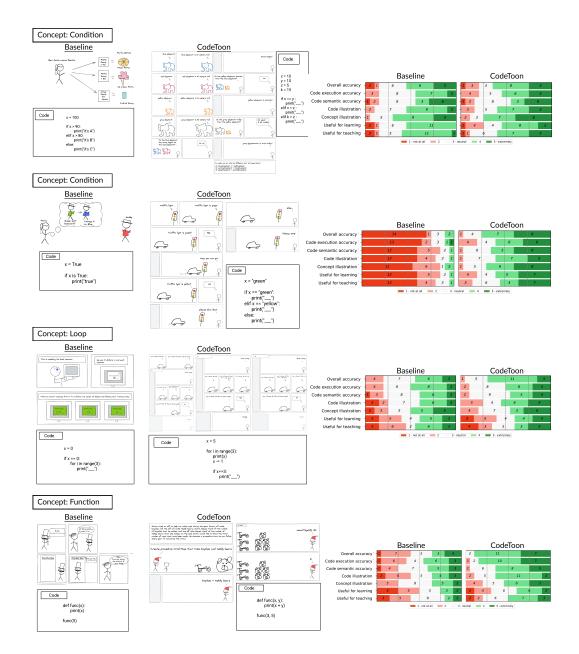


Figure 5.18: Comparisons of individual pairs according to their concepts. CodeToon comics were rated as being as good or better than Baseline comics in all cases across all measures.

Chapter 6

Use Cases for Coding Strip

The psychological profiling [of a programmer] is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large.

Donald Knuth

In previous chapters, I introduced two methods for designing and creating coding strips. They contribute to making the production of coding strips scalable. Once we have coding strips, the next question becomes how we can use them. To demonstrate how they can be used, this chapter presents an in-class study where we tested four use cases at an introductory computer science course with university students. In this chapter, I describe how they were administered and how students perceived them.

6.1 Methods

Coding strip was used to teach programming concepts in a CS1 course in four ways. In this section, we describe the course, the four use cases of coding strip, and the post-semester survey.

6.1.1 Course & Student Information

The study was conducted in a first-year computer science course (N=49) at the University of Waterloo. The course is designed primarily for students who do not major in computer science (e.g., students in Arts). Seventy percent of students in this course were in the Digital Arts Program, and for them the course was required. The course followed the *creative coding* approach [115], where students learn computer programming by manipulating media (e.g., graphics, sound) and creating interactive graphics. Students used p5.js, a popular Javascript library for creative coding. Therefore students learned and programmed using Javascript syntax and conventions.

6.1.2 Use Cases

While the four use cases of coding strip we tested are not an exhaustive list of all potential use cases (UCs), they represent common teaching tasks relevant for most, if not all, programming courses.



Figure 6.1: A Spiderman comic used to introduce the idea that values in variables change (UC1). Here, variables are *name*, *mood*, *age*, and *hobbies*.

UC1. Introduce Concept

The Spiderman comic shown in Fig. 6.1 was used in week 3 of the course when the concept variable was being introduced for the first time. The course instructor displayed a slide with the comic before the start of the class for students arriving early. Once the lecture began, students learned that variables store values and that associated values can change over time. The instructor then showed the Spiderman comic again relating it to how values associated with Peter's "name," "mood," "age," and "hobbies" change over time. For this use case, only the comic was used; its corresponding code was later used in a clicker question for review (Fig. 6.3a).

UC2. Introduce Code

Inspired by prior work [137, 141], comics were used to scaffold code expression in several ways, as shown in Fig. 6.2. Fig. 6.2a shows a sequence of three lecture slides which begins with a comic strip that portrays a character putting money into a piggy bank for the concept variable assignment, followed by its corresponding code, and a slide with its related concepts variable declaration and initialization in the same piggy bank context; in other words, it (1) begins with an intuitive, relatable abstraction, (2) connects it to its code expression, and (3) explains the concept by comparing and contrasting with related concepts. Fig. 6.2b shows another sequence of slides which progresses from comic to English and to code; students were shown a clip from the movie Dr. Strange [40] in which Dr. Strange traps a villain Dormammu in an endless looped time; then, the instructor showed three panels of an image (first slide of the sequence) from the clip to highlight the repeating sequence; this scenario was then presented in English (with indentations to mimic the code syntax) and finally its corresponding code. Fig. 6.2c shows a progression from a slide with comic & English to a slide with comic & code. In this sequence, the goal was to introduce the code expression for the array and highlight how array indices start at 0, not 1. As shown, students were first shown the comic with a frog jumping from one lily pad to another. In the following slide, the expressions 'lilypad' and 'lilypad + 1' changed to 'lilypad[0]' and 'lilypad[1]' to show the corresponding code expressions in array syntax. The +1 above the frog was used to provide an intuitive explanation for why array indices start at 0 and not 1. This is something novice learners often struggle to grasp because they think indices represent ordinal numbers (e.g., index 1 points to 1st element); they do not realize that indices represent offsets (i.e., index 1 points to 2nd element, index 1 means 1 position away from the 1st element).

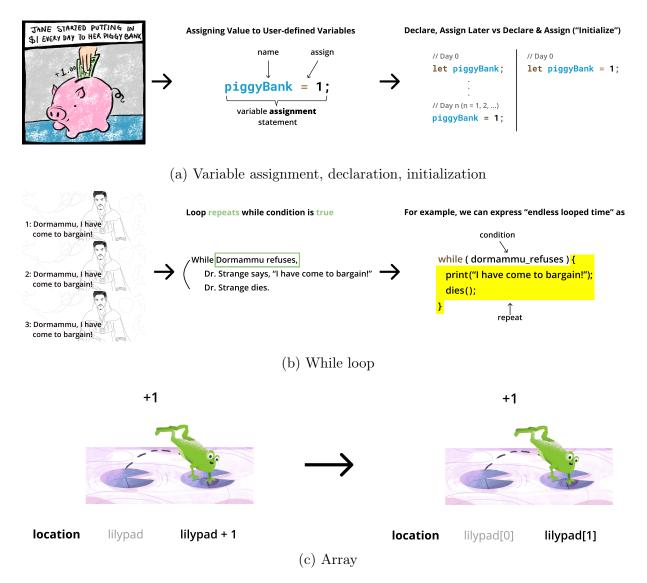


Figure 6.2: Sequence of lecture slides used to introduce code with comics (UC2)

UC3. Review Concepts and Code

After introducing concepts and code expressions with comics, students can review them using the same comic, its code, or context. Fig. 6.3 shows three clicker questions used to review concepts and code that were presented in Fig. 6.1 and 6.2. Fig. 6.3a required students to track changing values in a variable 'age' which was previously shown in the Spiderman comic (Fig. 6.1). Fig. 6.3b asked students to select an appropriate abstraction

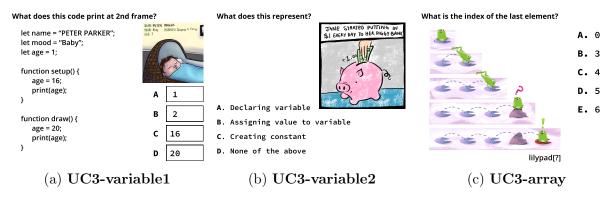


Figure 6.3: Examples of reviewing with clicker questions (UC3)

('assigning value to variable') for the piggy bank illustration from Fig. 6.2a. Fig. 6.3c was used to test whether students understood that array indexing starts from 0 as they were taught with Fig. 6.2c.

UC4. Write Code from Comics

Another use case suggested in prior work [143] was coding exercises, that is, have students translate comics into code. Fig. 6.4 and 6.7 show three coding strips used for the exercises. During the lecture, students were shown the comics without the code and were asked to submit their code on Socrative, a web-based response system [129]. Since it is unreasonable to expect students to submit the exact same code as the instructor, students were promised full participation marks for simply submitting. After students submitted, the instructor showed the list of submissions and his code, and used submissions that reflect students' unique interpretation of comics to highlight that coding is a tool for creative expression.

6.1.3 Survey

At the end of the semester, we asked students to complete a survey to evaluate their experience with each use case and the idea of using comics (we used the word *comics* instead of coding strip in the survey to avoid unnecessarily overloading students with new terminology). While one of the authors was the instructor of the course, we followed the guidelines from the university's ethics committee to ensure we did not exert undue influence on the students to give us permission to use their data or to answer with any bias. Specifically, a research assistant unaffiliated with the course administered the Google

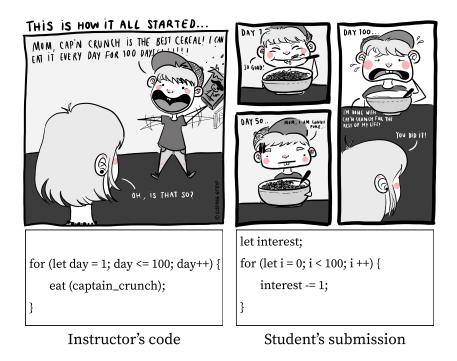


Figure 6.4: Coding strip example on for loop used for one of the use cases: writing code from comics (UC4). Students were asked to translate the comics into code. A sample submission shows how comics can be interpreted in multiple ways.

form survey via the course's online platform and collected the survey responses. Students were assured the instructor would not know who had permitted the use of their data until after their final course grades were submitted.

In order to help students remember and differentiate the four different use cases, we included a page in the survey with the description and lecture slides used for each use case (Fig. 6.4, 6.1, 6.2, 6.3, 6.7 were used), before the questions about each use case. The survey began with some demographic and programming attitude questions, and then included questions about the use of comics, which progressed from questions about the use of comics in general and then to questions specific to each of the four use cases. At the end, they were also asked to specify whether they thought other computer science instructors should use comics. Scale-based questions about the comics used a 7-point Likert scale where 1 indicated 'Really Disliked' and 7 indicated 'Really Liked.'

6.2 Results

We present the analysis of survey responses about all use cases and analysis of code submissions for use case 4 (**UC4**). Specifically, we describe what students liked and disliked about each use case, as well as why students do or do not recommend using comics in other computing courses, in order to present the benefits and challenges with using comics. To ensure anonymity, we refer to students as S141.

6.2.1 Demographics

Of the 49 students enrolled in the course, 42 students completed the survey and 41 students consented to the use of their data for research. Hence, our analysis is based on these 41 students (15 M, 26 F, 0 Non-binary; Arts: 28; Science: 9; Health Science: 4). Most students were undergraduate students in their first or second year (18 First, 15 Second, 4 Third, 3 Fourth), with one student in a post-degree program. The majority of the students were taking the course to meet degree requirements (28), while the rest (13) were taking the course out of interest in learning programming. Half of the students were retaking the course (21 Retake, 20 First). In terms of their programming experience before the course, most students had limited or no experience (15 No Experience, 18 Several Hours or Days, 8 Several Weeks or Months). Students were evenly split in terms of their interest in learning programming of the course (15 Not Interested, 16 Interested to Certain Degree, 10 Highly Interested). At the beginning of the course, more than half of the students (25) perceived learning programming as 'difficult,' while 14 saw it as 'manageable' and 2 saw it as 'easy.'

6.2.2 Analysis of Each Use Case

UC1. Introduce Concept

Students generally liked being introduced to concepts with comics (M=4.9/7, 95%CI=[4.3,5.4]). Thirty-five students (85%) who rated it positively (scores of 4-7) explained that they liked it because it was not only "fun" (S1, S29), "engaging" (S32), "related to something[, such as superheroes, they] like" (S8), "an interesting way to learn [the concepts]" (S33, S37), but also "piqued [their] interest for the concept" (S9), and made programming less "scary" (S40) and the "material less dry" (S32).

Students also mentioned that comics helped them better understand the concepts (S7, S28, S33, S35), as they explain "why" (S17), help them "visualize the concept" (S13, S15,

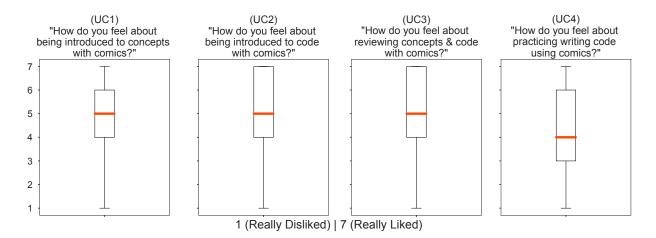


Figure 6.5: Students' assessment of four use cases

S41), "simplify the tricky concepts into a more digestible format" (S26), provide "good analogy" (S22) and "metaphor" (S2), and give "another point of view" (S28) to help make sense of the concept. The sequential nature of comics was also helpful in understanding the procedural aspect of the concepts:

Comics gave a logical explanation to the concepts applied. A lot of the time, we don't know what the program is doing; the comics made a logical sequence of concepts that made it easier to learn. (S19)

Another important benefit was that comics helped students remember and easily recall the concepts; S27 and S5 mentioned that they recalled the comics "during the midterm" (S27) and whenever they needed to remember, for instance, "what loop does" (S5).

Six students (15%) who rated this use case negatively (scores of 1-3) offered two major explanations. One was that the comics were confusing (S3, S16, S20); S25 said he had difficulty understanding "how [the comics and concepts] correlate." The other argument was that the method is "not applicable to all students" (S34), which coincided with S27's comment on his preference for learning with analogy alone to learning with comics.

Students also shared several ideas on how we could improve this use case. S30 observed that since "it may be harder to understand [the comics] if the student isn't familiar with the [concept already]," we may want to use comics some time after the student has learned the concept. Several students (S9, S20, S23) also suggested that comics may be more useful when introducing "more complex topics, like loops and arrays" (S9) and "concepts which need step-by-step explanations (ex: loops)" (S23). This seemed to align with S31,

who found the frog comic (Fig. 6.2c)—which shows a frog leaping from one lily pad to another in step-by-step manner to illustrate looping through an array—more useful than the variable comic (Fig. 6.1).

UC2. Introduce Code

Students also liked being introduced to code with comics (M=5/7, 95%CI=[4.4, 5.6]). Many of the benefits reported for this use case were similar to those for **UC1**: it was "engaging" (S7), "fun" (S31), and a nice way to introduce "the code in a more interesting way" (S14, S26). Several students also reported that introducing the code with comics made the code "easier to remember" (S5, 18) compared to being introduced to code with text only. S40 also contrasted it with the text-only approach to point out that learning code with comics allowed her to focus on understanding the code instead of merely memorizing it, which she resorted to whenever code was presented in text only.

Students (S1, S35, S40) reasoned that comics made code "easier to understand" (S1) because they provide visual structure (S39) and show "the logic behind code" (S22). S19 explained how comics helped her make sense of loops: "loops are hard to understand because [you need to make sense of] the exit point and amount of times it should run, but when Dormammu finally says stop, it made sense way easier than learning it from scratch." Finally, some students mentioned that being introduced to code in this manner "relieved some anxiety" (S2) and helped them develop a positive attitude towards learning programming (S26).

Like UC1, a few students mostly found comics "confusing" (S15) and "not useful" (S17). S3 and S24 thought it was "unnecessary" because they assumed "all [students already] understood the concept of a loop." Since most students, however, commented on their usefulness, it seems that it was a welcome intervention for most students other than a few who already understood the code.

UC3. Review Concepts and Code

Students also enjoyed reviewing the learned concepts and code with comics (M=4.9/7, 95%CI=[4.4,5.5]). They said it helped them better understand and remember the review content (S8, S31) and that it is "fun" (S11, S21) and "a good way to refresh memories" (S20, S22), as it makes them "pay attention" (S1). S14 shared that while "[he] was confused by some comics," he still liked it "because [reviewing with comics] was a good checkpoint for [him] to determine if [he] can apply the concept." S19 described how helpful one of our review questions (Fig. 6.3c) was for her understanding of array:

Absolutely brilliant. Using a leap frog going through an array because we don't know how to grasp the concept that the program goes from 0 to 1 to 2 in a set of array... this HELPED A LOT. (S19)

While it needs further testing, we also found that students generally performed better on clicker questions when they referenced comics (Fig. 6.3). The average percentage of correct answers for non-comic based clicker questions (60%) was lower than that of the comic-based clicker questions (74% for **UC3-variable1**, 67% for **UC3-variable2**, 86% for **UC3-array**). We also directly compared **UC3-variable1** with its isomorphic (i.e., corresponding in format/task) question, which was administered in the next class without comics references (i.e., 'let x = 1;' instead of 'let age = 1;' and no comics in the slide), and found that only 34% of the students responded correctly compared to 74% for **UC3-variable1**. We did not administer isomorphic questions for the other two.

While most students rated this use case positively, some students reminded us that a few factors could potentially undermine students' experience with this use case. S15 noted that he was able to enjoy reviewing with comics because he already knew the concept well enough and thus did not find the comics confusing. S40 who rated this use case highly (6/7) warned us that since this use case affects students' grades, we may want to ensure that the comics are clear. While the clicker question grades served more or less as a participation grade and represented a small portion of the overall grades (they were 5% of the total grade and the best 75% of clicker grades were used), a few students were careful about fully supporting it as they felt some students got certain questions wrong "simply because the [comic] was confusing, rather than them not understanding the topic" (S3).

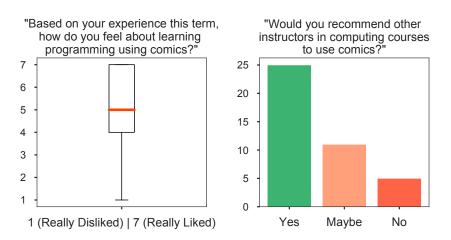
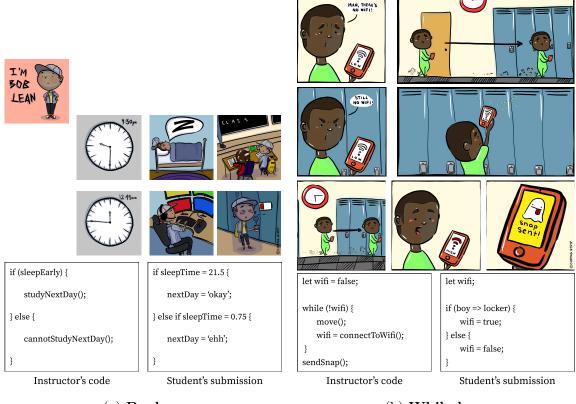


Figure 6.6: Students' assessment of comics in general



(a) Boolean

(b) While loop

Figure 6.7: Two of the three coding strips used for writing code from comics (UC4). One other coding strip is Fig. 6.4

UC4. Write Code from Comics

Compared to other use cases, students were not as positive about the idea of practicing writing code from comics (M=4.2/7, 95%CI=[3.6,4.8]). Many students found the exercise "difficult" (S11, S14, S18, S20, S25) because they had to write code and interpret comics. This is understandable since students can find writing code from scratch challenging and interpreting comics difficult due to the task's open-ended nature. Since these resulted in a less positive experience and confusion among some students (S3, S5, S24), this exercise would benefit from clear guidelines with examples of code submissions. On the other hand, there were also students who understood it after the first time (S12), were interested in "learn[ing] more" (S8), found it "fun" (S18, S31) and "very useful to practice and go over the concepts in a limited time" (S26). Some students felt the exercise "[made] programming seem less intimidating" (S39) and more interesting:

I have always tried to find the 'right' answer because I have been educated that there is only one right answer and I need to find it. So when I saw that [comic] in class, it seemed like a big difficulty and I didn't know what to do. However, this made me more interested in programming after realizing that in programming, there is no right answer and the result depends on what I'm creating and expressing. (S40)

This comment on how the exercise changed S40's view on programming was an unexpected, yet exciting, finding. Upon analyzing students' code submissions, we found many interesting interpretations of comics, some of which are shown in Fig. 6.4 and 6.7.

6.2.3 Analysis of Overall Experience

Students, in general, were highly positive (M=5.2/7, 95%CI=[4.7,5.7]) about the overall idea of learning programming using comics. When students were asked at the end of the survey whether they would recommend that other instructors teaching computing courses use comics, more than half of the students were in support, and very few were against it (25 Yes, 11 Maybe, 5 No). Students were asked to elaborate on their response and we present the analysis of their answers below.

Reasons for Recommending

More than 60% of students recommended the use of comics in other computing courses for a variety of reasons. Many students recited the benefits from previous sections, including the comics being "fun" (S21, S35), "engaging" (S18, S32), "motivating" (S31, S35), helpful in "remembering" the concepts and code (S22, S27). S7, who was taking the course for the second time, compared his experience from the previous term to stress that comics helped him understand the materials better. Students also mentioned that comics help "break down big coding ideas" (S31) and "give a visual representation of what is happening" (S20) and that it is an "appealing way to learn" (S8) for students who prefer learning with visuals (S6, S12, S15, S39). Last but not least, they also spoke of the positive impact it has on the classroom atmosphere by making lectures more engaging (S1, S20, S21, S29).

Students were divided, however, on the extent to which comics may be useful in other computing courses (Figure 6.6). S32 said, "it definitely works for the lower level CS classes. Maybe not for the CS major classes, though." On the other hand, S22 believed it could be used for learning any programming language. S8 noted that since the "majority of people like comics," it would be a welcome intervention in any course. S4, who had prior programming experience, speculated that while he did not need the comics, other students with a little programming background would like learning with comics.

Reasons for Reservation

More than a quarter of students had reservations about comics being used in other CS courses. S27 said it would be okay "if they don't solely rely on it as a teaching method." Some were hesitant because they felt that comics' usefulness depends on a number of factors, such as "[the clarity of the] the [comics'] link to code" (S24), "[its] content," "the student's learning style" (S10), "how [much they] enjoy/understand comics" (S1), and which use cases are used (S15, S23, S36).

Reasons for Not Recommending

Five students who did not recommend the use of comics in computing courses explained that they found comics "confusing" (S3), not "useful" (S17), not suiting their "learning style" (S25) and "hard to see from the back [of the classroom]" (S34). S38 thought that "because all computing courses cater to different needs [of the students] it is not for everyone."

6.2.4 Analysis of Demographics

We examined whether students' assessment of the overall use of comics and each use case varied by demographic attributes—gender, major, the reason for taking the course, prior experience with the course, prior programming experience, prior interest in learning programming. We found no statistically significant differences between the groups within these categories. This is noteworthy because it suggests that students' experiences with coding strips are similar, regardless of their backgrounds. However, our sample size was small and differences across these demographics may emerge in future work with larger samples.

6.3 Discussion

Students generally enjoyed learning with comics and experienced various benefits, some of which were related to dual coding effects, such as being able to "remember" and "understand" the concepts and code more easily [111, 90, 5]. Although our study does not contribute any measurement of learning impact, it still provides valuable findings to help facilitate the use of coding strip. Specifically, it contributes an understanding of how coding strips can be used to support teaching in an undergraduate CS1 course, how students perceive them, and what needs to be improved.

Our study also found that while the idea of learning style has been dismissed due to lack of evidence [112, 121, 80], numerous students still held this misconception and cited their learning style as the reason for liking or recommending the comics. Likewise, this was also the reason some students disliked or hesitated to recommend comics: they thought non-visual learners do not benefit from comics as it is "visual-based." In addition to stigma associated with comic books, we find that this misconception about learning style may be another challenge that needs to be addressed since it seems to significantly impact students' perception and acceptance.

As we move forward, the following issues need to be addressed. First, while the use cases were generally well-received, the coding exercise (UC4) needs to be improved. For instance, it needs a clear process to guide students and a variety of examples to help them feel comfortable with the idea that code can have multiple interpretations. Also, a number of students attributed the ambiguity of the comics causing confusion as the reason for disliking the use cases or not recommending comics. While a few students described what caused the confusion (e.g., lack of clear link between the comics and code), there is still a lack of clear knowledge on what (e.g., particular design aspect of the comics) made them confusing. Finally, our results imply that students' perception of coding strips can depend on the timing of when the comics are introduced; more investigations are needed to understand these more nuanced effects.

While we used comics throughout the course, they were not used for all concepts and

code. The concepts spanned multiple levels of difficulty (i.e., variable as "easy," loop as "medium," and array as "hard"). However, it is unclear if and how this may have impacted students' assessment. Furthermore, the students in the class were non-CS majors, with most of the survey responses (68%) coming from students in the Digital Arts Program. Thus we need to investigate whether CS majors experience the same benefits. While we explored several ways coding strips can be used in this study, we did not explore what aspects made certain coding strips more or less useful or confusing for learners. As future work, we can analyze specific coding strips' styles and their learning effects in order to develop design guidelines for coding strips. Finally, we need to find out in what alternative format or ways the instructors should use coding strips to accommodate blind or visually impaired students.

6.4 A Summary of Use Cases

Here, we summarize a set of use cases tested in this study and mentioned throughout this thesis. This list is not exhaustive but should provide concrete ways coding strips and the tools developed in this thesis can be used.

- Teaching
 - Introduce concept [Section 6.1.2]. Comic can be used to scaffold a concept. Comic provides an intuitive grounding for the concept through visual narrative. The corresponding code can be used when the concept is being reviewed or tested, as described in Section 6.1.2.
 - Introduce code [Section 6.1.2]. Comic can also be used to scaffold code expressions. As Fig. 6.2c shows, comic can hold textual/visual elements (e.g., 'lilypad' and 'lilypad + 1') that resemble target code expressions ('lilypad[0]' and 'lilypad[1]') for scaffolding.
 - Leverage analogies/stories [Section 5.4.1]. CodeToon makes creating analogies and stories easy. Thus when teaching programming concepts or explaining the code, teachers can utilize CodeToon to generate analogies and stories to enhance their explanations (Section 5.4.3).
- Learning
 - Learn code expressions using comics [Section 5.4.3]. This would be especially suitable for younger and novice learners, as comic can be much easier for them to

understand. However, it would be important to note the premise of this idea: as we did in Chapter 5, a pre-defined set of visual vocabulary must exist for the building blocks of a programming language—programming constructs and language-specific syntax or conventions.

- Review/Quiz
 - Review concept and code [Section 6.1.2]. After concept or code has been taught using coding strips, its comic, code, or context can be used when reviewing the concept or code. In this chapter, the review occurred using clicker questions. However, it does not have to be limited to clicker questions. As was discussed in 5.4.3, it could potentially be used in a quiz or test, with students actually creating comics to demonstrate their understanding of the concept.
 - Write code from comics [Section 6.1.2]. Presented in this chapter using the existing coding strips. This can be challenging for novice learners or first-timers, but once they get used to the exercise, they may actually enjoy it, as some of the students from this study professed. As noted, it may be important to not grade this exercise as there can be multiple answers. It may be okay to use this as quiz if students and teachers have reached understanding that answers can be constrained. Otherwise, this exercise should focus on enabling students to exercise their creativity, as that seems to be where this use case benefits students the most.
 - Fill-in-the-blank [Section 4]. Several participants suggested leveraging the 'fill-in-the-blank' format to test/quiz students. The blank part can exist in either code or comic. For instance, a teacher may provide code and corresponding comic that has one or more panels missing. Then a teacher can have the student describe what panels (e.g., execution step) are missing. Or the missing element in the comic can be text that correspond to the value in the memory. We can apply the same idea on code, e.g., cover a value in the code and make the student guess what the value must be.
 - Debugging [Section 5.4.3]. This is a debugging exercise where a learner has to pinpoint what needs to be changed in the code to output a given comic. similar to fill-in-the-blank in that a student is Debugging requires a programmer to find a bug in the program.
 - Answer with comics [Section 5.4.3]. Create comic to explain concept and demonstrate your understanding (in test context). This requires, however, that there is a way to avoid ambiguity intervening to make it difficult to discern whether students have correct understanding of the concpet. A potential solution can be using unified comic language, to narrow down the possible set from which students can use. This

way, students can provide an answer that teachers can expect. In terms of administration, rules concerning using the unified comic language would need to be informed in advance.

• Assignment/Exercise

- Group activity (Co-create with peers) [Section 5.4.3]. The design workshop study in Chapter 4 involved participants engaging in the group activity. Participants who did not even know each other prior to participating in the workshop—thoroughly enjoyed working together, as detailed in the chapter. As mentioned, teacher participants recognized its potential to afford engaging learning experience for their students. As students work together on simple to complex concepts, this activity can provide them with the opportunity to learn from their peers and reinforce their understanding through interaction. We can also consider administering a group activity with CodeToon. Currently, it does not have real-time collaboration support. Thus it can be done if the activity takes place in person. Once real-time collaboration feature is implemented, it would be nice to explore a remote group activity on CodeToon.
- Create comic and present to class to explain concept/code [Section 5.4.3]. Creating a comic that demonstrates one's understanding of the learned concept as a part of assignment or in-class exercise is an approach that has been explored in prior work [113, 20]. This can be an engaging activity for students as it incorporates social aspect to the learning process. Moreover, this is an example of learning-by-teaching. As students need to envision themselves as teachers, this can be a nice opportunity for students to refine their understanding and practice communication skills.

6.5 Conclusion

In this work, we tested four use cases of coding strips and contribute an experience report to share how they were used, how each use case was perceived by students, as well as the benefits and challenges associated with each use case. While the four use cases we deployed are not exhaustive, they address basic teaching tasks and we believe this contributes a useful report for instructors interested in using coding strips to introduce programming concepts and code. While more work is needed to understand the most effective ways to use coding strips, our work contributes an essential first step towards their use in computing education and demonstrate how they can be integrated into existing pedagogical approaches. Overall, the results shows that coding strip provides a useful and enjoyable way to learn programming.

Chapter 7

Discussion

But, as the abstraction ladder has shown, all we know are abstractions.

S. I. Hayakawa [73], p. 160

This thesis began with the following problem identification: dead-level abstracting permeates instructions and teaching in computing education, making it difficult to learn programming. The lack or absence of moving up and down the ladder of abstraction is not conducive to computational thinking, even though it is often touted as the aspiration of computing education. What contributes to this problem is the lack of model and tools to allow learners to move between multiple abstraction levels. Coding strip, a form of comic strip with corresponding code, was formulated and researched to address this. The following research questions guided this thesis:

RQ1: How do we design coding strips? What is the design process?

RQ2: How do we facilitate the authoring of coding strips?

RQ3: How do we use coding strips for teaching and learning programming?

To answer these questions, I conducted three separate studies. In the first two studies, I introduced methods for designing and creating coding strips. In the last study, I tested four use cases of coding strips in an introductory computer science course. The studies successfully addressed these research questions. Below, I summarize the findings from each study and discuss design implications and future work.

7.1 Summary of Findings

7.1.1 Designing Coding Strip with Ideation Workflow

This study was conducted to understand how to design coding strips. While one may argue that it is better to first explore the benefits of coding strips to understand whether this intervention is even a meaningful path to explore, measuring benefits is challenging if we have no way of knowing whether the administered coding strips were correctly 'designed' (or designed as intended) in the first place. In other words, without a proper and indepth understanding of the design of coding strips, interpreting results can be difficult as it is unclear on what basis we are evaluating them. For instance, if certain coding strips are perceived to be more useful than others, how can we explain this? Therefore the reasonable step was to investigate how we should design coding strips. For this, we developed a design process and tools (ideation cards and design board), and then evaluated them in two design workshops, one with students and the other with teachers. Our study showed that our design process and tools are effective at supporting the design of coding strips. We evaluated the effectiveness of this procedure and design tools (i.e., ideation cards and design board) by surveying participants and counting the number of ideas they generated, as well as how many cards were used on average.

Several ways we can extend our design process and tools were found from the study. Our design tools consisted of trigger and scenario cards to help brainstorm ideas related to concept and story ideas, and design cards to provide comic design ideas. We found that design patterns in design cards can delineate the design space for coding strips. While we cannot argue that our design patterns provide an exhaustive list of all possible designs for coding strips, some participants' responses suggest our set provides more than enough patterns as they indicated there were too many cards to go through and process (each card had a description of when and how to use the design). Teacher participants also noted that our design process and tools can be extended to cover concepts and topics in other subjects. Because they often use metaphors and analogies to explain ideas in their math or even social science classes, they felt our design process and tools would be useful in those cases as well.

Another interesting finding was that both students and participants found the process of creating coding strips to be a very engaging and fun exercise that offers additional benefits of refining students' knowledge and teaching them how to collaborate with others. While validating our design process and tools was the main goal of this study, both student and teacher participants suggested this would be a good learning exercise for students. They expected, however, that students would need to have a good grasp of the programming

concepts before they can engage in the activity as they need to be able to discern what is a "correct representation" for the concept. I also found that participants varied in their perceived extensibility of coding strips and ability to create coding strips for complicated concepts. Some student participants who created coding strips for simple programming constructs expressed doubts on whether coding strips can illustrate more difficult programming concepts, while others were able to create coding strips for more complicated concepts and algorithms such as recursion and merge sort.

7.1.2 Creating Coding Strip with CodeToon

This study was conducted to ease the creation of coding strips. While participants from the design study enjoyed the process of creating coding strips and showed they can draw comics after a brief stick figure drawing exercise and with design patterns, several participants nevertheless desired a digital authoring tool where they can use templates (e.g., design patterns) to quickly create comics, explaining that this allows them to direct their attention and energy to what they felt are more meaningful, fun part of the process—reflecting on the properties of the concept and generating relevant stories. To ease the creation of coding strip, we developed CodeToon which leverages two creativity support features—story ideation and auto comic generation.

To understand the effectiveness of CodeToon and the two features, I conducted a twopart user study, where the first part involved users using the system and the second part other users assessing the quality of the coding strips generated from the first part. The findings from the two-part studies showed that CodeToon and the two features not only enable users to quickly create coding strips but also yield coding strips of higher quality. These results suggest that our creativity support features which leveraged structure mapping theory and visual grammar were appropriate design choices for translating abstract programming language to natural language and then to visual language. CodeToon users acknowledged that metaphor suggestions helped them brainstorm story ideas and complimented our design choice to map each row of the automatically generated comic to each line of the code.

One interesting thing to note from the findings was how much both Baseline and CodeToon users enjoyed the process of creating coding strips. The CSI results suggest that CodeToon users enjoyed significantly more; this can be attributed to the two features that allowed CodeToon users to more easily and quickly create coding strips compared to Baseline users; in fact, Baseline users mentioned having have to create comics (albeit with templates) and make decisions on how to map comics to code and story as challenges, further supporting our analysis that the two features were critical in the user experience. But our analysis also showed the value of having users manually create stories and comics without the support of the two features as they can lead to more creative stories and comics. Overall, the results lend additional evidence that the process of moving up and down the ladder is a fun and engaging exercise with or without creativity support features.

7.1.3 Using Coding Strip

To enable the adoption of a new tool, it is important to demonstrate how it can be used. This study was conducted to show how coding strips can be used. Among many use cases suggested, we looked specifically into four possible use cases of coding strips in the context of teaching university students in an introductory computer science class. While the tested use cases are not exhaustive, I explored four use cases that represent common teaching tasks relevant for most, if not all, programming courses. They included teaching concepts and code, reviewing the learned concepts and code, and writing code from comics as an exercise. The survey responses suggested that users generally liked all the use cases as they helped them in a number of ways. For instance, they helped them understand and remember the learned concepts and code, and made them even more interested in programming. While many participants found writing code from comics difficult because it requires them to write code, we found that this exercise can be used to address any misconception students can have about a coding problem having only one solution.

7.2 Implications & Opportunities

Here, we draw out implications and open questions from the findings from these studies.

Rubric. As we explained above, we chose to first research how to design coding strips so that we can build on a rigorous understanding of coding strip designs for future analyses. While the design space for coding strips was mapped with design patterns and several factors that make certain coding strips better in terms of clarity and design were discussed, a rubric for coding strips has not been developed yet. Bettin et al. recently proposed a rubric for assessing comics students create to illustrate coding concepts [20], but they concern a case where only comic is used. It does not include design elements unique to coding strip, such as the mapping between code and comic. Thus rubric for coding strip will need to be visited. The results from this thesis provide useful insights to build on but further work is needed to format them into a rubric, test, and validate.

Design Recommendation. Some design workshop participants expressed difficulty of going through all design patterns and asked for a way to easily find the right designs to

use. This has design implication for systems that use design cards to guide users in the design process. A reasonable solution would be to implement a recommendation feature that—when activated—would inform users which cards may be appropriate for a given piece of code and then allow users to read further detail if they want to.

Ideation Workflow vs CodeToon. Design workshop participants noted that it can be difficult for beginner-level students to engage in creating coding strips right away as the ideation workflow requires users to know, for instance, which (e.g., visual) representation is appropriate for certain programming concepts; on the contrary, CodeToon participants suggested that it may be possible for students without prior programming experience may still be able to use it to learn programming as it automatically generates comics for users and they can use comics to understand code. This suggests that if someone wants to engage students in creating coding strips and students are inexperienced, CodeToon would be more appropriate than the ideation workflow.

Creative Experience. While all design workshop participants were mostly successful with creating coding strips, a particular feedback about lowering the burden of creating comics during the design process with digital authoring tool so that they can focus on the creative aspect (than the laborious task of creating comics) is worth highlighting. This suggests that even if participants are capable of creating coding strips manually, a digital authoring tool that can help lift off some burden from users are needed to scale. Moreover, some participants in the Baseline condition for the CodeToon study also found the task of creating comics laborious and expressed the challenges of figuring out how to ensure the code and comic map correctly. These suggest that tools for coding strip or layered representations where users are tasked with creating corresponding representation need to have proper support so that they can focus on the creative aspect of the design process.

Design Choices for Explicit Mapping. The findings suggest that it was right to use the line-by-line mapping for the mapping between code and comics. This adds further support to the suggestion in the literature on learning with multiple representations, which recommends making the mapping between multiple representations clear in order to maximize the learning experience and benefits. Our work shows that in the case of using code and comics together, this is, again, true. Thus the design implication here is that in future research that aims to use comics as a corresponding representation for programming languages or any other abstract programming languages, line-by-line mapping is effective and in the case alternative design choices are made, they should still focus on ensuring that the mapping between code and comic is explicit and clear.

Unified Comic Language. Several participants from the design workshop mentioned they would like to see 'unified [design] language' across coding strips, suggesting that this would make it easi'er to understand and create coding strips. The visual vocabulary we built for the auto comic generation in CodeToon was essentially this—comic language. One thing we learned from the CodeToon study, however, is that while what the workshop participants suggested is a 'unified' comic language that can be used for all programming languages, this can be challenging. For instance, for functional programming languages, it may be nice to have a visual representation such as stack in the comic style to represent function calls; arrows may need to be frequently used to visualize a function call being sent to the stack for execution. These are abstractions (actions) not used much in imperative programming languages. Of course, this is not to say that it is an impossible feat nor that it is not feasible. What we can say at this point is that we do not know enough yet. Once we have visual vocabularies for multiple programming languages (paradigms), then we may be able to compare the abstractions and potentially find universal, simple abstractions that can 'unify' the set of visual abstractions.

Design Activity. While the primary goal of our design workshop and CodeToon studies was to find an effective way to support the design and creation of coding strips, the findings spoke to the pedagogical value of the design activity itself. Many participants finding the activity enjoyable and useful for learning can have implications for research in layered representations; that is, in addition to tools that allow users to interact with the already built layered representations, it may be useful to build tools that enable users to engage in the creative process as well. While if benefits exist with another kind of layered representations is unclear, results in this thesis suggest that at least there are clear benefits with students creating comics from code or writing code from comics. Furthermore, if we regard the former as moving from abstract (code) to concrete (comic) and vice versa, it can offer interesting insights regarding layered representations. For instance, the benefits students experienced by creating comics from code may be the benefits of moving from abstract to concrete, and the same for when moving from concrete to abstract—i.e., writing code from comics.

7.3 Future Directions

In the previous section, I shared open questions and ideas that stem directly from the findings. Here, I share several ideas and directions—not mentioned in the studies—that we can explore to further advance research in this area.

Coding Strip. There are many exciting ways to expand coding strip and CodeToon. With coding strip, further research is needed to confirm its impact on learning outcome. This can include conducting A/B testing with students, from K-12 to university students. While many use cases of coding strips and their 'perceived' benefits were identified, em-

pirical studies are needed to test them. Also, the use cases we tested were with coding strips and not with CodeToon. One of the things to test also includes comparing different implementations of coding strips (e.g., 2-stage vs 3- or 4-stage progression) to understand the utility of different progressions. Moreover, we focused on using coding strips in the context of teaching programming. It remains unclear how we can apply it to other subjects within computing and other programming paradigms. For example, when learning database, students also learn languages, such as MySQL. What design patterns can we use in this context? Are there any other patterns more suitable for this subject? What would be the use cases in this scenario? How about for courses such as machine learning and artificial intelligence where code and graphical representations are frequently used to implement and visualize algorithms and neural network models? Furthermore, we saw how we can have different comic designs (e.g., 1-to-1 vs 1-to-many mapping) for the same code and concept. Since each design serves different purpose, it would be desirable to make comic in coding strip interactive so that users can switch between the designs to serve their needs. For instance, a teacher who is explaining loop might want to begin with 1-to-many mapping to present the general idea of the story and then have the comic dynamically expand the hidden panels to make it 1-to-1 mapping to explain what happens in each step. In the future, it would be great to design a system that facilitates this. Along with the curriculum, it would be interesting to test if children can still learn to program, just with comics; in other words, is it possible to learn programming without typing code and by reading coding strips? if this is possible, this can have significant impact for many communities around the world that do not have access to computers.

CodeToon. While CodeToon is developed as an authoring tool for coding strip, it functions as a visual programming environment as well. As a novel programming environment, it enables a new approach of teaching and learning programming with comics and opens up many avenues to explore. To ease its use, the next step would be to put together a curriculum (like the Creative Computing Curriculum developed for Scratch) containing a set of learning activities and guidelines on how to use CodeToon to facilitate its use. Aside from the use of comics, what separates CodeToon from other visual programming environments like Scratch is that it can host artistic activities such as drawing. Studies have shown that having students create their own visualization of algorithms helps them more compared to when simply reading visualizations created by others. Thus it would be interesting to explore how this approach of teaching programming through art compares to teaching programming through the media approach (e.g., Scratch) or more hands-on approaches such as CS Unplugged or coding with tangible programming language. One of the things CodeToon currently lacks is the support for different programming languages, such as Javascript. At the time of this writing, it supports Python and Blockly but support for other programming languages would be useful in making it more accessible. I also expect

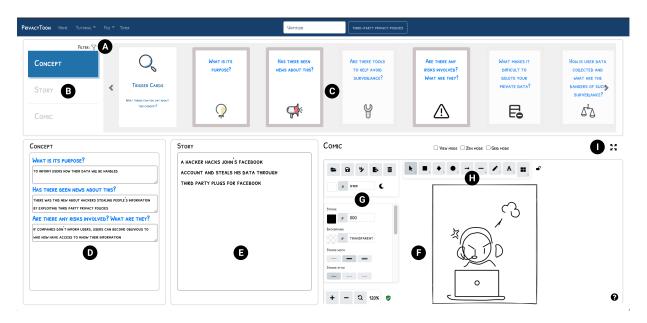


Figure 7.1: ConceptToon interface: (A) button for opening filter panel, (B) panel for selecting ideation card type (trigger/scenario/design), (C) panel for viewing and selecting trigger/scenario/design cards, (D) panel for adding ideas related to given concept/topic, (E) panel for writing stories, (F) canvas for creating comics, (G) style palette, (H) tool palette, and (I) button for maximizing comic canvas size.

that adding support for different programming languages or paradigms (e.g., imperative vs object-oriented vs functional programming) would lead us to generate a different set of visual grammars due to different syntax and conventions.

Concept-Driven Storytelling. The design process formalized in Chapter 4 represents concept-driven storytelling. Like data-driven storytelling—a process formulated by data visualization researchers to use stories to tell insights about data, concept-driven storytelling uses stories to enact, simulate, and explain concepts. One of the exciting feedbacks from the participants was the generalizability of the concept-driven storytelling process. Given the high-level nature of the process, the 3-stage design process (concept-story-comic) should work for concepts in any domain, although different ideation cards may need to be invented for the type of concepts being presented. This motivated the development of ConceptToon ¹ (Fig. 7.1) that leverages ideation cards and the scaffolding of the 3 stages. At the time of writing, the tool allows users to select programming or privacy as the domain of their concepts/topics. This selection determines which type of ideation cards shown to

¹https://privacytoon.github.io/

users; the user interface which scaffolds the 3-stage design process remains the same while only the ideation cards in Fig. 7.1 (C) change. A user study in which participants use the tool to create comics about a privacy topic has already been tested [139]. This suggests we can develop ideation cards for any domain, e.g., artificial intelligence, and enable users to create comics about any concept as long as these ideation cards are available. Hence it would be interesting to turn this tool into an open, flexible, community-driven platform where people can easily generate stories or comics about concepts in any domain and share with others. Some of the necessary features may be those that allow users to add their own set of ideation cards, import and edit existing cards, and enable collaborative activities on the platform via real-time collaboration on ideation cards, stories, and comic. This invites interesting research questions such as: how can we support collaborative authoring of ideation cards? what kind of ideation cards do people create? Many researchers and teachers already use comics to teach concepts in math, engineering, science, and more. ConceptToon can help them express and share the generated comics.

Code-Driven Storytelling. As we named the storytelling process from the design workshop concept-driven storytelling process, it may be appropriate to call the storytelling process in CodeToon code-driven storytelling. As mentioned in Chapter 5, the storytelling process in CodeToon differs from the ideation-based, concept-driven storytelling process in that it begins with *code* (expressions/syntax/conventions). It also differs from prior work that use programming (e.g., Scratch [120]) to create animation-based stories. In code-driven storytelling, code is translated into natural language and then into the visual language of comics. What is interesting about this approach is that it preserves the structure throughout these stages, making it easier for learners to connect code expressions to natural language and visual language of comics. One potential strength of this approach may be in helping novice students learn code expressions, syntax, and conventions. As this approach will likely work with any programming language, it would be interesting to explore how this process can be leveraged to facilitate the learning of new programming/computational languages.

Harnessing the Power of Generative AI Models. Exploring Human-AI partnership may open up interesting research opportunities. With generative models, we can automatically generate abstraction layers, as generative language models (e.g., GPT-3) can transform content in one language to another language. For instance, it can already produce realistic stories from code using proper instructions and examples. On request, it can already provide functional lines of code. Hence we will soon be able to teach and learn to code by partnering with AI. For instance, when explaining abstract concepts, teachers can have generative conversational AI generate and suggest relevant examples or metaphors for a given programming concept (e.g., [138]). When students have difficulty understanding code, they can request generative conversational AI to transform it into a more comprehensible representation such as diagrams or comics. Given these breakthroughs, there are many questions to explore, including but not limited to: How do we use AI to automatically generate abstraction layers? How do we design Human-AI interaction in systems capable of providing such assistance to enhance our creativity? How can generative conversational AI assist students' learning, creative, and sensemaking process in CodeToon?

Engineering Layered Representations & Supporting Tools. The lessons and insights from this research on computing education can be transferred to improving the teaching in other technical areas where they also face similar challenges. While many research tackled the abstract nature of concepts, languages, and procedures using the same or similar approaches and tools, seeing dead-level abstracting as the problem and building models and tools to support the interplay between abstraction levels can invite and contribute new research questions. For instance, how can we (automatically) transform math expressions into comics to intuitively communicate the underlying idea and procedure? How can we layer representations in math (e.g., graphs, diagrams), data science (e.g., charts), and AI (e.g., neural network architecture) to create layered representations for these areas? Also, layered representations can potentially provide rich insights, e.g., regarding users' cognitive process or level of understanding. Indeed, there can be interesting questions we can explore given that layered representations is a new kind of model that reflects our cognitive processes. For instance, if we collect data on users switching between multiple abstraction levels (e.g., which abstraction level they operated mostly in or in which order they proceed), what insights can we extract from this data? Do novice learners move from concrete to abstract, as concreteness fading and other related theories predict? Do experts move differently? Is the navigation associated with anything? If certain navigation patterns are associated with particular cognitive state, can we design a feedback loop to enable systems to intelligently provide users with an appropriate representation (abstraction)? How can we leverage crowdsourcing to build dynamic, layered representations?

Layered Representations in Virtual and Augmented Reality. In this study, we focused primarily on moving between abstraction levels inside the screen or on the design board. Building layered representations in virtual and augmented reality (VR/AR) can be another direction to explore given the growing accessibility of these mediums. In fact, they can be ideal mediums for several reasons. They can enable users to utilize the vast space. Compared to laptop screens, VR/AR offers potentially unlimited space to work with. Furthermore, interactions within VR/AR can be fluid and direct as they allow users to use their gestures and hand motions. This enables users to interact directly with representations using their hand, providing experience that cannot be replicated from

any screen. Also, given that the goal of layered representations is to enable users to gracefully move up and down the ladder of abstraction, i.e., switch between representations of varying concreteness, the ability to interact with representations, changing its shape and form directly with their hands can afford experiences more engaging and interesting than interacting over the screen.

7.4 Limitations

There are several limitations to note. First, as mentioned in Chapter 1, the scope of this thesis does not extend to proving the learning benefits. That is, we do not administer, for instance, A/B testing with a large group of students to show that students who learned using coding strips performed much better than another group of students who did not in categories related to learning (e.g., retention). The studies presented in this thesis explored the 'perceived' learning benefits from students—those with (Chapter 4,5) and without (Chapter 6) prior experience—and teachers. The qualitative evidence from them speaks to the usefulness of coding strips, tools, and use cases we present but need to be tested in the future.

7.5 Summary of Contributions

In addition to contributing to these research questions, this thesis adds to the existing body of research in CS education and creativity support in Human-Computer Interaction. In terms of CS education, this thesis makes the following contributions:

- 1. coding strip, a new pedagogical tool for teaching and learning programming,
- 2. systematic analysis of coding strips that elucidates their design dimensions and considerations,
- 3. design patterns that inform the design space for coding strips and demonstrate its ability to illustrate programming concepts, languages, and procedures,
- 4. synthesis of use cases and perceived utility of coding strips to help teachers and students interested in using coding strips use them for their teaching and learning.

In terms of creativity support, this thesis makes the following contributions:

- 1. it conceptualizes a new design process concept-driven storytelling and contributes ideation cards for creating concept-driven stories and comics;
- 2. it presents an example of creativity support tools for building layered representations;
- 3. it contributes a computational pipeline that builds on structure mapping theory, visual narrative grammar, and representations as a type of language.

Furthermore, our work on auto comic generation also extends prior work on auto comic generation by contributing a concreteness fading-based method to build a sequence of representations. Our work extends prior research in building computational pipelines in that our design is driven by the goal of creating representations that maintain invariant relations (abstract structure) across representations. This required us to leverage structure mapping theory and visual vocabulary (i.e., Fig. 5.11). This framework can support future research. Our findings on the utility of coding strips complement prior research in learning with multiple representations. Specifically, coding strip contributes an additional concreteness fading implementation and evidence on how it can offer cognitive benefits of representations.

Overall, this thesis shows coding strips as a useful pedagogical tool and comics as a useful, appropriate representation for computing. Additionally, it shows that our creativity support methods are effective at designing and creating coding strips. These studies have interesting theoretical and practical implications. Our creativity support methods, including the design process (concept \rightarrow story \rightarrow comic) and tools, used to support the mapping could potentially serve as a general framework for supporting the design of layered representations. Also, coding strip was devised to encourage moving up and down the abstraction ladder. While some prior approaches may unintentionally and/or unknowingly have this effect, this thesis makes its importance explicit and clear through coding strips. As Bret Victor puts it, "the ladder of abstraction is a technique for thinking explicitly about these levels, so a designer can move among them consciously and confidently." Indeed, making it explicit is important as it also helps our communication to be more precise. This notion that motivates coding strips can have implications on how we perceive teaching and learning in computing and beyond; it can help us recognize the critical dead-level abstracting issue present in computing and other technical disciplines such as engineering that also face similar challenges of teaching abstract concepts, languages, and procedures.

Chapter 8

Conclusion

Every self-conscious thinker must at all times be aware of — and hence be able to control — the levels of abstraction on which he is working. The capacity to shuttle between levels of abstraction, with clarity and ease, is a signal mark of the imaginative and systematic thinker.

C. Wright Mills

Today, the development of new technologies and rapid advancements in areas such as artificial intelligence are quickly outpacing our ability to keep track and learn the underlying concepts, languages, intuition, and engineering processes surrounding them. As they permeate every aspect of our lives and the tools we use, literacy in fields driving them—such as programming, data, and AI—is becoming a critical part of our training and education in the 21st century.

However, the challenge with achieving literacy in such domains is that their concepts, languages, and procedures build on many layers of abstractions. The layers of abstraction make them abstract and inaccessible. While having access to multiple abstraction levels can address this, our interfaces, systems, and instructional approaches have been designed to present information at a fixed level of abstraction—with no support for interplay, even though moving between abstraction levels is critical in gaining literacy in these technical areas.

This dissertation in large part was driven by my belief in the growing importance of such support in our education and curiosity towards what it would mean to move up and down the ladder of abstraction in the information space. While this thesis focused on coding strip, what was found in the course of this research—e.g., the mapping procedures and design principles (e.g., structure mapping theory)—provides a nice foundation to build on. By applying these insights, we can invent new forms of representations that are dynamic (transformable to other representations) and layered (from explainable/concrete forms to more abstract forms), ideal for moving between and across abstractions and abstraction levels.

References

- [1] Pixton. Accessed on January 5, 2020.
- [2] Fiona Affeldt, Daniel Meinhart, and Ingo Eilks. The use of comics in experimental instructions in a non-formal chemistry learning context. International Journal of Education in Mathematics, Science and Technology, 6(1):93–104, 2018.
- [3] Shaaron Ainsworth. Deft: A conceptual framework for considering learning with multiple representations. *Learning and instruction*, 16(3):183–198, 2006.
- [4] Shaaron Ainsworth, Peter Bibby, and David Wood. Examining the effects of different multiple representational systems in learning primary mathematics. *The Journal of the Learning Sciences*, 11(1):25–61, 2002.
- [5] Paul A Aleixo and Krystina Sumner. Memory for biopsychology material presented in comic book format. *Journal of Graphic Novels and Comics*, 8(1):79–88, 2017.
- [6] Giora Alexandron, Michal Armoni, Michal Gordon, and David Harel. Scenario-based programming: reducing the cognitive load, fostering abstract thinking. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 311– 320. ACM, 2014.
- [7] Tiago Alves, Adrian McMichael, Ana Simões, Marco Vala, Ana Paiva, and Ruth Aylett. Comics2d: Describing and creating comics from story-based applications with autonomous characters. *Proceedings of CASA*, 2007.
- [8] Yuichiro Anzai. Learning and use of representations for physics expertise. Toward a general theory of expertise: Prospects and limits, pages 64–92, 1991.
- [9] Ian Arawjo, Cheng-Yao Wang, Andrew C Myers, Erik Andersen, and François Guimbretière. Teaching programming with gamified semantics. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4911–4923. ACM, 2017.

- [10] David Paul Ausubel, Joseph Donald Novak, Helen Hanesian, et al. Educational psychology: A cognitive view, volume 6. Holt, Rinehart and Winston New York, 1968.
- [11] Steve Awodey. *Category theory*. Oxford university press, 2010.
- [12] Benjamin Bach, Natalie Kerracher, Kyle Wm. Hall, Sheelagh Carpendale, Jessie Kennedy, and Nathalie Henry Riche. Telling stories about dynamic networks with graph comics. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3670–3682, New York, NY, USA, 2016. ACM.
- [13] Benjamin Bach, Natalie Kerracher, Kyle Wm Hall, Sheelagh Carpendale, Jessie Kennedy, and Nathalie Henry Riche. Telling stories about dynamic networks with graph comics. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3670–3682, 2016.
- [14] Benjamin Bach, Nathalie Henry Riche, Sheelagh Carpendale, and Hanspeter Pfister. The emerging genre of data comics. *IEEE computer graphics and applications*, 37(3):6–13, 2017.
- [15] Benjamin Bach, Zezhong Wang, Matteo Farinella, Dave Murray-Rust, and Nathalie Henry Riche. Design patterns for data comics. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 38:1–38:12, New York, NY, USA, 2018. ACM.
- [16] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. Intl. Journal of Human–Computer Interaction, 24(6):574–594, 2008.
- [17] Jared S Bauer and Julie A Kientz. Designlibs: A scenario-based design method for ideation. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 1955–1958, 2013.
- [18] Susan Bell. Logo interpreter. Accessed on January 18, 2022.
- [19] Kirsten Berthold and Alexander Renkl. Instructional aids to support a conceptual understanding of multiple representations. *Journal of Educational Psychology*, 101(1):70, 2009.
- [20] Briana Bettin, Michelle Jarvie-Eggart, Kelly S Steelman, and Charles Wallace. Developing a comic-creation assignment and rubric for teaching and assessing algorithmic

concepts. In 2021 IEEE Frontiers in Education Conference (FIE), pages 1–5. IEEE, 2021.

- [21] Michael Bitz. The comic book project: Forging alternative pathways to literacy. Journal of Adolescent & Adult Literacy, 47(7):574–586, 2004.
- [22] Cynthia Bolton-Gary. Connecting through comics: Expanding opportunities for teaching and learning. *Online Submission*, 2012.
- [23] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting* of the American Educational Research Association, Vancouver, Canada, volume 1, page 25, 2012.
- [24] Jerome Seymour Bruner et al. Toward a theory of instruction, volume 59. Harvard University Press, 1966.
- [25] Ying Cao, Antoni B Chan, and Rynson WH Lau. Automatic stylistic manga layout. ACM Transactions on Graphics (TOG), 31(6):1–10, 2012.
- [26] Jorge Cham. Phd comics, 2020. Accessed on December 1, 2020.
- [27] Bingxin Chen, Rebecca Jablonsky, Jack Benjamin Margines, Raunaq Gupta, and Shailie Thakkar. Comic circuit: an online community for the creation and consumption of news comics. In CHI'13 Extended Abstracts on Human Factors in Computing Systems, pages 2561–2566. ACM, 2013.
- [28] Erin Cherry and Celine Latulipe. Quantifying the creativity support of digital tools through the creativity support index. ACM Transactions on Computer-Human Interaction (TOCHI), 21(4):1–25, 2014.
- [29] Sung-Bae Cho, Kyung-Joong Kim, and Keum-Sung Hwang. Generating cartoon-style summary of daily life with multimedia mobile devices. In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pages 135–144. Springer, 2007.
- [30] David Chung and Rung-Huei Liang. Understanding the usefulness of ideation tools with the grounding lenses. In Proceedings of the Third International Symposium of Chinese CHI, pages 13–22, 2015.
- [31] Neil Cohn. How to analyze visual narratives: A tutorial in visual narrative grammar.

- [32] Neil Cohn. Navigating comics: an empirical and theoretical approach to strategies of reading comic page layouts. *Frontiers in psychology*, 4:186, 2013.
- [33] Neil Cohn. The Visual Language of Comics: Introduction to the Structure and Cognition of Sequential Images. A&C Black, 2013.
- [34] Neil Cohn and Hannah Campbell. Navigating comics ii: Constraints on the reading order of comic page layouts. *Applied Cognitive Psychology*, 29(2):193–199, 2015.
- [35] Timothy R Colburn and Gary M Shute. Metaphor in computer science. Journal of Applied Logic, 6(4):526–533, 2008.
- [36] Kate Compton, Edward Melcer, and Michael Mateas. Generominos: Ideation cards for interactive generativity. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.
- [37] Stephen Cooper, Wanda Dann, and Randy Pausch. Alice: a 3-d tool for introductory programming concepts. Journal of Computing Sciences in Colleges, 15(5):107–116, 2000.
- [38] Quintin Cutts, Sarah Esper, Marlena Fecho, Stephen R Foster, and Beth Simon. The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition. In *Proceedings of the ninth annual international conference* on International computing education research, pages 63–70. ACM, 2012.
- [39] Ying Deng, Alissa N Antle, and Carman Neustaedter. Tango cards: a card-based design tool for informing the design of tangible learning games. In *Proceedings of the* 2014 conference on Designing interactive systems, pages 695–704, 2014.
- [40] Scott Derrickson, Doctor Strange. Marvel Studios, Hollywood, 2016.
- [41] SRM Derus and Ahmad Zamzuri Mohamad Ali. Difficulties in learning programming: Views of students. In 1st International Conference on Current Issues in Education (ICCIE 2012), pages 74–79, 2012.
- [42] Joseph A DeVito. Levels of abstraction in spoken and written language. Journal of Communication, 17(4):354–361, 1967.
- [43] Edsger W Dijkstra. The humble programmer. Commun. ACM, 15(10):859–866, 1972.
- [44] EW Dijkstra. Acm turing lecture 1972, 2014.

- [45] Anika Dreher, Sebastian Kuntze, and Stephen Lerman. Why use multiple representations in the mathematics classroom? views of english and german preservice teachers. *International Journal of Science and Mathematics Education*, 14(2):363–382, 2016.
- [46] Anna Eckerdal, Michael Thuné, and Anders Berglund. What does it take to learn'programming thinking'? In Proceedings of the first international workshop on Computing education research, pages 135–142. ACM, 2005.
- [47] Educatorstechnology. Teachers guide to the use of comic strips in class: Some helpful tips and resources, 2018. Accessed on January 17, 2020.
- [48] Anthony T Edwards and Anthony T Edwards. *hesiod's Ascra*. Univ of California Press, 2004.
- [49] Sarah Esper, Stephen R Foster, and William G Griswold. Codespells: embodying the metaphor of wizardry for programming. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education, pages 249–254, 2013.
- [50] Luciano Floridi. The method of levels of abstraction. Minds and machines, 18(3):303– 329, 2008.
- [51] ACM Joint Task Force. Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Technical report, Technical report, Association for Computing Machinery (ACM) IEEE Computer Society, 2013.
- [52] Robert A Fowles. Design methods in uk schools of architecture. *Design Studies*, 1(1):15–16, 1979.
- [53] Diana Franklin, Charlotte Hill, Hilary A. Dwyer, Alexandria K. Hansen, Ashley Iveland, and Danielle B. Harlow. Initialization in scratch: Seeking knowledge transfer. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 217–222, New York, NY, USA, 2016. ACM.
- [54] Emily R Fyfe, Nicole M McNeil, and Stephanie Borjas. Benefits of "concreteness fading" for children's mathematics understanding. *Learning and Instruction*, 35:104– 120, 2015.
- [55] Emily R Fyfe and Mitchell J Nathan. Making "concreteness fading" more concrete as a theory of instruction for promoting transfer. *Educational Review*, pages 1–20, 2018.

- [56] Mirta Galesic and Michael Bosnjak. Effects of questionnaire length on participation and indicators of response quality in a web survey. *Public opinion quarterly*, 73(2):349–360, 2009.
- [57] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive* science, 7(2):155–170, 1983.
- [58] Dedre Gentner. Metaphor as structure mapping: The relational shift. Child development, pages 47–59, 1988.
- [59] Arnab Ghosh, Richard Zhang, Puneet K Dokania, Oliver Wang, Alexei A Efros, Philip HS Torr, and Eli Shechtman. Interactive sketch & fill: Multiclass sketch-toimage translation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1171–1180, 2019.
- [60] Lucia MM Giraffa, Marcia Cristina Moraes, and Lorna Uden. Teaching objectoriented programming in first-year undergraduate courses supported by virtual classrooms. In *The 2nd International Workshop on Learning Technology for Education in Cloud*, pages 15–26. Springer, 2014.
- [61] Michael Evan Gold. Levels of abstraction in legal thinking. S. Ill. ULJ, 42:117, 2017.
- [62] Michael Golembewski and Mark Selby. Ideation decks: a card-based design ideation tool. In Proceedings of the 8th ACM Conference on Designing Interactive Systems, pages 89–92, 2010.
- [63] Michael J Green and Kimberly R Myers. Graphic medicine: use of comics in medical education and patient care. *Bmj*, 340:c863, 2010.
- [64] Volker Gruhn, Daniel Pieper, and Carsten Röttgers. MDA (R): Effektives Software-Engineering mit UML2(R) und EclipseTM. Springer-Verlag, 2007.
- [65] Vignir Gudmundsson. Coding is a key part in the 21st century literacy, jun 2016.
- [66] Philip J Guo. Online python tutor: embeddable web-based program visualization for cs education. In Proceeding of the 44th ACM technical symposium on Computer science education, pages 579–584. ACM, 2013.
- [67] Mark Guzdial. Computing education as a foundation for 21st century literacy. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pages 502–503. ACM, 2019.

- [68] Mark Guzdial. A new definition of computational thinking: It's the friction that we want to minimize unless it's generative, 2019.
- [69] Steven Halim, Zi Chun Koh, Victor Bo Huai Loh, and Felix Halim. Learning algorithms with unified and interactive web-based visualization. *Olympiads in Informatics*, 6, 2012.
- [70] Juris Hartmanis. Turing award lecture: On computational complexity and the nature of computer science. *Communications of the ACM*, 37(10):37–44, 1994.
- [71] Evangeline Haughney. Using comics to communicate qualitative user research findings. In CHI '08 Extended Abstracts on Human Factors in Computing Systems, CHI EA '08, pages 2209–2212, New York, NY, USA, 2008. ACM.
- [72] Samuel Ichiyé Hayakawa. Language in action. 1947.
- [73] Samuel Ichiyé Hayakawa and Alan R Hayakawa. Language in thought and action. Houghton Mifflin Harcourt, 1990.
- [74] Orit Hazzan. How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science. *Computer Science Education*, 13(2):95–122, 2003.
- [75] Orit Hazzan. Reflections on teaching abstraction and other soft ideas. ACM SIGCSE Bulletin, 40(2):40–43, 2008.
- [76] Michael Homer and James Noble. Combining tiled and textual views of code. In 2014 Second IEEE Working Conference on Software Visualization, pages 1–10. IEEE, 2014.
- [77] Richang Hong, Meng Wang, Guangda Li, Xiao-Tong Yuan, Shuicheng Yan, and Tat-Seng Chua. icomics: automatic conversion of movie into comics. In *Proceedings of the* 18th ACM international conference on Multimedia, pages 1599–1602. ACM, 2010.
- [78] Richang Hong, Xiao-Tong Yuan, Mengdi Xu, Meng Wang, Shuicheng Yan, and Tat-Seng Chua. Movie2comics: a feast of multimedia artwork. In *Proceedings of the 18th* ACM international conference on Multimedia, pages 611–614. ACM, 2010.
- [79] Jay Hosler and KB Boomer. Are comic books an effective way to engage nonmajors in learning and appreciating science? CBE—Life Sciences Education, 10(3):309–317, 2011.

- [80] Paul A Howard-Jones. Neuroscience and education: myths and messages. Nature Reviews Neuroscience, 15(12):817–824, 2014.
- [81] IDEO. IDEO Method Cards: 51 ways to inspire design. William Stout, 2003.
- [82] S.R. Jacob and Mark Warschauer. Computational thinking and literacy. Journal of Computer Science Integration, 1(1), 2018.
- [83] Benjamin D Jee and Florencia K Anggoro. Comic cognition: exploring the potential cognitive impacts of science comics. Journal of Cognitive Education and Psychology, 11(2):196–208, 2012.
- [84] Amy M Johnson, Jana Reisslein, and Martin Reisslein. Representation sequencing in computer-based engineering education. Computers & Education, 72:249–261, 2014.
- [85] Wendell Johnson. People in quandaries; the semantics of personal adjustment. 1946.
- [86] LuAnn Jordan, M David Miller, and Cecil D Mercer. The effects of concrete to semiconcrete to abstract instruction in the acquisition and retention of fraction concepts and skills. *Learning Disabilities: A Multidisciplinary Journal*, 9(3):115–22, 1999.
- [87] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference* on Human factors in computing systems, pages 1455–1464, 2007.
- [88] Michael Kölling, Neil CC Brown, and Amjad Altadmri. Frame-based editing: Easing the transition from blocks to text-based programming. In *Proceedings of the Work*shop in Primary and Secondary Computing Education, pages 29–38. ACM, 2015.
- [89] Rhonda G Kost and Joel Correa da Rosa. Impact of survey length and compensation on validity, reliability, and sample characteristics for ultrashort-, short-, and longresearch participant perception surveys. Journal of clinical and translational science, 2(1):31–37, 2018.
- [90] John Kounios and Phillip J Holcomb. Concreteness effects in semantic processing: Erp evidence supporting dual-coding theory. Journal of Experimental Psychology: Learning, Memory, and Cognition, 20(4):804, 1994.
- [91] Jeff Kramer. Is abstraction the key to computing? Communications of the ACM, 50(4):36–42, 2007.

- [92] David Kurlander, Tim Skelly, and David Salesin. Comic chat. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 225–236. ACM, 1996.
- [93] Google Creative Lab. Quickdraw. Accessed on August 14, 2021.
- [94] Michael J Lee and Andrew J Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop* on Computing education research, pages 109–116, 2011.
- [95] Ewa Luger, Lachlan Urquhart, Tom Rodden, and Michael Golembewski. Playing the legal card: Using ideation cards to raise data protection issues within the design process. In Proceedings of the 33rd Annual ACM conference on human factors in computing systems, pages 457–466, 2015.
- [96] Michelle Manno. Comics in the classroom: Teaching content with comics, 2014. Accessed on January 17, 2020.
- [97] Richard E Mayer. Cognitive theory of multimedia learning. *The Cambridge handbook* of multimedia learning, 41:31–48, 2005.
- [98] Scott McCloud. Understanding comics: The invisible art. Northampton, Mass, 1993.
- [99] Marshall McLuhan, MARSHALL AUTOR MCLUHAN, and Lewis H Lapham. Understanding media: The extensions of man. MIT press, 1994.
- [100] Timothy S McNerney. From turtles to tangible programming bricks: explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5):326–337, 2004.
- [101] Luke Moors, Andrew Luxton-Reilly, and Paul Denny. Transitioning from block-based to text-based programming languages. In 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE), pages 57–64, 2018.
- [102] Simone Mora, Francesco Gianni, and Monica Divitini. Tiles: a card-based ideation toolkit for the internet of things. In *Proceedings of the 2017 conference on designing interactive systems*, pages 587–598, 2017.
- [103] Cecily Morrison, Nicolas Villar, Anja Thieme, Zahra Ashktorab, Eloise Taysom, Oscar Salandin, Daniel Cletheroe, Greg Saul, Alan F Blackwell, Darren Edge, et al. Torino: A tangible programming language inclusive of children with visual disabilities. *Human-Computer Interaction*, 35(3):191–239, 2020.

- [104] Mitchell J Nathan, Rachaya Srisurichan, Candace Walkington, Matthew Wolfgram, Caro Williams, and Martha W Alibali. Building cohesion across representations: A mechanism for stem integration. Journal of Engineering Education, 102(1):77–116, 2013.
- [105] Cohn Neil. Visual language theory and the scientific study of comics. Empirical Comics Research, pages 305–328, 2018.
- [106] Tricia J Ngoon, Joy O Kim, and Scott Klemmer. Shöwn: Adaptive conceptual guidance aids example use in creative tasks. In *Designing Interactive Systems Conference* 2021, pages 1834–1845, 2021.
- [107] Tomohiro Nishida, Susumu Kanemune, Yukio Idosaka, Mitaro Namiki, Tim Bell, and Yasushi Kuno. A cs unplugged design pattern. ACM SIGCSE Bulletin, 41(1):231– 235, 2009.
- [108] Ministry of Education. The Ontario Curriculum Grades 10 to 12 Computer Studies. 2008.
- [109] Ana-Maria Olteţeanu, Mikkel Schöttner, and Arpit Bahety. Towards a multi-level exploration of human and computational re-representation in unified cognitive frameworks. *Frontiers in psychology*, 10:940, 2019.
- [110] The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. Computer science curricula 2013. 2013.
- [111] Allan Paivio. Coding distinctions and repetition effects in memory. *The psychology* of learning and motivation, 9:179–214, 1975.
- [112] Harold Pashler, Mark McDaniel, Doug Rohrer, and Robert Bjork. Learning styles: Concepts and evidence. *Psychological science in the public interest*, 9(3):105–119, 2008.
- [113] Leslee Francis Pelton and Timothy Pelton. The learner as teacher: Using student authored comics to "teach" mathematics concepts. In EdMedia+ Innovate Learning, pages 1591–1599. Association for the Advancement of Computing in Education (AACE), 2009.
- [114] Leslee Francis Pelton, Timothy Pelton, and Karen Moore. Learning by communicating concepts through comics. In Society for Information Technology & Teacher Education International Conference, pages 1974–1981. Association for the Advancement of Computing in Education (AACE), 2007.

- [115] K Peppler and Y Kafai. Creative coding: Programming for personal expression. *Retrieved August*, 30(2008):314, 2005.
- [116] Jean Piaget. The psychology of intelligence. Routledge, 2005.
- [117] Detlef R Prozesky. Communication and effective teaching. *Community eye health*, 13(35):44, 2000.
- [118] Jens Rasmussen. The importance of communication in teaching: A systems-theory approach to the scaffolding metaphor. *Journal of Curriculum Studies*, 33(5):569–582, 2001.
- [119] John Rawls. A theory of justice. Harvard university press, 2009.
- [120] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay S Silver, Brian Silverman, et al. Scratch: Programming for all. Commun. Acm, 52(11):60–67, 2009.
- [121] Doug Rohrer and Harold Pashler. Learning styles: Where's the evidence?. Online Submission, 46(7):634–635, 2012.
- [122] David E Rumelhart and Donald A Norman. Analogical processes in learning. Cognitive skills and their acquisition, pages 335–359, 1981.
- [123] Nazmus Saquib et al. Embodied mathematics by interactive sketching. PhD thesis, Massachusetts Institute of Technology, 2020.
- [124] Nazmus Saquib, Rubaiat Habib Kazi, Li-yi Wei, Gloria Mark, and Deb Roy. Constructing embodied algebra by sketching. In *Proceedings of the 2021 CHI Conference* on Human Factors in Computing Systems, pages 1–16, 2021.
- [125] Jabari Sellars. Comics in the classroom. https://www.gse.harvard.edu/news/uk/ 17/12/comics-classroom, 2017. Accessed on December 5, 2019.
- [126] Ariel Shamir, Michael Rubinstein, and Tomer Levinboim. Generating comics from 3d interactive computer graphics. *IEEE computer graphics and Applications*, 26(3):53– 61, 2006.
- [127] Jeremy C Short, Brandon Randolph-Seng, and Aaron F McKenny. Graphic presentation: An empirical examination of the graphic novel approach to communicate business concepts. *Business Communication Quarterly*, 76(3):273–303, 2013.

- [128] Hourya Benis Sinaceur. Facets and levels of mathematical abstraction. *Philosophia Scientiæ. Travaux d'histoire et de philosophie des sciences*, (18-1):81–112, 2014.
- [129] Socrative. https://www.socrative.com/, 2020. Accessed on April 18, 2020.
- [130] Ji Y Son, Linda B Smith, and Robert L Goldstone. Connecting instances to promote children's relational reasoning. *Journal of experimental child psychology*, 108(2):260– 277, 2011.
- [131] Warren WD Sones. The comics and instructional method. The Journal of Educational Sociology, 18(4):232–240, 1944.
- [132] DA Sousa. Recognizing and addressing mathematics difficulties: In how the brain learns mathematics (pp. 186-188), 2008.
- [133] Thomas Hvid Spangsberg. Teaching programming to non-stem novices: a didactical study of computational thinking and non-stem computing education. In Proceedings of the 17th Koli Calling International Conference on Computing Education Research, pages 201–202. ACM, 2017.
- [134] Amy N Spiegel, Julia McQuillan, Peter Halpin, Camillia Matuk, and Judy Diamond. Engaging teenagers with science through comics. *Research in science education*, 43(6):2309–2326, 2013.
- [135] David Statter and Michal Armoni. Learning abstraction in computer science: A gender perspective. In Proceedings of the 12th Workshop on Primary and Secondary Computing Education, pages 5–14. ACM, 2017.
- [136] Keith Stenning and Jon Oberlander. A cognitive theory of graphical and linguistic reasoning: Logic and implementation. *Cognitive science*, 19(1):97–140, 1995.
- [137] Sangho Suh. Using concreteness fading to model & design learning process. In Proceedings of the 2019 ACM Conference on International Computing Education Research, pages 353–354, 2019.
- [138] Sangho Suh and Pengcheng An. Leveraging generative conversational ai to develop a creative learning environment for computational thinking. In 27th International Conference on Intelligent User Interfaces, pages 73–76, 2022.
- [139] Sangho Suh, Law Edith Lamorea, Sydney, and Leah Zhang-Kennedy. Privacytoon: Concept-driven storytelling with creativity support for privacy concepts. In *Designing Interactive Systems Conference 2022*, 2022.

- [140] Sangho Suh, Celine Latulipe, Ken Jen Lee, Bernadette Cheng, and Edith Law. Using comics to introduce and reinforce programming concepts in cs1. In Proceedings of the 52nd ACM technical symposium on Computer science education, pages 585–590, 2021.
- [141] Sangho Suh, Martinet Lee, and Edith Law. How do we design for concreteness fading? survey, general framework, and design dimensions. In *Proceedings of the 19th ACM Conference on Interaction Design and Children*, 2020.
- [142] Sangho Suh, Martinet Lee, and Edith Law. How do we design for concreteness fading? survey, general framework, and design dimensions. In *Proceedings of the Interaction Design and Children Conference*, pages 581–588, 2020.
- [143] Sangho Suh, Martinet Lee, Gracie Xia, and Edith Law. Coding strip: A pedagogical tool for teaching and learning programming concepts through comics. In 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2020.
- [144] Tevita Tanielu, Raymond 'Akau'ola, Elliot Varoy, and Nasser Giacaman. Combining analogies and virtual reality for active and visual object-oriented programming. In Proceedings of the acm conference on global computing education, pages 92–98, 2019.
- [145] Mico Tatalovic. Science comics as tools for science education and communication: a brief, exploratory study. *Journal of Science Communication*, 8(4):A02, 2009.
- [146] Carly Melissa Tribull. Sequential science: A guide to communication through comics. Annals of the Entomological Society of America, 110(5):457–466, 2017.
- [147] Anthony Trory, Kate Howland, and Judith Good. Designing for concreteness fading in primary computing. In Proceedings of the 17th ACM Conference on Interaction Design and Children, pages 278–288. ACM, 2018.
- [148] Unknown. Single level of abstraction (sla), 2019.
- [149] Unknown. Comics, 2021. Accessed on December 21, 2021.
- [150] Rocco Versaci. How comic books can change the way our students see literature: One teacher's perspective. *The English Journal*, 91(2):61–67, 2001.
- [151] Bret Victor. Up and down the ladder of abstraction, 2011. Accessed on November 16, 2021.

- [152] Leslie J Waguespack Jr. Visual metaphors for teaching programming concepts. ACM SIGCSE Bulletin, 21(1):141–145, 1989.
- [153] Jane Lisa Waite, Paul Curzon, William Marsh, Sue Sentance, and Alex Hadwen-Bennett. Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. Online Submission, 2(4):14–40, 2018.
- [154] Zezhong Wang, Shunming Wang, Matteo Farinella, Dave Murray-Rust, Nathalie Henry Riche, and Benjamin Bach. Comparing effectiveness and engagement of data comics and infographics. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, page 253. ACM, 2019.
- [155] Jeannette M Wing. Computational thinking. Communications of the ACM, 49(3):33– 35, 2006.
- [156] Wired. 5 levels, 2014. Accessed on November 17, 2021.
- [157] Reading with pictures. Reading with pictures, 2020. Accessed on January 17, 2020.
- [158] Stephen Wolfram. Finally we may have a path to the fundamental theory of physics, and it's beautiful. *Stephen Wolfram's Writings*, 2020.
- [159] C Wylie and K Neeley. Learning out loud (lol): How comics can develop the communication and critical thinking abilities of engineering students. In *Proceedings of* the 2016 ASEE Annual Conference, 2016.
- [160] Haijun Xia, Ken Hinckley, Michel Pahud, Xiao Tu, and Bill Buxton. Writlarge: Ink unleashed by unified scope, action, & zoom. In CHI, pages 3227–3240, 2017.
- [161] Benjamin Xie, Greg L Nelson, and Amy Ko. An explicit strategy to scaffold novice program tracing. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, pages 344–349. ACM, 2018.
- [162] Gene Yang. Why comics belong in the classroom. https://www.youtube.com/ watch?v=0z4JqAJbxj0&feature=youtu.be, 2016. Accessed on February 10, 2020.
- [163] Xin Yang, Zongliang Ma, Letian Yu, Ying Cao, Baocai Yin, Xiaopeng Wei, Qiang Zhang, and Rynson WH Lau. Automatic comic generation with stylistic multi-page layouts and emotion-driven text balloon generation. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 17(2):1–19, 2021.

- [164] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. Penrose: from mathematical notation to beautiful diagrams. ACM Transactions on Graphics (TOG), 39(4):144–1, 2020.
- [165] VN Yulian. Developing teaching materials using comic media to enhance students' mathematical communication. In *IOP Conference Series: Materials Science and Engineering*, volume 335, page 012110. IOP Publishing, 2018.
- [166] R Zeeders. Comics-comic generation from story content graphs. PhD thesis, Master's thesis, University of Twente, Department of Electrical Engineering ..., 2010.
- [167] Jiaje Zhang and Donald A Norman. Representations in distributed cognitive tasks. Cognitive science, 18(1):87–122, 1994.

APPENDICES

Appendix A

Materials for the Formative Study

A.1 Pre-Study Survey

What is your age? (Free form text input box)

What is your gender?

- Male
- Female
- Other

What is your major? (Free form text input box)

Please specify your prior experience with programming.

- No experience
- Some experience
- Much experience

How interested are you in learning programming?

- Not Interested
- Interested to a certain degree
- Highly Interested

How difficult do you think learning programming is?

- Easy

- Manageable

- Difficult

A.2 Task Instruction

Instruction: Use the provided pen and paper to make notes.

The link to a video on recursion: https://youtu.be/pSq0G2ch91c (If the link does not work, try https://codingstrip.github.io/download).

Practice: Based on the example you saw in the previous video, solve the given examples using a provided pen and paper.

A.3 Post-Study Survey

How interested are you in learning programming compared to your interest before this task?

- More interested after this task
- Same as before this task
- Less interested after this task

How difficult was learning programming?

- Easier than expected
- As expected
- More difficult than expected

Are you satisfied with this learning experience?

- Yes
- Somewhat
- No

Do you feel that you have learned some about programming?

- Yes
- Not sure
- No

How engaging was the task?

- Very engaging
- Somewhat engaging
- Not engaging

How confident are you to agree with the following sentence, "I can explain what recursion is"?

- Extremely Confident
- Confident
- Somewhat Confident
- A little Confident
- Not at all

How confident are you to agree with the following sentence, "I can solve recursive function problems like the ones I solved"?

- Extremely Confident
- Confident
- Somewhat Confident
- A little Confident
- Not at all

How confident are you to agree with the following sentence, "From what I learned from the tasks, I can understand code like the ones I saw"?

- Extremely Confident
- Confident
- Somewhat Confident
- A little Confident
- Not at all

Any comment (Free form text input box)

Appendix B

Materials for the Design Study

This chapter provides materials used in the design workshop study (Chapter 4).

B.1 Pre-Study Survey

Below, we provide pre-& post-study surveys, design tools, and instructions.

(1) What is your age?(Free form text input box)

(2) What is your gender?

- Male
- Female
- Others
- Prefer not to answer

(3) How confident are you in drawing?

- Confident
- Somewhat confident
- Not confident

(4) How many years of teaching experience do you have?

- No experience

- 0 - 1 year - 1 - 3 years - 3 - 5 years - 5 + years

(5) On a scale of 1-5, please rate your teaching ability (1: Poor; 5: Excellent).

(5) How many years of teaching experience do you have on computer science (e.g., programming)?

- No experience
- 0 1 year
- 1 3 years
- 3 5 years
- 5 + years

(6) How confident are you with designing comics for teaching coding concepts?

- Confident
- Somewhat confident
- Not confident

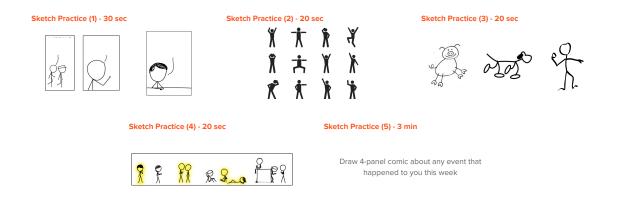
(7) On a scale of 1-5, please indicate your thoughts on how useful you think teaching/learning with comics is.

- 1 (Not useful)
- 2
- 3
- 4
- 5 (Useful)

B.2 Task Intruction

The slides used to explain the task is available at https://docs.google.com/presentation/ d/132_KCo4knCUDEBDYQHFkbEm9S9VTRvIwRbYuqO-fkFo/edit?usp=sharing (If the link does not work, try https://codingstrip.github.io/download).

B.2.1 Sketch Warm-Up





B.2.2 Translate Story to Code

Story to Code: Cinderella (1)		Story to Code: Cinderella (2)		Story to Code: Cinderella (3)	
 At the ball, Cinderella had to check the time constantly when dencing with Prince. She had to leave before 12:00 AM. 	while (time < 12:00 AM) (cinderella.dance();) cinderella.Jeave();	 Before Cinderella went to the ball, Godmother (fairy) changed Cinderella's whole outfit. 	GodmothettransformCinderella(): Cinderella.clothes = beautiful dress; Cinderella.ahoes = glass high heels;	 Prince went around the whole kingdom, checking the feet of every girl until he found Cinderelia whose feet fit the shoes perfectly! 	Prince_Find_Cinderella: while (FitShoes(girt/feet) — false)(Girl — Next_Girl); } print("Found Cinderellal");
	Concept: Conditional loop		Concept: Variable assignment		Concept: Conditional loop

Figure B.2: Translate story to code

B.3 Discussion Questions

- 1. "Would it have any effect on the classroom atmosphere? If yes, what would they be?"
- 2. "Would it have any effect on the learning? If yes, what would they be?"
- 3. "Would it have any effect on the time it takes to learn/teach? If yes, how would it affect?"

- 4. "Do you have any suggestions on how coding strips may be used for learning or teaching?"
- 5. "Would you suggest any ideation or design patterns to be added or removed?"
- 6. "What patterns did you find useful?"
- 7. "Would you suggest any modifications to the design process?"

B.4 Post-Study Survey

(1) Tell us about your experience.

Instruction Clarity

"It was easy to follow the instruction" 1 (Strongly disagree), 2, 3, 4, 5 (Strongly agree)

Engagement

"It kept me engaged"

1 (Strongly disagree), 2, 3, 4, 5 (Strongly agree)

(2) How confident are you with designing comics for teaching coding concepts compared to before this task?

- More confident after this task
- Same as before this task
- Less confident after this task

(3) How useful do you think using comics to teach is compared to before this task?

- More useful after this task
- Same as before this task
- Less useful after this task

(4) How useful do you think using comics to learn is compared to before this task?

- More useful after this task

- Same as before this task

- Less useful after this task

(5) On a scale of 1-5, please indicate your thoughts on how useful the design process is.

- 1 (Not useful)
- 2
- 3
- 4
- 5 (Useful)

(6) On a scale of 1-5, please indicate your thoughts on how useful teaching with comics is. - 1 (Not useful)

- 2
- 3
- 4
- 5 (Useful)

(7) On a scale of 1-5, please indicate your thoughts on how useful learning with comics is. - 1 (Not useful)

- 2
- 3
- 4
- 5 (Useful)

(8) Would you like to learn using coding strips?

- Yes
- Not sure
- No

(9) Could you elaborate on your response to question 8? (Free form text input box)

(10) Would you like to teach using coding strips?

- Yes
- Not sure
- No

(11) Could you elaborate on your response to question 10? (Free form text input box)

(10) Do you have additional ideas on how coding strips may be used in classrooms or other situations?(Free form text input box)

(11) If you have any last comments, you can write here. (Free form text input box)

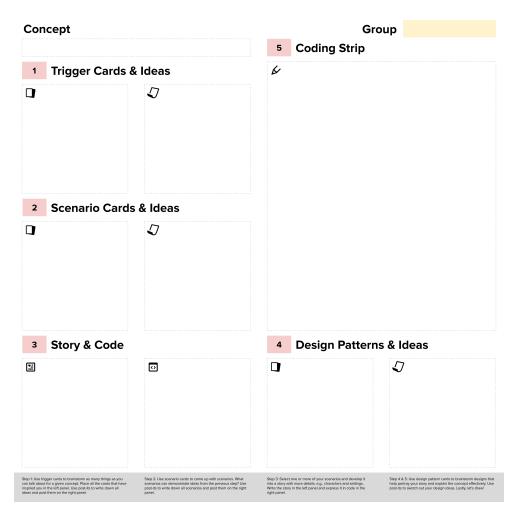


Figure B.3: Design board used in the workshops.

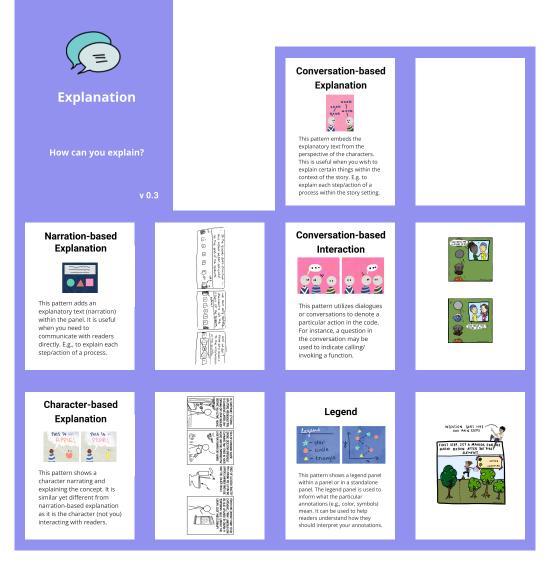


Figure B.4: Design cards for explanation theme. Number of times each card was used in design workshops: Narration-based Explanation: 4/18; Conversation-based Explanation: 4/18; Conversation-based Interaction: 4/18; Character-based Explanation: 5/18; Legend: 1/18.

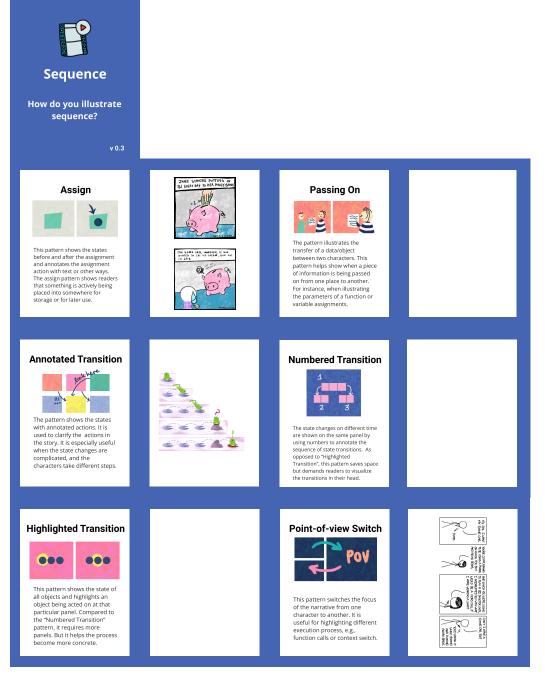


Figure B.5: Design cards for sequence theme. Number of times each card was used in design workshops: Assign: 6/19; Passing On: 3/19; Annotated Transition: 4/19; Numbered Transition: 3/19; Highlighted Transition: 3;/19 Point-of-View Switch: 0/19

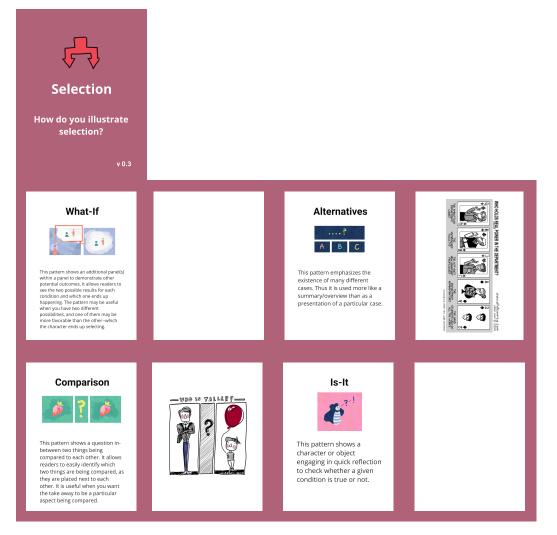


Figure B.6: Design cards for selection theme. Number of times each card was used in design workshops: What-If: 6/16; Alternatives: 3/16; Comparison: 2/16; Is-It: 5/16.

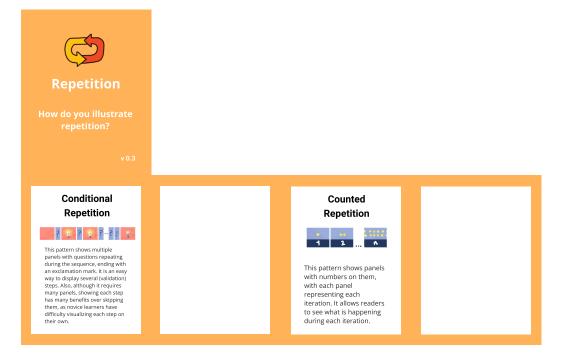


Figure B.7: Design cards for repetition theme. Number of times each card was used in design workshops: Conditional Repetition: 5/8; Counted Repetition: 3/8.



Figure B.8: Design cards (1/2) for presentation theme. Number of times each card was used in design workshops: Moments: 1/27; Visualization: 3/27; Concretization: 0/27; Abstraction: 2/27; Flowchart: 5/27; Zoom: 1/27; Grouping: 0/27.



Figure B.9: Design cards (2/2) for presentation theme. Number of times each card was used in design workshops: Multiple Scenarios: 6/27; Gradual Reveal: 2/27; Question&Answer: 2/27; Lens: 0/27; Break-the-fourth-wall: 0/27; Visualization: 3/27.

Appendix C

Materials for the CodeToon Study

This chapter presents materials used for the two-part studies for the CodeToon study.

C.1 Part I. User Study

To download comics created from this user study, go to https://codetoon-research.github.io/download/ (If the link does not work, try https://codingstrip.github.io/download).

C.1.1 Pre-Study Survey

What is your age? (Free form text input box)

What is your gender?

- Male
- Female
- Non-binary / third gender
- Prefer not to specify

What is your major? (Free form text input box) Please specify your prior experience with programming.

- No experience
- Some experience
- Much experience

How frequently do you write code?

- Never
- Once every few years
- Once a year
- Once every few months
- Once a month
- Once a week
- Daily

Please specify how likely you are to agree with the following sentences.

"I frequently engage in artistic activities (e.g., doodling, sketching, drawing, designing)"

- Strongly disagree
- Disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Agree
- Strongly agree

"I enjoy drawing"

- Strongly disagree
- Disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Agree
- Strongly agree

How frequently do you draw?

- Never

- Once every few years
- Once a year
- Once every few months
- Once a month
- Once a week
- Daily

How confident are you in drawing?

- Confident
- Somewhat confident
- Not confident

Please specify your prior experience with digital drawing tools? (i.e., Adobe, Canva, Paint, etc.)

- No experience
- Some experience
- Much experience

Please specify your prior experience with drawing comics.

- No experience
- Some experience
- Much experience

Please specify how likely you are to agree with the following sentences.

- "I like comic books."
- Strongly disagree
- Disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Agree
- Strongly agree

"I have created comic books before."

- Strongly disagree
- Disagree

- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Agree
- Strongly agree

"I enjoy reading comic books."

- Strongly disagree
- Disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Agree
- Strongly agree

How frequently do you read comic books?

- Never
- Once every few years
- Once a year
- Once every few months
- Once a month
- Once a week
- Daily

How confident are you with creating comics for teaching programming concepts?

- Confident
- Somewhat confident
- Not confident

How many years of teaching experience do you have?

- No experience
- 0 1 year
- 1 3 years
- 3 5 years
- 5+ years

On a scale of 1-5, please rate your teaching ability.

- 1 (Poor) - 2 - 3 - 4 - 5 (Excellent)

How many years of teaching experience do you have in computer science (e.g., programming)?

- No experience
- 0 1 year
- 1 3 years
- 3 5 years
- 5+ years

On a scale of 1-5, please indicate your thoughts on how useful you think teaching/learning with comics is.

- 1 (Not useful) - 2 - 3 - 4 - 5 (Useful)

C.1.2 Post-Study Survey with CSI

Part 1. CSI questions

Please rate your agreement with the following statements (1 as Highly Disagree, 10 as Highly Agree):

1. What I was able to produce was worth the effort I had to exert to produce it.

- 2. I enjoyed using the tool.
- 3. The tool was helpful in allowing me to track different ideas, outcomes, or possibilities.

4. I was able to be very creative while doing the activity inside this tool.

5. My attention was fully tuned to the activity, and I forgot about the tool that I was using. [Engagement] (i.e., the tool was easy enough it did not prevent me from being engaged) 6. I became so absorbed in the activity that I forgot about the tool that I was using. [Immersion] (i.e., the tool was easy enough that it did not prevent me from being immersed in the activity)

7. The tool allowed me to be very expressive.

8. It was easy for me to explore many different ideas, options, designs, or outcomes, using this tool.

9. I would be happy to use this tool on a regular basis.

10. I was satisfied with what I got out of the tool.

Part 2. Post-Study Survey

How would you rate the difficulty level of creating comics about programming concepts with this authoring tool?

- Extremely difficult
- Very difficult
- Moderately difficult
- Slightly difficult
- Not at all difficult

How useful was this authoring tool in creating comics about programming concepts?

- Extremely useful
- Very useful
- Moderately useful
- Slightly useful
- Not at all useful

How useful were object/idea suggestions (i.e., dropdown in the story section) for creating stories?

- Extremely useful
- Very useful
- Moderately useful
- Slightly useful
- Not at all useful

How useful was the automatic comic generation for this task?

- Extremely useful

- Very useful
- Moderately useful
- Slightly useful
- Not at all useful

How accurately did the automatically generated comics map to code?

- Extremely accurate
- Very accurate
- Moderately accurate
- Slightly accurate
- Not at all accurate

How accurately did the automatically generated comics represent ..

... code execution (i.e., what's happening in code)?

- Extremely accurate
- Very accurate
- Moderately accurate
- Slightly accurate
- Not at all accurate

... code semantics (i.e., meaning assigned to code)?

- Extremely accurate
- Very accurate
- Moderately accurate
- Slightly accurate
- Not at all accurate

How confident are you with creating comics about programming concepts compared to before this task?

- More confident after this task
- Same as before this task
- Less confident after this task

How useful do you think this tool can be for ...

- ... **teaching** programming?
- ... **learning** programming?

... novice learners (beginners) in programming?

How useful do you think using comics to learn/teach is compared to before this task?

- More useful after this task
- Same as before this task
- Less useful after this task

Would you use this tool for teaching programming?

- Yes
- Maybe
- No

Do you think you would have enjoyed learning programming with comics created with this tool (i.e., when you were learning programming for the first time)?

- Yes
- Maybe
- No

Any further comments? (Free form text input box)

C.1.3 System Usability Scale

Please rate the following statements.

- Strongly disagree
- Disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Agree
- Strongly agree

"I think that I would like to use this comic authoring tool frequently."

"I found this comic authoring tool unnecessarily complex."

"I thought this comic authoring tool was easy to use."

"I think that I would need the support of a technical person (even after going through tutorial/practice tasks) to be able to use this comic authoring tool."

"I found the various functions in this comic authoring tool were well integrated."

"I thought there was too much inconsistency in this comic authoring tool."

"I would imagine that most people would learn to use this comic authoring tool very quickly."

"I found this comic authoring tool very cumbersome to use."

"I felt very confident using this comic authoring tool."

"I needed to learn a lot of things before I could get going with this comic authoring tool."

C.1.4 Paired Factor Comparison

"When doing this task, it's most important that I'm able to..."

- Work with people
- Become immersed in the activity

"When doing this task, it's most important that I'm able to..."

- Be creative and expressive
- Enjoy using the system or tool

"When doing this task, it's most important that I'm able to..."

- Work with other people
- Explore many different ideas, outcomes, or possibilities

"When doing this task, it's most important that I'm able to..."

- Produce results that are worth the effort I put in
- Become immersed in the activity

"When doing this task, it's most important that I'm able to..."

- Explore many different ideas, outcomes, or possibilities
- Produce results that are worth the effort I put in

"When doing this task, it's most important that I'm able to..."

- Be creative and expressive

- Enjoy using the system or tool

"When doing this task, it's most important that I'm able to..."

- Be creative and expressive
- Explore many different ideas, outcomes, or possibilities

"When doing this task, it's most important that I'm able to..."

- Enjoy using the system or tool
- Produce results that are worth the effort I put in

"When doing this task, it's most important that I'm able to..."

- Become immersed in the activity
- Explore many different ideas, outcomes, or possibilities

"When doing this task, it's most important that I'm able to..."

- Explore many different ideas, outcomes, or possibilities
- Enjoy using the system or tool

"When doing this task, it's most important that I'm able to..."

- Enjoy using the system or tool
- Work with other people

"When doing this task, it's most important that I'm able to..."

- Be creative and expressive
- Produce results that are worth the effort I put in

"When doing this task, it's most important that I'm able to..."

- Enjoy using the system or tool
- Become immersed in the activity

"When doing this task, it's most important that I'm able to..."

- Produce results that are worth the effort I put in
- Work with other people

"When doing this task, it's most important that I'm able to..."

- Become immersed in the activity

- Be creative and expressive

C.2 Part II. Comic Evaluation Study

C.2.1 Survey

The survey was administered using Qualtrics. Responses to all the questions below were required. Participants could not move to the next page without answering the questions.

Page 1. Requirement

I can read and understand basic Python code.

- Yes

- No

If the above answer is "No," please STOP now and inform researcher (sangho.suh@uwaterloo.ca) to WITHDRAW from the study.

Page 2. Demographic

What is your age? (Free-form text input box)

What is your major? (Free-form text input box)

What is your prior experience with programming?

- [Pro] I'm a pro

- [Semi-pro] I'm pretty comfortable with coding
- [Amateur] I can do basic coding
- [Semi-Amateur] I would need some help with coding
- [Beginner] I would need a lot of help with coding

How frequently do you write code?

- Never
- Once every few years
- Once a year
- Once every few months
- Once a month
- Once a week
- Daily

How many years of teaching experience (e.g., TA) do you have in computer science (e.g., programming)?

- No experience
- 0 1 year
- 1 3 years
- 3 5 years
- 5+ years

On a scale of 1-5, please indicate your thoughts on how useful you think comics is for teaching programming

- 1 (Not useful) - 2 - 3 - 4
- 5 (Useful)

```
... learning programming - 1 (Not useful)
- 2
- 3
- 4
- 5 (Useful)
```

Page 3. Task Explanation

Now, you'll see comics and their corresponding code.

Please read them carefully for evaluation.

Page 4-11. Comic Evaluation Task

Note that questions below form a set of questions asked for each of the 8 coding strips. '{{ }}' below represents placeholder for any one of the 8 coding strips as the order was randomized.

Programming Concept: {{ Concept }}

 $\{\{ Comic \}\}$

This comic is based on the following code.

 $\{\{ Comic \}\}$

Overall, how **accurately** does the comic **map** to code?

- Extremely accurate
- Very accurate
- Moderately accurate
- Slightly accurate
- Not at all accurate

How accurately does the comic represent ..

- ... code execution (i.e., what's happening in code)?
- Not at all accurate
- Slightly accurate
- Moderately accurate
- Very accurate
- Extremely accurate

... code semantics (i.e., meaning assigned to code)?

- Not at all accurate
- Slightly accurate
- Moderately accurate
- Very accurate
- Extremely accurate

How well does the comic illustrate (visually represent) \ldots

... code?

- Not at all accurate

- Slightly accurate
- Moderately accurate
- Very accurate
- Extremely accurate

... "{{ concept }}"?

- Not at all accurate
- Slightly accurate
- Moderately accurate
- Very accurate
- Extremely accurate

How **useful** is this comic for ..

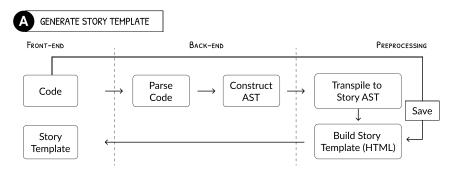
- ... **learning** "{{ concept }}"?
- Not at all accurate
- Slightly accurate
- Moderately accurate
- Very accurate
- Extremely accurate

 \dots teaching "{{ concept }}"?

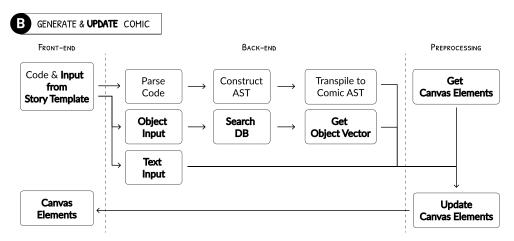
- Not at all accurate
- Slightly accurate
- Moderately accurate
- Very accurate
- Extremely accurate

C.3 CodeToon Implementation Details

CodeToon (Fig. 5.2) is implemented using Bootstrap and Django. The drawing canvas is implemented using open-source tool, Excalidraw. CodeToon users can use either touch or stylus (e.g., Apple pencil, Wacom stylus) for drawing and editing comics on the canvas. Below, we explain in detail how the two main functionalities in CodeToon, (1) generating story from code and (2) adding & updating comic from story, are implemented.



(a) Pipeline for generating story template. When a user clicks 'generate story' button (Fig. 5.2C), the code is sent to the back-end and parsed into an abstract syntax tree (AST) in JSON. To build a story template, the program traverses through the code AST (JSON) recursively. As it traverses, it checks node types (e.g., <Assign>) and extracts relevant information to build the story template, which consists mostly of <input> HTML tag. Once the story template has been generated, it is sent to the front-end to be rendered in the story section (Fig. 5.2D).



(b) Pipeline for generating and updating comic from story. When a user clicks the 'generate comic' button (Fig. 5.5), CodeToon collects (1) code that was used to generate the story template and (2) values in the story template and makes an Ajax request. In the back-end, the program parses code and turns it into comic AST, which is essentially an instruction of how comic should be presented graphically. At the same time, the back-end checks whether any text made reference to the object in the object database. If there is a match, the back-end returns the vector information so they can be added to the comic. Once these information come together, they are sent to the front-end to generate comic. As for updating the comic, when a user clicks the 'update comic' button, it follows the same procedure (highlighted in bold), except that code is not sent to the back-end, because we are not generating a new comic.

Figure C.1: Implementation of computational pipeline for (a) generating story template and (b) generating and updating comic from story.

Appendix D

Materials for the In-Class Study

We administered a survey at the end of the class. There was no pre-study survey.

D.1 Post-Study Survey

The following set of questions examines your relationship with programming before and after the course.

Before taking this course, what was your prior experience with programming?

- No experience
- Some (several hours/days)
- Much (several weeks/months)

Before taking this course, how difficult did you think learning programming is?- Easy

- Manageable
- Difficult

Before taking this course, how interested were you in learning programming?

- Not interested
- Interested to a certain degree
- Highly interested

After taking this course, how interested are you in learning programming compared to your interest before taking this course?

- More interested after this course
- Same as before this course
- Less interested after this course

After taking this course, how difficult do you think learning programming is when compared to before taking this course?

- Easier than expected
- As expected
- More difficult than expected

After taking this course, do you feel that you have learned some about programming? - Yes

- Not sure
- No

Evaluating Use Cases

The following set of questions examines your experience with comics in the course.

This term, comics were used in 4 different ways.

First, they were used to INTRODUCE concepts, such as ...

... variables and how their values can change

(Comic)

... while loop and how while loop repeats as long as the condition is true

(Comic)

... array and how array index represents offset (distance from the first position) (Comic)

Second, they were used in clicker questions to REVIEW the learned concepts, as shown below.

(Comic)

(Comic)

(Comic)

Third, they were used to help you PRACTICE writing code by asking you to translate the following scenarios into code.

How would you express in code the following scenarios, such as ...

- ... "eat cap'n crunch for 100 days" challenge?
- ... Bob Lean's daily schedule depending on his sleep time?
- ... a child looking for Wifi to send a snap?

Fourth, they were used to INTRODUCE code (first show comics and then map them to code).

For instance, ...

... putting a dollar bill to a piggy bank was mapped to the variable assignment code.

... the "endless looped time" in Dr. Strange was mapped to the while loop code.

Based on your experience this term, how do you feel about learning programming using comics?

1 (Really Dislike), 2, 3, 4, 5, 6, 7 (Really Like)

Please provide the reason(s) for your response. (Free form text input box)

Did you find the comics useful for learning? Did they make concepts easier to understand? Did they make learning programming more fun? Did they make learning programming more motivating? Did they make learning programming more engaging? Did they have positive impact on your perceived difficulty of programming? Did they have positive impact on your interest in learning programming? - Not at all

- Slightly
- Moderately
- Very
- Extremely

Please evaluate the following use case.

Use Case #1: Introduce concept

How do you feel about being introduced to concepts with comics?

1 (Really Dislike), 2, 3, 4, 5, 6, 7 (Really Like)

Please provide the reason(s) for your response.

Did you find being introduced to concepts with comics useful for learning? Did it make concepts easier to understand? Did it make learning programming more fun? Did it make learning programming more motivating? Did it make learning programming more engaging? Did it have positive impact on your perceived difficulty of learning programming? Did it have positive impact on your interest in learning programming?

Please evaluate the following use case.

Use Case 2: "REVIEWING" the concepts.

How do you feel about reviewing the learned concepts with comics?

1 (Really Dislike), 2, 3, 4, 5, 6, 7 (Really Like)

Please provide the reason(s) for your response.

Did you find reviewing the learned concepts with comics useful for learning? Did it make learning programming more fun? Did it make learning programming more motivating? Did it make learning programming more engaging? Did it have positive impact on your perceived difficulty of learning programming? Did it have positive impact on your interest in learning programming?

Please evaluate the following use case.

Use Case #3: "PRACTICING" writing code by translating comics to code.

How do you feel about practicing writing code using comics (i.e., translating comics into code)?

1 (Really Dislike), 2, 3, 4, 5, 6, 7 (Really Like)

Please provide the reason(s) for your response. (Free form text input box)

Did you find practicing writing code using comics useful for learning?

Did it make learning programming more fun?

Did it make learning programming more motivating?

Did it make learning programming more engaging?

Did it have positive impact on your confidence in writing code?

Did it have positive impact on your perceived difficulty of learning programming? Did it have positive impact on your interest in learning programming?

1 (Really Dislike), 2, 3, 4, 5, 6, 7 (Really Like)

Please evaluate the following use case.

Use Case 4: "INTRODUCING" code (first show comics and then map it to code).

... the "endless looped time" in Dr. Strange maps to the while loop code

How do you feel about being introduced to code with comics (see comics and then code in the context of comics scenario)?

1 (Really Dislike), 2, 3, 4, 5, 6, 7 (Really Like) Please provide the reason(s) for your response. (Free form text input box)

Did you find being introduced to code with comics useful for learning? Did it make code easier to learn? Did it make code less frightening? Did it make learning programming more fun? Did it make learning programming more engaging? Did it make learning programming more engaging? Did it have positive impact on your perceived difficulty of learning programming? Did it have positive impact on your interest in learning programming?

Would you recommend that other instructors in computing courses use comics?

- Yes

- Maybe

- No

Please elaborate. (Free form text input box)

Any last comments? (Free form text input box)

Is this your first time taking CS105? - Yes, 1st time

- No, 2nd time

What is your major? (Free form text input box)

Which year are you in?

- 1st year

- 2nd year

- 3rd year

- 4th year

What is your reason(s) for taking this course?

- It is a degree requirement
- Interest in learning programming

What is your gender?

- Male
- Female
- Other
- Prefer not to answer

What was your attendance like this term?

- -0 25% (attended 0 5 lectures)
- 21 50% (attended 6 11 lectures)
- 51 75% (attended 11 15 lectures)
- 76 100% (attended 16 22 lectures)

Glossary

- abstraction ladder A term S. I. Hayakawa proposed to describe levels of abstraction. xiv, 4, 5, 7, 102, 113
- concreteness fading An evidence-based instructional technique for teaching abstract concept by introducing it in three stages using representations with decreasing levels of concreteness. xiv, xv, 17–19, 29, 30, 33
- data comics A form of comics that communicates insights in data with the visual language of comics. 23, 24, 38, 46
- data-driven storytelling A type of storytelling that tries to communicate insights about data. 109
- dead-level abstracting A linguistic phenomenon where a speaker is stuck at certain levels of abstraction v, 5, 102, 111, 113
- ladder of abstraction A term S. I. Hayakawa proposed to describe levels of abstraction. v, 15, 17, 20, 21, 102, 112, 113, 115
- levels of abstraction The amount of complexity or detail by which a system is viewed, conceptualized, or programmed. v, xiv, 2, 3, 5, 6, 15, 19, 29, 85, 114
- structure mapping theory A theory of analogical reasoning, developed by Dedre Gentner. 17, 20, 83, 104, 113, 115
- visual narrative grammar A theory proposed by Neil Cohn that posits that each panel of the comic can be categorized as one of the narrative categories. 49, 83, 113

Abbreviations

CSI Creativity Support Index 26, 27, 68, 70, 72, 80, 104

ECS External Compositional Structure 23

 \mathbf{PFC} Paired Factor Comparison 68

SUS System Usability Scale 68, 69

VNG Visual Narrative Grammar 23, 63, 64

Nomenclature

- **code-driven storytelling** A type of storytelling that uses code as the story structure 80, 110
- coding strip A form of comic strip accompanied by its corresponding code v, vi, xi, xiv, xv, xvii, xix, 1, 7, 10, 12–16, 19, 25, 29, 34–37, 39–47, 49, 53–55, 57, 58, 66, 68, 70, 71, 75, 81, 83, 85, 86, 89, 90, 95, 98–108, 112, 113, 115
- concept-driven storytelling A type of storytelling that aims to create stories from a concept 13, 109, 110, 113
- layered representations A conceptual model with two or more layers of representations sharing the same invariant relations (abstract structure) 7, 10, 13, 15–17, 19, 106, 107, 111–113