Literate Programming for Authorship of Interactive Textbooks for Programming-centric Courses

Dr. Bryan A. Jones, Mississippi State University

Bryan A. Jones received the B.S.E.E. and M.S. degrees in electrical engineering from Rice University, Houston, TX, in 1995 and 2002, respectively, and the Ph.D. degree in electrical engineering from Clemson University, Clemson, SC, in 2005. He is currently an Associate Professor at Mississippi State University, Mississippi State, MS.

From 1996 to 2000, he was a Hardware Design Engineer with Compaq, where he specialized in board layout for high-availability redundant array of independent disks (RAID) controllers. His research interests include engineering education, robotics, and literate programming.

Ms. Jane N. Moorhead, Mississippi State University

Jane received her B.S. in Electrical Engineering from North Carolina State University. Her career has been all about hardware and software development; with NASA she designed cut-down systems for weather balloons and telemetry systems. Working for IBM, she designing modems and routers and had the opportunity to work at IBM Research Yorktown Heights on the first large-scale voice recognition system. Moving to Mississippi, Jane took a job at Mississippi State University teaching courses in Digital Design using FPGAs, Microprocessors course based on the PIC. She enjoys playing around with new designs for technology, presently playing with the Raspberry PI and Arduino as the basis for projects. Obtaining a GIS-Remote Sensing certificate, she is working on a design of a GPS-based system interfaced with a Raspberry PI. She now is working on a Broadcast Meteorology certificate to find ways to embed hardware into the geosciences curriculum. Always looking for fun educational instructional methods, Jane designed and taught BullyBots - a summer robotics camp for junior-high students; with her college-age students, she has held line-sensing robotics competitions and sea-perch competitions. She requires her students to utilize the development boards to control the robots. To improve the challenge for her students, and always hoping to get back to the water, she decided to embark on the Mate challenge.
Literate programming for authoring interactive
textbooks for programming-centric courses

1. Abstract
Software defines and empowers much of our modern society, from water treatment plants to banking, social media to anti-lock brakes. A wide variety of programming languages (C/C++, JavaScript, Python, etc.) enable computer scientists and software engineers to create the software and firmware necessary to our modern society. As educators, we must therefore find effective ways to instruct students in the use of these complex languages. Therefore, this paper presents a novel approach to programming pedagogy by combining literate programming, in which a program is presented as a web page, with Runestone Interactive, an open-source platform to author and host interactive textbooks. Each program/web page becomes a section in the text, meaning that students can both read the text as a web page, or execute it in their development environment as a program. Questions and programming exercises directly embedded within these program/web pages allow authors to present brief, concise exposition of a new concept followed by comprehension questions and automatically-graded programming exercises to reinforce this concept. Since the textual version of the web page can also be executed as a demonstration program, students can observe these programs in operation, while instructors can automatically check that all their code compiles and executes correctly. A comprehensive set of tools enables instructors to quickly and easily compose books consisting of program/web pages, embed exercises, check the correctness of the programs, then host their textbooks for student use. Student feedback provides preliminary indicators of this approach’s effectiveness.

2. Introduction
Software has become pervasive in every aspect of modern society. Traditional areas defined by mechanical ingenuity have become fertile ground for software engineering; for example, the 2016 Ford F-150 pickup truck relies on 150 million lines of code [1]. Our everyday lives are shaped by smart phones providing a multitude of social media platforms, a secure banking system which moves trillions of dollars electronically, a vast array of industrial control systems to purify water, operate street lights, produce diapers, and much more. This large and growing need for software development expertise must be met by educating more students in computer science, computer engineering, software engineering, and related disciplines. At the same time, persistent problems in effective instruction of introductory computer science courses (CS1) [2] demonstrate a need for innovative methods to effectively instruct these students. To improve this situation, this paper therefore introduces a novel approach to the creation of interactive textbooks by integrating the Runestone Interactive platform [3], Knuth’s literate programming paradigm [4], and traditional software engineering approaches such as unit testing [5].

3. Background
While most large publishers race to embrace the digital revolution with on-line, interactive textbooks, few provide domain-specific tools needed to effectively provide the feedback necessary
to train students in programming-centric courses. Focusing on textbooks for an introductory microprocessors course, the topic of the textbook introduced in this paper, reveals few offerings; traditional textbooks dominate the markets, with titles dating from 1998 still available [6]. Massive online open courses (MOOCs) provide a number of modern offerings; for example, EdX provides an offering of Valvano’s embedded systems course [7] and Coursera offers an ARM and a TI MSP 430 course [8]. However, both courses offer large, weekly programming projects rather than small exercises integrated with the instruction. In contrast, the approach introduced in this paper relies on the use of small, low-stakes assessments and exercises to enable the student to quickly determine what they don’t yet know, then review the missing information.

To fill this gap, the Runestone Interactive platform [3] provides a set of tools for creating interactive textbooks, with a focus on in-browser execution of (JavaScript-emulated) Python, a popular programming language; this platform currently serves over 20,000 students per day at over 600 institutions [9]. Several free textbooks built using this platform focus on Python programming [10, 11, 12]. However, few tools exist to support other languages. As a part of the research underlying this paper, extensions which allowed the use of compiled languages such as C now provide a wider reach for this powerful platform.

Knuth’s literate programming paradigm [4] nicely complements the idea of an online, interactive textbook. Literate programming views a program not only as a formal specification of what a computer should do, but an explanation of why a programmer chose that particular approach. From another perspective, a program can and should be an executable document that both records its purpose while accomplishing its stated goals. In the context of a textbook, a program/document can explain to students why an approach was chosen while also illustrating what steps implement this approach. Figure 1 and Figure 3 illustrate this approach. CodeChat [13, 14], a tool which transforms a program into a web page based on literate programming principles and Enki [15], a powerful text editor which produces Runestone Interactive books, significantly ease the task of authoring a textbook.

4. Approach
The process begins with a set of programs, each of which provides a page in the overall textbook. A build system based on waf [16] complemented by a unit testing framework, both developed by the author, then compiles and tests each program to ensure its correctness. The CodeChat tool also takes each program and transforms it into reStructuredText, from which the Runestone Interactive builder then produces a set of linked web pages representing the textbook. The Runestone Interactive server then responds to student requests to execute a program/web page by invoking a literate programming builder developed for this paper. This builder compiles student code along with test code, runs it on a simulator, then provides feedback on its execution to the student.

4.1 Structure of a program/textbook
Figure 1 shows the composition of a textbook program/web page in Enki. On the left, the source code for this example program demonstrates a working example of a multiply instruction taken
Figure 1 - A program demonstrating multiplication on the left, along with its translation to a web page on the right. Circled numbers are discussed in the accompanying text.
from PIC24/dsPIC33 assembly language [17]. On the right, the web page produced from this program provides a readable, interactive experience for students. Referring to this figure:

1 On the left, structured PIC24/dsPIC33 assembly language comments\(^1\), which begin with a semicolon (\(;\)) and extend to the end of the line, are translated to typeset HTML on the right. The re-use of the code as a document improves:

- Spatial locality and resulting temporal efficiency: concepts presented are interleaved with the corresponding implementation. Therefore, students are both reading a textbook and reading a program simultaneously.\(^2\) Likewise, authors compose both program and document simultaneously.
- Feedback: students can compile\(^3\) and run working code to better understand it.\(^4\) Authors employ automated tools to compile and test every example program, ensuring correctness.

2 A very brief description of the new concept is reinforced by interactive questions and programming exercises, rather than relying on lengthy exposition in the text. Instant, low-stakes feedback (see Figure 2) provides:

- Spatial and temporal efficiency: less writing for authors means less reading for students, saving both space and time.
- Improved engagement: Students are more interested in learning what they don’t know after instant, low-stakes feedback demonstrates this lack. Brief explanations also enable students to rapidly re-read the text to glean from it these missing concepts.

3 A hyperlink directly to the dsPIC33 programmer’s reference manual allows students to directly interact with manufacturer-supplied documentation. This supports ABET criteria (i), “a recognition of the need for, and an ability to engage in life-long learning.”

4 A specialized directive in the example code includes a snippet of test code from `multiplication-test.c` in the resulting document. This saves author time and eliminates many typical copy-and-paste errors made by authors, by automatically copying the needed snippet from test code to example code.

5 Code in the left-side program view is syntax highlighted in the right-side document view, providing a clear visual distinction between interleaved code and comments/exposition.

6 A built-in annotation system, provided by Hypothesis.is, allows students to annotate the text with questions, highlight text for assigned readings, or quickly report errors in the text. In

---

\(^1\) CodeChat, the translation system, employs a readable, lightweight markup language (reStructuredText).
\(^2\) In the author’s humble, somewhat cynical, and completely unsubstantiated opinion, this is such a rare occurrence that it should stand as a significant contribution on its own.
\(^3\) Technically, assemble. “Compile” is a more familiar term for many.
\(^4\) Although students will rarely do so. Instead, they will gravitate toward the convenience of in-browser execution discussed in Figure 3.
the Fall 2017 semester, students left almost 30 annotations which led to rapid improvements of the book.

Next, Figure 2 gives an example of instant, low-stakes feedback integrated into the web page/textbook. The Runestone platform provides a wide variety of question types and a rich, powerful set of tools to create assignments from these questions, automatically grade them, download grades, analyze student performance, and more. In addition, extensions developed by the author to support the use of literate programming automatically remove answers from source code before including it in a compressed (zip) archive for students to download.

![Figure 2 - An example of instant, low-stakes feedback.](image)

These questions reinforce the concepts presented in the textbook, by focusing on primarily on knowledge and comprehension, the lowest levels of Bloom’s taxonomy. Programming exercises, which challenge students to analyze a programming problem then create a solution for it, evaluating any errors in their approach based on feedback from test code, reach to the top of Bloom’s taxonomy. A set of tools and processes developed for this paper provide authors with the ability to easily create exercises. In the textbook currently under development using this toolset, 50 conceptual pages are accompanied by 99 programming exercises, meaning that students typically spend a more time working these exercises than reading and answering questions. Figure 3 provides an example of a typical programming exercise during composition in the Enki editor. In this figure:
; Convert the following C program to assembly. `Test code <multiplication_ex_1-test.c>` checks it. ①

; .. literalinclude:: multiplication_ex_1-test.c ②

; Code
; =====
.text
; void multiplication_ex_1(void) {
  _multiplication_ex_1:
  .global _multiplication_ex_1

; Write assembly which implements the C given above.
;
; SOLUTION_BEGIN
  ; Step 0 - register assignment. Note that `W1` will be overwritten by the multiply, so it isn't assigned or used.
  ;
  ; W0   W0   W2   W3
  ; u16_a = (u16_a + 1401)*u16_b;
  ;
  ; Step 1 - input.
  mov _u16_a, W0
  mov #1401, W2
  mov _u16_b, W3
  ; Step 2 - process.
  add W0, W2, W4
  mul uu W4, W3, W0
  ; Step 3 - output.
  mov W0, _u16_a
; SOLUTION_END

Code
.text
; void multiplication_ex_1(void) {
  _multiplication_ex_1:
  .global _multiplication_ex_1

Write assembly which implements the C given above.

; } ③
return

Save and run ④

Figure 3 - On the left, a program used to create an exercise accompanied with its translation to a web page on the left. Circled numbers are discussed in the accompanying text.
Test code provides feedback on the correctness of the exercise for both authors and students. Authors use a build system which compiles and tests all exercises in the textbook; students may either run their code in-browser or by downloading the code archive then executing a program using all the facilities of an interactive development environment (IDE) and debugger.

As in Figure 1-4, a specialized directive (some lines of which were omitted for brevity) includes a test code snippet.

Solutions written by authors are automatically removed by the system described in this paper to produce student exercises. Web pages have solutions replaced with a text editor in which students compose solutions, while the downloadable source code has solutions replaced with “put your solution here” comments. This enables authors to ensure that their problems are of reasonable difficulty while automatically generating exercises for students.

The “save and run” button causes student source code to be compiled with the appropriate test code, executed, and feedback provided in the box below the button. Test code provides not only simple correct or incorrect feedback, but details inputs, expected outputs, and actual outputs for a number of test cases. The detailed feedback provided assists students in diagnosing problems in their code, enabling students to recognize and correct a variety of errors.

5. Results and conclusions

The Runestone Interactive platform provides a rich instructor interface, allowing the creation of assignments based on an arbitrary combination of low-stakes assessments, such as Figure 2, and programming exercises demonstrated in Figure 3. The platform was first employed in ECE 3724, an introductory course on microprocessors offered in the Department of Electrical and Computer Engineering at Mississippi State University by the author, who instructs one section of this course. A free registration at http://interactive-ebooks.com grants full use of the book. In the Fall 2017 semester, 72 students were required to complete 18 exercises, most of which consisted of two expository pages and all exercises and assessments associated with these pages. Students achieved an average score of 73% on all these exercises. While informal feedback provided by students in class was generally positive, students also reported frustration with the inevitable bugs and errors which accompanied a first-time implementation and draft edition of the book.

A revised and expanded version of the book was assigned in ECE 3724 in Spring 2018 in all four sections of the course by all three instructors of the course, covering a total of 116 students. At this point, students have completed 23 assignments, with an average grade of 79%. A total of 99 students in the authors’ sections of the course were surveyed after completion of 8 assignments on the best and worst aspects of the system. The most common responses were:

Best:

• The programming exercises were short and fairly easy to solve after reading a section.
• The programming exercises provided good practice with assembly language.
Worst:
- Assignments were due every day.
- The assignments page didn’t clearly indicate if all the problems in an assignment were completed.
- Navigation of the book was awkward.

Interestingly, students’ primary concerns were structural, with concerns based primarily on the assignments. Most students found the built-in programming exercises helpful. Based on this preliminary feedback, addition to Runestone Interactive’s assignments feature would address many of these concerns.

Further analysis based on the copious quantities of data automatically collected by the Runestone Interactive platform should provide additional insight into student learning using this novel approach.

In conclusion, the combination of a free, widely-used interactive e-book platform with the literate programming paradigm provides authors with powerful tools to create engaging student experiences in a wide variety of electrical engineering and computer science topics.

References


