String of Perls: Using Perl to Teach Perl

Matthew Z. Smith and Joseph J. Ekstrom
Brigham Young University

Abstract:
This paper discusses the features and design of a web-based system utilizing the unique abilities of Perl to teach Perl to students having little or no prior background in the language, with the goal of writing useful and functional CGI scripts.

The system provides a customized experience instructing students on basic to intermediate skills. The student is presented with interactive content and instruction as well as individual tasks to apply their developing skills, with evaluation and intelligent feedback provided by the system. The system receives student Perl code as input, tests it for compilation, evaluates both the syntax and the semantics of the input code, measures its effectiveness at completing the assigned task, provides feedback to the student, and supports revision and resubmission. The system also contains reference information and direction to external resources.

Using Perl to test and evaluate Perl code has inherent advantages over other methods. Perl is able to compile and execute code in a restricted compartment with a new namespace and share variables and data between the student code and the instructional system. This enables input code to be safely executed and makes possible the generation of detailed feedback.

This paper also discusses the progressing implementation of the system and future plans and expectations. Early experience with limitations, strengths, and applicability of the system to online, lab, and distance learning situations are also discussed.

Introduction:

Over the past three years, Brigham Young University has been implementing a program in the emerging academic discipline of Information Technology[1]. We do not believe that there is sufficient time in any 4 year curriculum to provide proficiency in all of the programming languages that are in common use. We have taken an approach that uses Java as the primary programming language and expect some proficiency in this language. However, we introduce other languages in various courses through providing examples and having students modify working code to provide additional functionality in lab exercises.

We have been seeking ways to increase language proficiency without using class time and without forcing all students to learn the same set of languages. As the authors

discussed this issue of language instruction we decided that Perl provided a unique opportunity to experiment with delivery of web-based language instruction.

One of the authors was employed for a number of years as part of the team that built and maintained web services for millions of websites and developed web-based tools to build, enhance, promote, and maintain those services. This team serviced a broad clientele of enthusiasts, hobbyists, communities, ISPs, schools, portals, businesses, and personal and corporate e-commerce sites. All of this development was done in Perl. This background in web hosting and web-based development in Perl, provided impetus to the idea of creating a web-based system that would provide instruction on basic to intermediate skills in Perl.

Design and Implementation:
We designed the system with a basic distance learning model in mind. Students would access the system through a web browser and progress through various lessons providing a customized experience during the presentation of core language syntax and semantics. Additionally, the system would present various problems for skill exercise and evaluation with intelligent feedback to the student upon submission of the exercise. Reference materials and links to external resources would also be amassed within the system to additionally aid students.

Because our system to teach Perl has been developed in Perl[3], we call it "A String of Perls" (SOP). For the SOP system to provide a customized experience for each student, it begins with a registration process. Each student chooses a unique username and is registered in the system with some basic information. Students choose a password, and are logged into their account on the system upon successful completion of the registration process. Students are able to return to the system at any time and login to their account. Instructors are able to define various lessons as "required", and students' actions and progress are individually logged and tracked, providing them with guided direction through the required lessons based on their completion of requisite instruction and skills. Individual student logs also allow us to collect and analyze data throughout the student's experience with the SOP system, giving us the opportunity to measure the effectiveness of the system and make improvements. The SOP system also provides a notes repository for each student. Students can, at any point within the SOP system, create and save notes as well as retrieve and view any previous notes they may have made. They can also retrieve a history of the code they have written for skill evaluations and the feedback they received. At any time, students can also provide complaints, suggestions, bug reports, or praise through a system-wide feedback mechanism, again in an effort to measure and improve the SOP experience.

After the initial registration process, students are taken directly to the first of the predefined required lessons. Currently the content of the lessons is concise and assumes a basic understanding of programming principles. After each required lesson, a Skill Exercise and Evaluation (SEE) task is presented to the student, and successful completion is required before progressing on to subsequent lessons. The SEE task focuses on performing the core elements of the lesson and asks the student to write a small piece of

Perl code that would meet the requirements of the task. The code is written in a text area within the browser page and submitted to the system. The SOP system then executes the code that the student submitted, testing it for compilation, execution, effectiveness at accomplishing the task, and efficiency. The SOP system compiles intelligent feedback based of these parameters and provides the feedback to the student. The required skills also require a successful completion of the SEE for each skill. If the student passes the evaluation, the next lesson is presented. When the submission of a SEE task does not pass, the student is presented with the compiled feedback and permitted to repeat the task or return to the content of the lesson. The SOP system also provides lessons for skills that may not be defined as required. The majority of these skills will still have an associated SEE task, but a passing response would not be required to proceed to other optional lessons. The SEE is the heart of the SOP system, and we will focus on its function and design separately.

Upon completion of the required lessons, students are presented with their individualized Student Area. The Student Area provides organized access to all the functionality of the SOP system. Menus and customized focus elements provide links to the optional lessons, back into the required lessons, to reference materials and documents, to collections of links to external resources, and to the student's history and notes. They also provide access to the community aspects of the SOP system where the instructor(s) and TA(s) can provide announcements or information, and where students can communicate with each other in a student forum, fostering collaborative progress and assisted development. After becoming familiar with the Student Area, students are asked to complete an evaluation survey on the SOP system, rating their experience and providing feedback for evaluating and improving the system. Even after all lessons are completed, the SOP system can serve the class as both reference material and portal-community resource.

Specific Advantages of Using Perl:

Though language choice for software development is often the result of pure inertia, there have been several benefits of using Perl for development of the SOP system. A module has been developed in Perl by Andy Wardley called Template Toolkit[4]. The overview description from his website follows:

> The **Template Toolkit** is a fast, powerful and easily extensible template processing system written in **Perl** with certain key elements coded in **C** for maximum speed. It is ideally suited (but not limited) to the creation of static and dynamic web content, and incorporates various modules and tools to simplify this process. The Toolkit is highly portable, with minimal dependencies or restrictions on how and where it can be used. It is robust, reliable, well documented and freely available as **Open Source**[5].

Template Toolkit allowed us to develop the format of the pages within the SOP system independently of the content, providing the needed flexibility for producing, modifying, maintaining and expanding the lessons and reference materials as well as dynamically generating the individualized Student Area. Using Template Toolkit will also allow us to easily add components and features we anticipate developing in the future, such as class and school grouping, branding, and customizations.

One of the standard modules provided with the Perl language is the Safe[6] module. Originally designed and implemented by Malcolm Beattie, reworked by Tim Bunce, and currently maintained by Arthur Bergman, Safe is one of the key components that give the SEE subsystem its functionality. The Safe module was developed to provide a protection layer between a Perl program and possibly dangerous operations. It allows for the creation of a new namespace contained within a compartment and provides a way to restrict the viable operations that can be executed within the compartment. To the compartment code, it appears to have its own root (main::) namespace, while the parent program is able to both access the compartment's namespace and to define variables and subroutines it wishes to share with the compartment.

Safe does not, however, prevent many malicious behaviors that will allow code to have a negative effect on the system as a whole. To control these behaviors we have utilized some of the functionality of BSD Resource[7], a Perl module by Jarkko Hietaniemi. BSD Resource allows the getting and setting of process resource limits, which, when used in conjunction with Safe, allows us to adequately control the arbitrary execution of code. The final restriction that we implemented on the SEE subsystem was a modification to the Linux[8] kernel, restricting the rate and number of forks a process can call. These additional restrictions currently restrict the portability of the SOP system to a Linux-based architecture. We will re-evaluate portability in the future, but have accepted this limitation for now.

The SEE subsystem accepts the code submitted from the student and creates a Safe compartment for it. It sets process resources to acceptable limits and executes the code within its Safe compartment. Feedback to return to the student is collected based several factors. If the code does not compile, all compile errors are collected. In addition to compilation errors, the SEE subsystem parses the code and compares it syntactically for key elements that would be required to accomplish the task. Whether or not it finds what it was looking for is translated to English comments and suggestions and added to the feedback collection. The feedback is then returned to the student and logged, giving the student the opportunity to re-attempt the exercise and giving the SEE subsystem the ability to reference previous attempts during subsequent retries. This process also gives the SOP system the opportunity to gather data for evaluation and improvement. If the submitted code successfully compiles, it is executed and the SEE subsystem tests for completion of the assigned task. If the task was not completed, the SEE subsystem parses the code as if it had not compiled, then logs and returns feedback to the student. If the submitted code compiles, executes, and accomplishes the assigned task, it is considered to have passed. Additionally, a rudimentary test for efficiency is provided. If the code is determined to be relatively inefficient, it is parsed and any basic suggestions that can be made for improving the efficiency are logged and returned to the student as feedback.

Present Limitations:

During development tests, it was discovered that the SEE subsystem had a possible weakness in discovering the accomplishment of a task if strange and arbitrary limitations of scope occur within the student code submitted. Perl utilizes different methods of limiting scope, among which using "my" to create a lexically-scoped private variable

proved to function as intended, making it impossible for the SEE subsystem to explore such memory structures. It became necessary to modify the SEE subsystem, to at least attempt to return from the Safe compartment the requested result of the task. Therefore, if "my" is used significantly within a student's submitted code feedback may be more limited than if private variables were not used. However, accommodations have been made for most cases that can be currently anticipated.

Future work:

The first course to use the SOP system will be taught during Spring 2004. We will use this class as a field trial to collect data and draw conclusions about the effectiveness of the SOP system. We will also undoubtedly gain experience with limitations, strengths, and applicability of the system to online, lab, and distance learning situations. Additional development plans include a contextual help system derived from collected data and feedback. We expect that development and improvement of the system will continue over the next several semesters as we use the system for instruction purposes.

Bibliographic Information:

[1] Lunt Designing an IT Curriculum: The Results of the First CITC Conference, ASEE 2002 Session 1626

[2] BYU 2003–2004 Undergraduate Catalog. Retrieved from http://ar.byu.edu/catalog/undergrad_cat/2003/departments/Tech.pdf

[3] Resource: perl, v5.8.0 built for i386-linux-thread-multi (with 1 registered patch).

[4] Resource: Template Toolkit version 2.10, released on 24 July 2003. Author: Andy Wardley.

[5] Text retrieved from http://www.template-toolkit.org/info.html

[6] Resource: Safe version 2.09, released on 06 Oct 2002. Author: Arthur Bergman.

[7] Resource: BSD Resource version 1.23, released on 07 Oct 2003 Author: Jarkko Hietaniemi.

[8] Resource: Red Hat 9.0 OS (2.4.20-8smp)

Biographic Information:

MATTHEW Z. SMITH

Matthew Z. Smith is an Information Technology major at Brigham Young University. Prior to returning to school, he worked as Systems and Quality Assurance Manager for About Web Services. In addition to being a full-time student, he continues to contribute to online development on a contract basis.

JOSEPH J. EKSTROM
Joseph J. Ekstrom (Ph. D. Computer Science, BYU 1992) has been Associate Professor of Information Technology at BYU since 2001. During 30 years of industrial experience he held positions from developer through senior management. His research interests include network and systems management, distributed computing, system modeling and architecture, system development, and IT curriculum and instruction.