Integrating a Colony of Code Critiquers into WebTA

Leo C. Ureel II
ureel@mtu.edu
Michigan Technological University
Houghton, MI, USA

ABSTRACT

WebTA is an LTI-based Code Critiquer; a system that accepts student code submissions and provides computing students with immediate feedback on their program design. Previously, WebTA was used to critique Java programs in the introductory CS courses. Our goal is to extend WebTA to include a colony of code critics so that we can use WebTA in our Engineering Fundamentals courses that teach programming in MATLAB and later add code critics to support a diverse range of computing and non-computing students in courses that utilize programming in a variety of languages.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **CS1**; • **Applied computing** → **Computer-assisted instruction**; • **Software and its engineering** → **Patterns**; *Object oriented languages.*

KEYWORDS

CS1, design patterns, autograder, critiquing systems

ACM Reference Format:
Leo C. Ureel II. 2021. Integrating a Colony of Code Critiquers into WebTA. In *Proceedings of Seventh SPLICE Workshop at SIGCSE 2021 "CS Education Infrastructure for All III: From Ideas to Practice" (SPLICE '21).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Computing is playing an increasingly important role in all academic disciplines. Recent years have seen the creation of many computing-centric degrees, such as Bio-Informatics, Digital Humanities, and Human-Centered Computing. Simultaneously, many students in non-computing majors must take programming courses to develop the required computational background for their field.

This has motivated us to extend WebTA, our LTI-based Code Critiquer, to provide feedback and guidance to students learning to program in other disciplines using languages other than Java. For instance, the *MATLAB* language for numerical computing is commonly used to introduce engineering students to programming. We have developed a prototype MATLAB Code Critic that can critique homework submissions and provide just-in-time feedback while students are engaged in development.

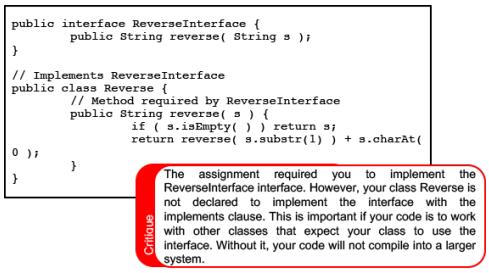

Figure 1: Example of a code critique.

This paper describes our experience integrating the MATLAB Critic with WebTA. The result is an LTI-based critiquer system containing a colony of language-specific Code Critics that we plug into the Canvas Learning Management System (LMS) to assist students in introductory courses featuring programming assignments.

2 CRITIQUER SYSTEMS

In the classroom instructors will, based on experience, identify patterns in student code and call them out. Instructor feedback is quite different from what a compiler would tell a student.

A *Critiquer System* analyzes student programs and responds highly interactive and targeted feedback.[1–3, 5, 9] While approaches vary, these systems tend to make strong use of the instructor's domain knowledge to identify design issues and formulate meaningful responses. This makes them well-suited for providing novices with the kinds of feedback that an instructor might give in a classroom setting. For example, in Figure 1, the students code will compile and may execute without error. However, the instructor can configure the Java Critic to identify when the student has implemented code directly instead of using inheritance. The Java Critic can then provide the student with advice to improve their solution.

3 WEBTA

WebTA is an LTI-based tool that critiques novice code in conjunction with the Canvas LMS. [6–8] WebTA facilitates learning through automatic critique of student source code while the student is engaged in the develop cycle. Students submit code via a Canvas assignment. Their code is stored in a database and passed in sequence to a code critiquer for compilation, testing, and analysis. The critiquer, highlights problems in the student code and provides suggestions for improvement.

When analyzing student code, WebTA looks for *Novice Antipatterns.* Novice antipatterns are recurring code snippets that novices use to solve problems, however the these antipatterns result in errors and bugs. Novice antipatterns represent the kinds of mistakes that experts do not make. Consequently, professional tools

Seventh SPLICE Workshop at SIGCSE 2021 "CS Education Infrastructure for All III: From Ideas to Practice", SPLICE '21, March 15-16, 2021, Virtual Event

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

often do not provide good feedback on these kinds of mistakes. An example of a common antipattern in CS1 courses is the *Empty Loop Antipattern*. After learning looping constructs, a student will sometimes introduce an empty loop to their code simply because "we just covered loops so they must be important!" A code critic can identify the empty loop and suggest to the student that it might not be needed as it contributes nothing to their solution.

3.1 Corpus of Student Submissions

Continued use of WebTA has enabled us to gather a large corpus of student submissions from our CS1 and CS2 courses. We started collecting data in 2014. Since that time, the system has been used by 1,421 students in 27 courses. These students made 64,964 submissions to 119 assignments.

3.1.1 Submission Database. When the student connects through the Learning Management System (LMS) via LTI to the code critic, a unique LTI Session ID is generated that serves as a key for their submission. This and other session data are stored in a `LtiSession` table in the database. Student userid, email, and name are stored in a `Person table`. Course and Assignment identifiers are stored in `Assignment`. Configuration data such as total points, critiquer processes to run (i.e. compile, test, slice, trace, check style), and target language are stored in `Config`. The following summary details important fields in the submission tables.

- Table: `SUBMISSION` - contains information about submitted code files, who submitted them, timestamp, and if the submission has been processed.
- Table: `PROCESS` - keyed to a `SUBMISSION`, this table contains the results of running a critique process (i,e, compiling, testing, style check, etc.) on a submission.
- Table: `Critique` - contains generated critiques keyed to a `PROCESS`. A critique contains the feedback text and the associated line and column range in the code. Entries also contain links to antecedent data indicating how they were triggered and why (compilation error, runtime error, test failure, style violation, antipattern, etc.)

We are currently analyzing the corpus to identify common antipatterns. Developing an awareness of the misconceptions or bad practices student can fall into, we hope provide students with feedback that is more focused, more appropriate to their level of understanding, and less intimidating than what they would encounter in a more traditional development environment.

4 THE MATLAB CRITIC

A prototype MATLAB Critic, being developed at Michigan Technological University, analyzes a student's code providing error and style guidance and feedback. [10]

Similar to the original Java Critic in WebTA, MATLAB Critic compiles, tests, and analyzes code looking for antipatterns. The critic uses the MATLAB Java API; connecting to the MATLAB Engine that parses and executes student code submissions. [4] When an antipattern is detected, the critic generates a critique for the student. The critique covers code structure, shakedown test results, and programming style in a manner appropriate for novice coders.

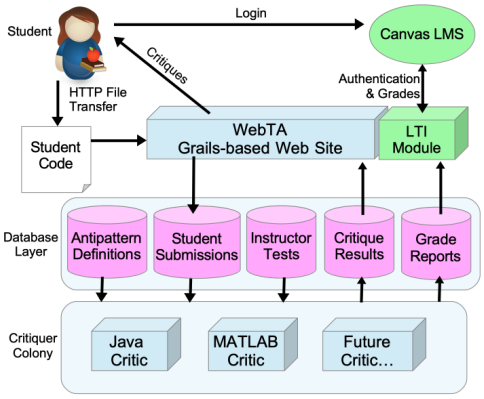

Figure 2: Integrated WebTA Critiquer System with Critic Colony Architecture

5 INTEGRATION WITH WEBTA

WebTA was initially implemented as a single application using a layered architecture comprised of an LTI module for interfacing with canvas, a Grails-based Website front-end, a Java Critiquer layer consisting of a compiler, test stand, and style analyzer, and finally a database.

The decision was made to split the WebTA architecture into a federated architecture with the Grails-based Web Site and LTI Module comprising a front-end that handles all communication with the student through the Canvas LMS (See Figure 2.) The Java Critic module was extracted into its own application. Inter-module communication issues were resolved through the database. A field was added to the assignment table allowing the instructor to specify the critiquer to be applied to submissions.

Once the new architecture was tested with the Java Critic, the MATLAB Critic was added. The prototype MATLAB Critic was initially developed as a standalone application controlled by command line arguments and utilizing flat-files instead of a database. The biggest challenge in this migration was restructuring the code to work with the established WebTA database.

6 ONGOING DEVELOPMENT

We are currently working with Engineering instructors to develop MATLAB assignments that use WebTA with the MATLAB Critiquer. We plan to test the new system in the classroom in a single section of Engineering Fundamentals in Fall Semester 2021 with a full roll-out in Spring 2022. We will report more details as we complete the testing phase of the project.

7 FUTURE WORK

Our goal is to extend the critic colony to several languages used in courses across campus. Our immediate plan is to create a prototype Python Critic for our Introduction to Computing Principles course for non-CS majors. Another goal is to extend our current corpus of Java assignment submissions with MATLAB and Python submissions, then to analyze the corpus to identify cross-language antipatterns. In the classroom, we plan to look more longitudinally on the effects of critiquer-based supports. Moreover, we wish to move our exploratory project into a broader sphere,involving other institutions.

ACKNOWLEDGMENTS

This work builds on M.S. thesis work by Marissa Walther at Michigan Technological University. [11]

This research has been supported by a Jackson Blended Learning Grant, NSF IUSE Grant, and an ICC Seed Grant.

REFERENCES

[1] Norhayati Mohd Ali, John Hosking, and John Grundy. 2013. A taxonomy and mapping of computer-based critiquing tools. *IEEE Transactions on Software Engineering* 39, 11 (2013), 1494–1520.

[2] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically grading programming assignments. *Proceedings of the 13th annual conference on Innovation and technology in computer science education - ITiCSE '08* (2008).

[3] Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, and Alexander Chatzigeorgiou. 2011. JDeodorant: Identification and Application of Extract Class Refactorings. *Proceeding of the 33rd international conference on Software engineering - ICSE '11* (2011).

[4] Inc. Mathworks. [n.d.]. MATLAB. https://www.mathworks.com/products/matlab.html

[5] Lin Qiu and Christopher Riesbeck. 2008. An incremental model for developing educational critiquing systems: experiences with the Java Critiquer. *Journal of Interactive Learning Research* 19, 1 (2008), 119–145.

[6] Leo C Ureel II. 2020. *Critiquing Antipatterns In Novice Code.* Ph.D. Dissertation. Michigan Technological University, Houghton, MI.

[7] Leo C Ureel II and Charles Wallace. 2019. Automated Critique of Early Programming Antipatterns. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* ACM, 738–744.

[8] Leo C Ureel II and Charles R Wallace. 2018. WebTA: Online Code Critique and Assignment Feedback. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* ACM, 1111–1111.

[9] Arto Vihavainen, Matti Luukkainen, and Martin Pärtel. 2013. Test my code: An automatic assessment service for the extreme apprenticeship method. In *2nd International Workshop on Evidence-based Technology Enhanced Learning.* Springer, 109–116.

[10] Marissa Walther, Leo Ureel, II, and Charles Wallace. 2019. A Prototype MATLAB Code Critiquer. In *Proceedings of the 2019 ACM Conference on innovation and technology in computer science education (ITiCSE '19).* ACM, 325–325.

[11] Marissa L. Walther. 2020. MatlabTA : A Style Critiquer For Novice Engineering Students.