RESPECT the RECIPE: What Happened When HBCU Students New to Java Programming Were Given a Recipe to Code

Leshell Hatley, Ph.D.
Mathematics & Computer Science Department
Coppin State University
Baltimore, MD, USA
lhatley@coppin.edu

Abstract— **There are extremely low rates of participation by underrepresented groups (e.g. African American, Latino American, and Native American) in Computer Science. Learning to program (design and issue sequential commands to instruct a computer to solve a problem) can be challenging for many reasons and is often referred to as a large barrier to entry into the field. Novice programmers focus simultaneously on learning how to design a program, converting that design into efficient working code, and communicating their conceptual ideas towards completion. Design Recipes have been used to counter these challenges, giving beginning programmers instructional steps towards designing, programming, and testing their code. This report shares the experience of what happened when novice programmers at an HBCU (Historically Black College/University) were given a customized Design Recipe. It describes improvements regarding students' ability to design, code, test, and communicate their ideas as well as benefits related to how these outcomes can be assessed. The results of this experience contribute teaching and learning approaches towards the success of underrepresented students in programming and computer science.**

Keywords—Computer Science Education, Beginning Programmers, Design Recipe, Historically Black College/University, Assessment and Feedback, Broadening Participation

I. Introduction

There are extremely low rates of participation by underrepresented groups (e.g. African American, Latino American, and Native American) in Computer Science at all levels of education – K-12, undergraduate, masters, PhD – as well as beyond the classroom, within industry and the federal government [27].

"Computer programmers [are computer scientists who] write code to create software programs. They turn the program designs created by software developers and engineers into instructions that a computer can follow" (Bureau of Labor Statistics, U.S. Department of Labor, 2014 -15 Edition, para. 1). When learning how to program, a novice computer programmer focuses on both of these functions: 1) design the program and 2) write the code/program (computer instructions) needed to create the desired program to accomplish a desired outcome. They are then faced with having to share and explain their idea.

For the past six decades, many computer science instructors and education researchers report that learning to program is challenging [1, 4, 8, 9, 11, 21, 22, 23, 25]. The above cited researchers and others have identified some perceived barriers to entry when learning how to program. They include, but are not limited to:

- the difficulty of understanding the purpose of programs and their relationship with the computer [23];
- difficulty in grasping the syntax and semantics of a programming language [23];
- misconceptions of programming constructs [26];
- inability to problem-solve [15];
- inability to read and understand program code [13, 14]; and
- motivation (or lack thereof) [4].

This wide recognition and acknowledgment that students just "*don't get it*" led to many tools and approaches for teaching and learning computer programming [4, 6, 10, 12, 14, 21]. As such, introduction to programming courses have been continuously redeveloped with changes in language, paradigm, and swapping between breadth and depth of content [4].

This experience report shares the experiences of teaching and learning how to program in the CS0, CS1, CS2 (entry level programming courses) at a HBCU in the northeastern part of the United States. The experience includes the ongoing research and development of a customized design recipe (i.e. guide) for novice programmers as they begin to learn how to code and the resulting lessons learned.

II. Design Recipe for Computer Programming

A. Origins

Introduced by [4], a design recipe guides a beginning programmer through the entire problem-solving process. The authors described the steps toward the proper design of a program as follows:

- "Analyze a problem statement, typically stated as a word problem;
- Express its essence, abstractly and with examples;
- Formulate statements and comments in a precise language;
- Evaluate and revise these activities considering checks and tests; and
- Pay attention to details."

From this, a six-step "design recipe" checklist was born [4]:

978-1-5386-1363-4/18/$31.00 ©2018 IEEE

- STEP 1: Problem Analysis & Data Definition – describe the collections of data that the program must consume and produce
- STEP 2: Contract, Purpose & Effect Statements, Header – formulate the task is student's own words
- STEP 3: Examples – create small functional examples that illustrates what the program computers from given data
- STEP 4: Template – construct a template for the program's organization
- Step 5: Code – write the program using desired programming language
- Step 6: Test – turn the examples from STEP THREE into functional tests

B. Reported Uses

Since its inception, the design recipe has been used in various forms within several learning environments to provide guidance to novice programmers. [5, 7, 15, 19, 20, 24] are a few examples.

III. Motivation for a More *Customized Design recipe*

Given the perceived barriers described above and more observed in the classrooms at this HBCU, students in the CS0, CS1, and CS2 courses often have trouble knowing where to start when writing a program, which steps they should follow, and often even have trouble with gathering the confidence to begin. In response to these observations, a step-by-step guide was produced to be shared at the beginning of each course. Using design-based research methodology, this guide has continuously improved and is now an interactive template accompanied with guiding principles and encouragement. Going forward, this guide is called *this customized design recipe*. It is still a work-in-progress and is enhanced after each major assignment/project as a result of student feedback. It is now used by students in the intended courses (CS0, CS1, CS2) as well as for various informal learning experiences within the Math and Computer Science Department at this HBCU, including but not limited to coding competitions, hackathons, and undergraduate student research and development projects. The current version can be found at http://www.coppincs.com/designrecipe.

A. Student Knowledge and Classroom Context

Each semester, computer science majors, minors, and non-majors enroll in CS0, CS1, and CS2 courses with no programming experience (CS0), and whatever experience they may have was usually obtained from the previous course (CS1, CS2). CS0 introduces students to the Scratch, Python, and Java programming languages and typically has about 20 students enrolled, most of which are male. CS1 and CS2 are much smaller and focus primarily on the Java programming language. For the past several semesters (at the time of this writing), ninety-two percent of students entering CS0 reported that they never programmed before and many admitted not knowing much about how computers work. Even more, students lacked strong problem-solving skills, could not think algorithmically, and between semesters and during summer breaks from school, students often do not practice programming and/or rarely have enough knowledge to obtain (or do well while participating in) internships. This sometimes results in forgetting a great deal of what was learned in the previous course and requires a great deal of review at the start of each semester.

IV. Methodology

This customized design recipe was developed using Design-Based Research (DBR). DBR is an educational research approach that engages research in an iterative process of design, implementation, and evaluation cycles towards the creation and/or refinement of learning environments and the theories that are reflected with them [3,18]. It seeks to simultaneously understand the processes of learning, enhance learning outcomes, and contribute to theories of learning, all through the creation of instructional interventions (e.g. materials, strategies, software, professional development, etc.) [2]. Even more, this iterative process happens in real-world learning environments, usually in collaboration with instructors - resulting in practical applications of learning interventions, rather than in a laboratory or under controlled conditions [18].

V. Experience

A. Creating & Introducing the Design Recipe

The initial attempt at teaching the program development process included sharing the *Program Development Process* diagram found in [16] at the beginning of the course/semester. However, the terms and descriptions used in this diagram were unfamiliar therefore difficult to follow and remember. It also only explained the process of programming, presuming the programming challenge had already been solved following this process helped turn that solution into code. It did not explain or give hints to how to solve the program or how to think algorithmically or computationally (i.e. how to express solving problems using instructions that a computer should follow instead of the way humans do), which is a skill many novice programming students lack. This lead to the design and sharing of a completely customized guide known as the *Template for Problem-Solving and Program Design*. The *Program Development Process* was used after this *template* was followed and program design was complete. This template included 9 steps and included steps for visualizing final product, listing actions the program needed to perform to accomplish its goal, choosing variable and other identifier names, progressively pseudocode and then turning that into code using the desired programming languages, and for how, where, and when to insert comments. It was also shared at the beginning of each course/semester as a reference. This template provided more help to students than the previous *Program Development Process* diagram used, but without constant reminders, students often forgot to refer to and use it. A copy of this *template* can be found on the design recipe website referenced above.

The next iteration of *this customized design recipe* is when it received its current name. The name design recipe was introduced to the author after attending a workshop on BootStrap, a web-based programming (game design) environment used to help reinforce concepts from algebra [24], by its creator and sponsored by the DC Office of the State Superintendent of Education. BootStrap used a printed worksheet of the design recipe, demanding that students follow the steps and provided space for students to write their thoughts and ideas for each step. This led to the requirement of students documenting ideas, responses, and conclusions in any form they desired and submitting it along with their resulting code. Some students submitted hand written versions, others submitted their responses to each step in the body of an email or as a word attachment, while others submitted their responses as comments at the beginning of their code. One student created an excel spreadsheet to capture each step and provide space for him to document his design ideas in the spaces that followed. Thus, the design recipe took on an interactive form and is now being used by students in two forms: 1) a shared Google spreadsheet or 2) a Google form, with each step described with an example and space for recording their responses. Upon completion, the student either shares the resulting Google spreadsheet or submits the Google form along with his/her assignment.

B. Assignments, Feedback, and Design Enhancements

Once *this customized design recipe* is introduced at the beginning of the course, students follow and complete several guided and open-ended examples during class to understand how to use it. Homework assignments and class projects both require the use of *this customized design recipe*, which is often assigned with a submission deadline before any programming begins. Students are also given a printed copy of the design recipe and can use the recipe's website for guidance at any time. Feedback is collected regarding the number of students who used *this customized design recipe* throughout their attempt at completing a programming assignment. When a steps have been unclear, it explanation in the recipe was either elaborated or split into easier steps. *This customized design recipe* is currently in version 4, has 15 steps, and is filled with encouraging thoughts and hints – including conducting research if needed, thinking positively, documenting assumptions, and adding comments and submitting the resulting code and steps toward solutions. It also includes space where students can express opinions about their performance on each step using emojis and long text descriptions. Performance indicators options include: "step understood," "confused," "need help," "ready to give up," "a piece of cake," and a space to add other performance opinions that may not already be present.

C. Students' Perspective

Initially, students had the impression that the scope of *this customized design recipe* was only in class. They thought it was an assignment and not a guide used to build skills needed by all programmers regardless of skill-level or expertise. Their idea of it was that it was something required by the professor and that it or the steps it described would not be used outside the course. Ultimately, students understood that the steps described are completed by all who programmed, in one form or another. As the time progressed, students used *this customized design recipe* at varying levels. Some started using it more after witnessing their peers use it with some degree of success. Others thought it was too time consuming to fill out for simply programming tasks and concluded that some steps were not needed. Overall, however, students realized that *this customized design recipe* helped them get started, stay organized, and ensure the completion of every required step. Some student sentiments are captured in the following feedback:

- "*It helped me create a program to do what I intentionally want it to do*."
- "*The design recipe really helps me personally. This isn't my field of interest but it has become very interesting. I am very thankful that you provided this for us."*
- *"It helps me understand the process easier."*
- *"The design recipe helps me organize my thoughts as well as my code."*
- *"I never thought about making sure that others can read and understand my thought process. This helps me do this easily."*
- *"I finally solved my problem. Step 7 really helped me figure out what I was doing wrong*." [Step 7 advises students to notice what changes from example to example to determine logic and what should be represented as variables.]

D. Assessment and Instructor's Perspective

Students' abilities to reason computationally, convert ideas to coding logic, and express thought patterns improved tremendously throughout the first few weeks of use. Students use *this customized design recipe* in varying ways and with varying consistency. Some still need to be reminded to use it and what each step is asking them to do, while others think they no longer need it. Overall, those who use it regularly have the most success in completing a programming challenge, have extremely readable code emphasized with appropriate comments, and are less hesitant to start coding. Some students use it to help them formulate questions. Additionally, those who prefer not to speak and/or ask questions during class, use the emojis and text reflections to offer more elaborate descriptions and performance opinions. When students need to express their ideas and answer questions about their code, they often refer to their response to the steps to help guide their responses. This was introduced as "battling," a technique designed by the makers of BootStrap [20], which helps students express their ideas, reasoning, and problem-solving paths. Moreover, knowing students' assumptions and being able to follow their ideas from start to finish provides more insight into what they understand and what they do not. Students are not as reluctant to speak in

front of the entire class about what they did to solve and complete their assignments when their ideas are in tangible form and within reach. *This customized design recipe* also assists students with asking each other questions about challenges and final results as each is already familiar with the steps needed to accomplish a task. This familiarity helps them focus on what is important when reading and understanding one another's code during code reviews.

VI. Lessons Learned and Future Work

Since its initial use, *this customized design recipe* has helped highlight that students have varying problem-solving, reasoning, and coding abilities - as expected. However, it and the results it encourage provide great insight into the exact skills students need to strengthen, those they can help one another with, and areas that need improvement related to attitude and to approaches taken (or not taken) when embarking on a programming project. Students' ability to regulate their learning is increased as well as insight regarding plans for and implementation of differentiated instruction. Overall, it emphasized that when learning how to think computationally and to convert ideas to efficient code, novice programming students at this HBCU need more guidance then just a (check)list of steps. This guidance needs to combine how and when to use all elements of Guilford's Structure of Intellect Model [7]. Incorporating the elements of this model to enrich *this customized design recipe, modifying it to support teaching 'Objects First,'* and making this tool web-based outline future work.

Acknowledgment

Acknowledgement and thanks to Emmanuel Schanzer, creator of Bootstrap, for inspiring the use of the name *design recipe* and to the CS0, CS1, CS2 programming students at my home institution for their trust, their willingness to follow my lead, and their honest and continuous feedback.

References

[1] R. Brooks, Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, *18*(6), 1983, pp. 543–554.

[2] A. L. Brown. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. The Journal of the Learning Sciences, 2(2), 1992, pp. 141-178.

[3] A. Collins, D. Joseph, and K. Bielaczyc. Design research: Theoretical and methodological issues. The Journal of the learning sciences, 13(1), 2004, pp. 15-42.

[4] M. Corney, D. Teague, and R. N, Thomas, R. N. Engaging students in programming. In Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103, 2010, pp. 63–72. Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1862219.1862230

[5] C. Eastlund, D. Vaillancourt, and M. Felleisen. "ACL2 for freshmen: First experiences." In *Proc. 7th Intern. ACL2 Symposium*, 2007, pp. 200-211.

[6] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. "How to design programs: An introduction to computing and programming." 2001.

[7] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. "The TeachScheme! project: Computing and programming for every student." *Computer Science Education* 14, no. 1 2004, pp. 55-77.G. H. L. Fletcher, and J. J. Lu. Education: human computing skills: rethinking the K-12 experience. *Commun. ACM*, *52*(2), 2009, pp. 23–25. http://doi.org/10.1145/1461928.1461938

[8] J. P. Guilford, "The structure of intellect." *Psychological bulletin* 53, no. 4 (1956): 267.

[9] Hatley, L. "Culture as customization: HCI, cultural relevance, and learning technology." In *ACM CHI*, vol. 2011. 2011.

[10] C. Kelleher, and R. Pausch. "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers." *ACM Computing Surveys (CSUR)* 37, no. 2, 2005, pp. 83-137.

[11] R. S. Lemos. "Teaching programming languages: A survey of approaches." In *ACM SIGCSE Bulletin*, vol. 11, no. 1, ACM, 1979, pp. 174-181.

[12] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R.McCartney. "A multi-national study of reading and tracing skills in novice programmers." In *ACM SIGCSE Bulletin*, vol. 36, no. 4, ACM, 2004, pp. 119-150.

[13] L. Mannila and M. de Raadt. "An objective comparison of languages for teaching introductory programming." In *Proceedings of the 6th Baltic Sea conference on Computing education research: ACM, 2006*, pp. 32-37.

[14] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Yifat B. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students." *ACM SIGCSE Bulletin* 33, no. 4, 2001, pp. 125-180.

[15] R. Morelli, and R. Walde. "Java, Java, Java object-oriented problem solving." *Compare* 1, no. 2 (2000): 3.

[16] M. Piteira and C. Costa. Computer programming and novice programmers. In *Proceedings of the Workshop on Information Systems and Design of Communication,* New York, NY, USA: ACM. *2012,* pp. 51–53. http://doi.org/10.1145/2311917.2311927

[17] T. Plomp, and N. Nieveen. An introduction to educational design research. In Proceedings of the Seminar Conducted at the East China Normal University [Z]. Shanghai: SLO-Netherlands Institute for Curriculum Development, 2007.

[18] V. K. Proulx and T. Cashorali. "Calculator problem and the design recipe." *ACM SIGPLAN Notices* 40, no. 3 2005, pp. 4-11.

[19] V. K. Proulx and W. Jossey. "Unit test support for Java via reflection and annotations." In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*, ACM 2009, pp. 49-56.

[20] Y. A. Rankin and J.O. Thomas. "Leveraging food as the context for developing computational algorithmic thinking in an entry-level college course." In *Moving Students of Color from Consumers to Producers of Technology*, IGI Global, 2017, pp. 113-130.

[21] Y. A. Rankin, J. O. Thomas, Q. Brown, and L. Hatley. "Shifting the paradigm of african-american students from consumers of computer science to producers of computer science." In *Proceeding of the 44th ACM technical symposium on Computer science education*, pp. 11-12. ACM, 2013.

[22] A. Robins, J. Rountree, and N. Rountree. "Learning and teaching programming: A review and discussion." *Computer science education* 13, no. 2, 2003, pp. 137-172.

[23] E. Schanzer, K. Fisler, and S. Krishnamurthi. "Bootstrap: Going beyond programming in after-school computer science." In *SPLASH Education Symposium*. 2013.

[24] J. Sheard, S. Simon, M. Hamilton, and J. Lönnberg. Analysis of research into the teaching and learning of programming. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop,* New York, NY, USA: ACM, *2009,* pp. 93–104. http://doi.org/10.1145/1584322.1584334

[25] J. C. Spohrer and E. Soloway. "Simulating student programmers." *Ann Arbor* 1001, 1989.

[26] Zweben, Stuart, and Betsy Bizot. "2016 Taulbee Survey." *Computing* 26, no. 5 2017.