A Software Development Course Based on Server-Side Javascript

Mark A. Holliday and Andrew S. Scott
Dept. of Mathematics and Computer Science
Western Carolina University
Cullowhee, NC USA 28723
holliday@wcu.edu

***Abstract*—Full stack (both server-side and client-side) web application development has become one of the primary forms of software development. Consequently, our students should develop skill in software engineering within this context. This presents a challenge because multiple frameworks and languages need to be used, they are rapidly changing, and the programming styles that are important (such as functional programming and event-driven programming) are non-trivial and often new to students. Here we provide a preliminary report on a course that addresses this challenge by being both a first course in a software engineering sequence and an introduction to full stack web application development using current technologies. The course is the third course in the curriculum of a computer science major.**

Keywords—software development course; javascript; Node.js; server-side; web application development, computer science major

I. INTRODUCTION

The graduates of our undergraduate computer science program almost always become software developers upon graduation or after going to graduate school to earn a master's degree in computer science. Thus, a primary emphasis of the design of the courses in our curriculum is to maximize how well our courses prepare our students to become software developers. A challenge we encounter is the role of web application development. Web applications have become one of the primary forms of software development. Unfortunately, web application development is challenging to teach in a substantive manner. Partly this is due to the complexity of the number of languages and frameworks involved and partly due to the rapidly changing nature of those languages and frameworks.

This paper describes one attempt to address this challenge in one of the courses in our computer science major. In Section II we provide the context for our approach. By context we mean where the course fits into our curriculum with respect to the prior courses and the follow-up courses, especially the courses that make up the complete software engineering sequence of our major. Section III presents the goals of the course. Section IV covers the overall structure of the course and key features. Sections V is an overview of the projects in the programming sequence which are on server-side Javascript. In the last section we describe the steps we have taken and plan to take to provide an assessment of the efficacy of this approach and conclude.

II. THE CONTEXT OF THE COURSE

Our university utilizes semesters comprised of fourteen weeks of classes followed by a week of final examinations. Our computer science major starts with a two course introduction to programming sequence using the Java programming language. We focus on developing competency in the programming features and basic data structures (e.g. linked lists, stacks, and queues) so the programs the students develop are console-driven. Our weekly closed laboratory sessions use Window-based computers and an Integrated Development Environment (IDE) (BlueJ [1] and Eclipse [2]). As a result the students have no experience with Linux or with writing graphical user interface code at the end of the second course.

After the second course our students take two courses typically at the same time. One course is entitled Data Structures and Algorithms and the other course is entitled Software Development, the course this paper addresses. The goal of the Software Development course is to move our students from programming competency to learning the skills to be a professional software developer.

Our students typically take the Software Development course in the fall semester of their second year at the university. It starts a four course software engineering sequence. The second course in that sequence is called Software Engineering and is typically taken in the spring semester of the student's second year. In the Software Engineering course the students work in teams of four students on a semester-long programming project. A number of deliverables are produced by the student teams over the semester in addition to the working program. These deliverables and an emphasis on Unified Modeling Language (UML) diagrams help ensure that the students can demonstrate all the steps in the software development life cycle.

The last two courses in our software engineering sequence makeup the two course capstone project the students complete during their senior year. Each student team designs, implements, and evaluates a software project under the direction of a faculty supervisor. Our aim is to reinforce the

principles of good software engineering in a larger and longer scale than is possible within a single course earlier in the curriculum.

III. Goals of the Software Development Course

We first examined some of the key resources regarding what students need to learn about software engineering within a computer science curriculum which are:

- The recent Computer Science Curricula 2013 (CS 2013) [3] defining the Body of Knowledge for computer science with two of the knowledge areas being Software Development Fundamentals (SDF) and Software Engineering (SE).
- The Software Engineering Body of Knowledge, Version 3 [4] which goes into more detail in each of the phases of the software engineering life cycle.
- The recent book by Fox and Patterson [5] focusing on software engineering for web application development when the application is provided as Software as a Service (SaaS) via cloud computing. Fox and Patterson address both agile methodologies and non-agile methodologies.

. Based upon these resources and the context described in the previous section for the courses in our major, we identified the following goals for our Software Development course to be

- Introduction to the software development life cycle including an overview of the key methodologies;
- A development of skills in some of the key specific areas and tools that are used in software development as discussed in the first subsection below;
- Introduction to the Linux command line. The students had only programmed within an IDE and with the Windows operating system. The ability to work from the Linux command line and to edit and work on files using Secure Shell (ssh) is expected by employers of our graduates. Moreover, acquiring these abilities strengthens the student's understanding of what is happening on the computer; and
- Web application development using current technologies as discussed in the second subsection below.

A. Goal of Specific Software Development Ideas and Tools

Given the limited time available in the course we had to be very selective about which software development skills and tools we would focus on. The follow-on course, Software Engineering, covers the topics we did not have time to address. We chose the following:

- Software Testing. An overview of the different types of testing, verification versus validation, and test-driven development; a more in-depth focus on student deliverables for unit testing and code coverage in the context of server-side Javascript.
- Software Version Control. Git [6], a widely-used distributed software version control system.
- UML Class Diagrams. There is only time for one type of UML diagram. We chose class diagrams because they naturally describe object-oriented design patterns.
- Design Patterns. Object-oriented design patterns function as the natural next step given the Java programming the students have done. However, this is best taught in Java, not Javascript [7].

Table 1 depicts the mapping between the primary software development goals of the course and the topics within the SDF and SE knowledge areas of CS2013.

Table 1: Mapping of Software Development Course Goals to CS2013 Topics

Goal	Topic within SDF Knowledge Area	Topic within SE Knowledge Area
Unit-Testing	Development Methods	Verification and Validation
Code Coverage	Development Methods	Verification and Validation
Software Version Control		Tools and Environments
UML Class Diagrams		Software Design
Design Patterns		Software Design

B. Goal of Web Application Development

Web applications have evolved into one of the primary types of software. Unfortunately, the mixture of languages and frameworks used in web applications is complicated and constantly changing.

1. Early web programs used PHP embedded in the client-side HTML/CSS/Javascript with the PHP providing the server-side support and access to database systems.
2. The next generation introduced more traditional languages for the server-side to support Java Server Pages and Active Server Pages.
3. The third generation emphasized the use of Model View Controller (MVC) frameworks and, in particular, Ruby on Rails. It also used dynamically-typed languages for the server-side that are distinct from the languages used for the client-side.
4. The fourth and current generation relies on server-side Javascript. Server-side Javascript allows the same language to be used for the both the client and the server. The asynchronous, message-passing approach of the Node.js Javascript library [8] yields better performance. Thus, the approach that has rapidly become widely employed for new web applications uses server-side Javascript in the form of Node.js in combination with supporting frameworks. The

Express Model View Controller (MVC) framework [9] which we use in the course, is one of the most common frameworks.

Given the nature of the two prior courses in our major we assumed the incoming students had no knowledge of web programming (including none of HTML and CSS) and no knowledge of Javascript or of any dynamically-typed language. Consequently, a substantial part of the course provides this background.

IV. Course Structure

Considering the goals identified in the previous section we developed the following outline for the course which we divide into three tables for the three parts of the course. Graded student work involves four programming projects described in the next section, two quizzes, three tests, and a comprehensive final examination. Quizzes and tests are both closed book, full class period (fifty minutes) and on paper. The only difference is that quizzes count half as much as a test. In addition to the graded student work, I developed fifteen in-class exercises that we do together during class periods. I post answers on the course website after we have gone over each one in class.

A. First Third of the Course: Javascript

The first four weeks of the course focus on Javascript basics in a Linux command-line environment because the students have no background in dynamically-typed languages or in using Linux. We follow the first four chapters of the Javascript book by Haverbeke [10]. Four in-class exercises are completed in class to practice Javascript concepts.

Table 2: First Third of the Course: Javascript

Week	Topic
1	Linux command line and file system manipulation. Basic Javascript concepts using Node.js at the Linux command line. Data types and that a variable can hold values of different types at different times.
2	Javascript operators, expressions, statements, function definitions, function expressions, nested functions. Project 1 posted.
3	Javascript arrays and objects.
4	More Javascript objects including shallow versus deep clones. Quiz 1.

B. Middle Third of the Course: Tools and Web

The middle six weeks are the heart of the course and are summarized in Table 3. The software development tool focus is on unit testing (Mocha [11] and Chai [12]), code coverage (Istanbul [13]), and version control (Git [6]). One in-class exercise on testing concepts and five in-class exercises on practicing with Git are completed in class.

Table 3: Middle Third of the Course: Tools and Web

Week	Topic
5	Overview of software engineering including key phases in the software development cycle and different methodologies. Start on software testing terminology and concepts. Project 2 posted.
6	Unit testing in server-side Javascript using the Mocha framework and code coverage using Istanbul. Mocha is illustrated in Brown's book in Chapters 3-5. The running example in these chapters is a web application so also introduce the students to web concepts including HTML/CSS/template languages and MVC frameworks.
7	Continue unit testing with Mocha framework and add code coverage using Istanbul. Test 1.
8	Start version control using Git as described in Chapter 1 of Pro Git [x] which includes comparisons with previous version control systems.
9	More Git using Chapters 2-3 of Pro Git. Project 3 posted. Quiz 2.
10	Finish Git using Chapter 3 of Pro Git.

C. Final Third of the Course: Design Patterns

The content of the last four weeks of the course is to a large extent independent of the first ten weeks of the course. UML class diagrams and object-oriented design patterns are key concepts in software engineering. This content is outlined in Table 4. However, these concepts are best developed in the context of a traditional object-oriented language such as Java, rather than Javascript. Moreover, the best, in my opinion, coverage of these design patterns can be found in the book Head First Design Patterns [7] which is Java-based. Five in-class exercises give us practice with five design patterns.

Although the new content presented during this part of the course is independent, the student programming is not. During this time period the students are working on the last of the four programming projects. This is the project that turns the version of Poetry Writer developed in Project Three into a web application following the approach of the example applications in the book Web Development with Node and Express [14]. As a result, a significant part of Week 11 is spent going over the Project 4 handout and creating a web-based application using Node and Express.

Table 4: Final Third of the Course: Design Patterns

Week	Topic
11	Introduce UML class diagrams. Overview of object-oriented design patterns. Strategy pattern (Chapter 1 of [7]). Project 4 posted. Go over it using examples from Chapters 3-5, 8, and 10 of Brown [14] as well as a

	working demonstration web application program I wrote.
12	Observer design pattern and begin Decorator pattern (Chapters 2-3 of [7]). Test 2.
13	Finish Decorator pattern. Simple Factory pattern (Chapter 4 of [7]).
14	Factory Method pattern (Chapter 4 of [7]). Test 3.
15	Comprehensive Final Examination

V. The Project Sequence

The four programming projects in the course form a sequence and completing them is fundamental to the student's developing the ability to create and understand web application development using the Node and Express frameworks.

This project was motivated by a one assignment Java command-line program called Poetry Writer. Poetry Writer, when given a word sequence, generates a new word sequence that in a particular sense is similar to the original word sequence. Just like a poem, the words of the output word sequence are grouped into lines which in turn are grouped into stanzas. So number of words per line, number of lines per stanza, and number of stanzas are also input arguments to the program.

- The first project focus on learning Javascript basics by creating a sequence of four data structures (wordCount, wordFrequency, condWordCount, condWordFrequency) each of which builds on the previous one.
- Project Two uses the four data structures to create the output poem. The step of picking the next word given the current word in a manner consistent with the probabilities in the condWordFrequency data structure actually presents quite a challenge for the students. To support deterministic testing of a program that has random output we introduce the concept of allowing one of the command-line arguments to the program to be an array of probabilities.

  The main new Javascript concept is creating your own Node module and using it by exporting the names of the functions you want to make available.
- Project Three introduces the software development techniques of unit testing and code coverage. Unit testing is done using the Mocha Node package which supports both Test-Driven Development (similar to Junit [15]'s approach) and Behavior-Driven Development (similar to the Rspec tool [16]). I require that the unit tests provide 100% statement code coverage; that is, every statement in every function is tested at least once and introduced the Istanbul [13] code coverage tool.
- Project Four introduces version control using Git [6] and conversion to a web application. This project uses the concepts from the first part of Chapter 2 of Pro Git [6]. In particular, the project involves a single Git repository that is not a clone of any other repository and that has only one branch. In class we cover the more advanced cases where the Git repository is a clone of an existing repository that is a remote (that is on a different server) and where there are multiple branches.

  For the conversion of Poetry Writer to a web application Project Four uses the Express framework [9] and Node. The final web version of Poetry Writer is complex so the students first develop an intermediate version.
  - The intermediate version is motivated by the description in Chapter 3 of Ethan Brown's book [14]. Both controller and view parts of a MVC application are created including routing functions, layouts for templating of HTML pages.
  - The final web version of Project Four introduces forms, session variables, and the use of session variables to send form data into a newly rendered HTML page. .

VI. Assessment and Conclusions

We have conducted a preliminary assessment of the course using three approaches: 1) instructor scoring of each of the four projects, 2) Likert scale reporting by the students on three categories of student perception of learning, and 3) qualitative comments by students on course evaluations. We are in the process of completing a more in-depth assessment that we plan to report on in the future. As part of that assessment we plan to conduct a thematic analysis of student comments. Thematic analysis is a widely used technique in qualitative research [17].

This paper describes a design for a particular type of software development course for the computer science major. The course assumes the students have completed a two semester initial programming sequence. Our two semester initial programming sequence has been in Java, but this type of software development course could also follow equivalent courses in other languages such as Python. The primary intent of the course is to provide a first course in the software development lifecycle, tools, and techniques. The specific tools and techniques we have emphasized are unit testing, code coverage, version control, and design patterns.

A secondary intent is to provide an introduction to web application development and Javascript. Web applications have become ubiquitous. Thus, acquiring understanding and skill in web development is important for computer science majors. We describe a four part programming project sequence that helps the student develop those skills in web development as well as skills in the software development tools studied.

Acknowledgments

We thank Benjamin Shults for introducing the original version of the Poetry Writer programming project while he was a computer science faculty member at Western Carolina University. The original version was a single assignment to develop a command-line Java linked-list program using a single data structure.

REFERENCES

[1] The BlueJ Java Integrated Development Environment, http://www.bluej.org/.

[2] The Eclipse Java Integrated Development Environment, https://eclipse.org/ide/.

[3] The Join Task Force on Computing Curricula, Computer Science Curricula 2013, http://www.acm.org/education/CS2013-final-report.pdf, December 2013.

[4] P. Bourque and R.E. Fairley (editors), IEEE Computer Society, The Software Engineering Body of Knowledge, Version 3 (SWEBOKV3), https://www.computer.org/web/swebok/v3G, 2014.

[5] A. Fox and D. Patterson, Engineering Software as a Service: An Agile Approach Using Cloud Computing, Strawberry Canyon LLC, 2nd Edition, 2013.

[6] S. Chacon and B. Straub, Pro Git, 2014, https://Git-scm.com/book/en/v2.

[7] E. Freeman, B. Bates, K. Sierra, and E. Robson, Head First Design Patterns, O'Reilly Media, 2004.

[8] Node.js: a Javascript runtime, https://nodejs.org/en/.

[9] Express: Node.js web application framework, http://expressjs.com/.

[10] M. Haverbeke, Eloquent Javascript: A Modern Introduction to Programming, Second Edition, 2014, http://eloquentjavascript.net/.

[11] Mocha Javascript Testing Framework, https://mochajs.org/.

[12] Chai Assertion Library, http://chaijs.com.

[13] Istanbul: Code Coverage for Javascript, https://Github.com/gotwarlost/istanbul

[14] E. Brown, Web Development with Node and Express, O'Reilly Media, 2014.

[15] JUnit, http://junit.org.

[16] D. Chelimsky, The RSpec Book: Behavior-Driven Development with RSpec, Cucumber, and Friends, The Pragmatic Bookshelf, 2010.

[17] R.W. Boyatzis, Transforming qualitative information: Thematic analysis and code development, Thousand Oaks, London, & New Delhi,, SAGE Publications, 1998.