The Programming Exercise Markup Language: A Teacher Oriented Format for Describing Auto-graded Assignments

Divyansh Shankar Mishra

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Stephen H. Edwards, Chair
Clifford A. Shaffer
Chris Brown

April 28, 2023
Blacksburg, Virginia

Keywords: programming assignment, automated grading, web service, notation, markup language, interchange format

The Programming Exercise Markup Language: A Teacher Oriented Format for Describing Auto-graded Assignments

Divyansh Shankar Mishra

ABSTRACT

Automated programming assignment grading tools have become integral to CS courses at introductory as well as advanced levels. However a lot of these tools have their own custom approaches to setting up assignments and describing how solutions should be tested, requiring instructors to make a significant learning investment to begin using a new tool.

In addition, differences between tools mean that initial investment must be repeated when switching tools or adding a new one. Worse still, tool-specific strategies further reduce the ability of educators to share and reuse their assignments.

As a solution to this problem, we describe our experiences working with PEML, the Programming Exercise Markup Language, which provides an easy to use, instructor friendly approach for writing programming assignments. Unlike tool-oriented data interchange formats, PEML is designed to provide a human friendly authoring format that has been developed to be intuitive, expressive and not be a technological or notational barrier to instructors.

We describe the design of PEML and also discuss its implementation as a programming library, a web application, and a microservice that provides full parsing and rendering capabilities for easy integration into any tools or scripting libraries. We also describe the integration of PEML into two automated testing and grading tools used at Virginia Tech by the CS department: Code Workout and Web-CAT. We then describe our experiences using

PEML to describe a full range of programming assignments, laboratory exercises, and small coding questions of varying complexity in demonstrating the practicality of the notation.

We evaluate the feasibility of PEML using this encoding exercise as well as the effect of its integration into the aforementioned automated grading tools. We finally present a framework for integrating PEML into existing grading tools and then draw our conclusions as well as list down avenues PEML can be expanded into in the future.

The Programming Exercise Markup Language: A Teacher Oriented Format for Describing Auto-graded Assignments

Divyansh Shankar Mishra

GENERAL AUDIENCE ABSTRACT

Automated grading tools have become ubiquitous to CS courses focused on programming concepts at both the undergraduate as well as graduate level. These tools allow instructors to provide near instant feedback to students as well as spend more time focusing on the curriculum rather than grading.

However, these tools use a variety programming assignment representation formats and without a standardized representation, instructors and educators may struggle to share and reuse assignments across different tools and platforms.

To address this need, we have developed the Programming Exercise Markup Language (PEML), a standardized format for representing programming exercises, designed to be human-friendly as well as easy to learn and use. PEML includes information about the problem statement, input and output formats, constraints, and sample test cases, and can be used for a wide range of exercise types and programming languages.

As part of this master's thesis project, we encoded 50 assignments of varying size and difficulty into PEML as well as integrated support for PEML into Web-CAT and Code Workout, two commonly used automated grading tools used at Virginia Tech. Building upon our experience performing this task, we also designed a framework that can be utilized when integrating PEML into other automated grading tools.

By providing a standardized way of representing programming assignments, PEML can help to streamline programming education and make it easier for instructors and educators to create and share assignments across different tools and platforms.

Dedicated to Virginia Tech.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. DRL-1740765. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer Science is a challenging major. These challenges are not only faced by students trying to grasp complex mathematical and logical concepts and programming languages to implement them in, but also, the instructors who have to work said knowledge to students while providing them with helpful and actionable feedback on their performance. Several forms of automation ranging from course management to automated assignment grading have been made available to instructors but they are not without their own issues. In this section, we will explore the challenges with the latter as well as introduce PEML and discuss how it can help mitigate these challenges. To wrap up this section, we will pose our research questions which we will answer throughout the following chapters.

1.1 Programming Education Challenges

Programming education has become increasingly important as software systems play an integral role in different aspects of our society, ranging from finance to healthcare. However, with the rise of enrollment in computer science courses, it has become challenging for instructors to both focus on the curriculum as well as provide effective and actionable feedback to students. This problem is further exacerbated by multiple factors such as the rapid pace at which technology evolves, and diversity of background, learning styles, and interests enrolled students have.

Programming assignments can be challenging for students at the undergraduate level as they frequently involve learning to use unfamiliar tools and programming languages. They can also turn out somewhat easy for students who have a substantial background in programming and may end up not being fruitful for them. Assignments that are too easy may not provide enough challenge or engagement to students and those that are too difficult may end up discouraging or demotivating them.

In this case, it becomes essential for the instructors to focus on the cohort they have and be able to provide effective feedback to students to keep them engaged with the coursework. Manual grading, in this case, has many challenges [6] which is both time-consuming and error-prone making it difficult for instructors to provide appropriate and timely feedback. Automated grading tools come in clutch in this scenario, providing instructors an effective way to grade students and provide feedback [7] while also spending time focusing on the curriculum. We will touch upon this in the next section.

1.2 The Role of Automated Grading Tools

Automated grading tools are becoming an increasingly common resource for educators, both as a way to cope with increasing student enrollments and as a mechanism to provide immediate and repeatable feedback on the syntax and runtime behavior of solutions. Popular tools such as CodeRunner [17], Gradescope [27], and Web-CAT [10] offer instructors a way to grade programming assignments quickly and effectively, while also providing students with the feedback, the quality and type of which depends on the tools employed. By automating, the grading process, these tools help reduce the workload on instructors, which then allows them to focus on providing more personalized focus to students [2] [26]who might need it. These tools also help address the issue of consistence in grading which is posed by multiple

manual graders grading the same programming assignment.

However, as new adopters explore possible tools, they find that each tool uses separate strategies for how assignments are specified, how behavioral tests are described, and in many cases even what styles of assignments are supported. Further, once one sets up an assignment for one system, the configuration and description is not very portable if one changes platforms or tools as needs evolve.

Further, many tools require direct interaction through a user interface to configure and set up assignments, without providing any facilities for uploading or downloading assignment descriptions. Other tools use docker images, requiring educators to know (or learn) how to use docker to produce assignment configurations. These choices often make writing and maintaining assignment descriptions more time-consuming and involved than necessary, while also restricting the ability of instructors to share their assignments with colleagues, edit or modify assignments across semesters, or maintain their own internal collections of assignments they can adapt and reuse.

1.3 Standard Assignment Specification Format

A lot of automated grading tools utilize different specification formats for describing a programming assignment and how to grade it. While this allows tools to apply the same submission criteria to assignments and provide consistent scores and feedback, it also becomes a substantial operational issue for instructors who utilize different tools across courses, as they are not only forced to learn these different specification formats but also have to spend time and effort converting existing assignments from one format to the other if they want to reuse said assignments across different tools. This reduces the availability of high quality programming assignment for students.

The current lack of a standardized assignment format has led to proliferation of formats that utilize YAML, JSON, and XML. It should be noted that while converting assignments from one specification to another which uses the same mark-up language representation is a challenge in itself, but this is further exacerbates when multiple mark-up language representations are involved.

A standardized programming assignment specification can curb many of these challenges by providing instructors a way to record their assignments in a single format and have them run on a large set of tools. Such a format will also aid sharing and reuse of high quality assignments across courses and even institutions, while providing instructors an easy path to generate and maintain high quality collections or repositories of assignments. Such a format will also allow instructors to reduce the time and effort it takes to develop new assignments, in turn leading to greater consistency and quality in programming education as well as more efficient utilization of resources and greater opportunities for collaboration and innovation.

1.4 Role of PEML

We seek to address these issues through the use of PEML, the Program Exercise Markup Language. PEML provides a human-friendly authoring format for describing programming assignments, while also providing an easy path for parsing and use by automated tools of many kinds. PEML provides instructors an easy-to-learn, expressible, and robust way to write and maintain assignment descriptions and accompanying test suites.

The development of PEML started as part of the CSSPLICE project and the design has been focused on providing a standardized format for representing assignments, including elements and attributes for describing the instructions, input and output formats, constraints on these elements, as well as how to test and validate submissions made for the assignment. The key

features of PEML are its flexibility and extensibility which allow instructors to encode a wide variety of programming assignments ranging from small exercises that are solved by individual students and take a few hours to solve to complex group projects requiring weeks or months to solve.

Another feature of PEML is its emphasis on being human-friendly and readable. The syntax and semantics have been designed to be simple and intuitive for instructors to create and modify programming assignments without requiring extensive knowledge of mark-up languages and specification formats.

By providing a standardized way of representing programming assignments, PEML can help address challenges discussed in the previous subsections and aid sharing and reuse of programming assignments. PEML can also improve efficiency and effectiveness of automated grading tools by providing a common format that can be understood and processed by these tools.

1.5 Research Questions

Having discussed some of issues with programming education, the role of automated grading tools, the issues stemming from their usage and how PEML can help rectify them, we pose the following questions to determine the feasibility of using PEML as a way to bridge the gap posed.

1. What types of programming assignments can be described using PEML?

2. Is it feasible to integrate PEML into a variety of educational tools?

3. Is it feasible to generate software tests in different programming languages from the

same PEML description leveraging PEML's programming-language-agnostic nature?

1.6 Main Contributions

In this section, we list the main contributions of this work. Each of these contributions are discussed in more depth in the following chapters.

1. A **PEML parser** library implemented in Ruby is publicly available as an open-source resource.

2. A **web service** uses a REST API to provides an easy, robust, and programming-language-independent way to access PEML parsing capabilities and features from any tool or scripting language. The REST API provides full access to all the features of the library, including client access to JSON, YAML, or XML representations of parsed contents to promote easier integration with existing tools. The web service supports HTML rendering of assignment writeups as well as programming language rendering of executable software tests that are "ready to use" for grading tools.

3. A **user-facing website** that serves as a "PEML playground" for authors to experiment with and learn PEML, which provides direct user-level access to JSON, YAML, or XML results of parsed descriptions just like the tool-oriented REST API.

4. A **repository** of exercises ranging in size and difficulty encoded in PEML that can be used by instructors as examples when developing their own assignments in PEML.

5. An experience report covering the challenges and benefits of integrating PEML into two existing tools at Virginia Tech: Web-CAT and Code Workout.

6. A framework for how PEML can be integrated into other automated grading tools that builds upon our experiences from the previous point.

1.7 Outline

To discuss the requirement and motivation for development of PEML, the next chapter goes into more depth on other contemporary works in the field of automated grading tools and programming assignment specifications for them. Following that, in Chapter 3, we discuss the design of PEML itself. In Chapter 4, we go into depth about integrating PEML into existing automated grading tools which is followed by our discussion on a framework for PEML integration in Chapter 5. Chapter 6 focuses on our evaluation of capabilities of PEML and we finally draw our conclusions in Chapter 7.

Chapter 2

Literature Review

A central focus of our work is to allow assignment authors to be able to use PEML directly with automated grading tools by making it streamlined for tool developers to incorporate PEML support. This section provides background on automated grading tools, their design specifications, design specifications for testing, and data interchange formats that could be employed.

2.1 Automated Grading Tools

Automated grading tools [18, 34] have allowed instructors of CS courses to focus more on the instructional aspects and on helping students by automating a large stretch of the assignment feedback pipeline. Some of these tools not only allow instructors to automatically grade students but also teach them how to test their code themselves using the tool's testing suite [28], while there are other tools that can even help predict parts of the assignment students will struggle most with [25]. Ihantola et al. [14] provide a survey of many well-known tools where they talk about the development context for some of them as well as common features provided including testing, grading, manual assessment, and submission policies. They conclude with describing a rise in number of new tools being developed, as well as providing recommendations on areas newer tools should focus on. Nayak et al. [22] provide a more recent survey conducted on well-known automated grading tools where they

discuss about features such as type of analysis, feedback generation, grading methodologies, and degree of manual involvement. They also introduce an ideal automatic assessment model that would remedy certain issues current automated grading tools suffer from.

J.C. Paiva et. al. [24] provide another survey on a wide variety of automated grading tools. Their motivations stem from challenges faced by instructors to accurately grade and provide feedback to large cohorts of students which echo our own. In this survey, they examine the types of exercises, the kinds of testing methodologies, the security measures, as well as the types of feedback produced by a set of automated grading tools. They conclude by talking about existing challenges and how they can be bridged. James Peretta [15] also provides a comparative analysis between manual grading versus automated grading for computer science assignments where they show that human graders tend to be inconsistent when grading a large number of students

With the help of automated grading tools, instructors are able to provide instant feedback [8, 11], the nature of which, depends on the choice of tool, its grading approaches, assessment types, and any customization conducted. Some of these tools [16] can generate problems and associated test cases automatically to provide feedback. Feedback from such tools benefits students in many ways [11, 12]. While many such tools are in current use, the focus of this paper is supporting a human-friendly and tool-friendly assignment description format that is tool-neutral, so we do not focus on the use of a specific auto-grader.

2.2 Design Specification for Automated Grading

Automated grading tools tend to be based around certain design specifications which govern the formats of programming assignments, test-suites, and graders [9]. The most commonly used formats are JSON and YAML with a wide variety of further design constraints in

each depending upon the nature of the tool [13]. These can range from support for in-lined external files and support for markdown/HTML all the way to tool specific key-value pair representation of entire assignment descriptions.

Agrawal and Reed [1] provide a survey of grading formats used by well-known automated grading tools. They discuss the prevalence of both tool-specific and JSON-like notations. While it may require additional development effort for a tool to parse PEML descriptions, using formats like JSON and YAML requires less work.

2.3 Design Specification for Test Generation

Automated grading tools often use test cases to grade submitted assignments [31]. Test suites can be part of the tool-specific formal representation of the assignment [30] or be provided as separate input. Some tools use x-unit style unit tests expressed in programming code [4]. Other tools define tests using pairs of given and expected values, while other forgo test suites and instead focus on trace analysis [5]. Recently, there also have been tools [29] that utilize machine learning to automate grading but this remains a challenging implementation. PEML supports both x-unit style as well as data-driven approaches. Test cases can be embedded directly into an assignment description or provided as separate files, using general-purpose or tool-specific formats.

2.4 Data Interchange Formats

When thinking of possible input formats for automated tools, it is natural to think of existing data interchange formats. There are several commonly used data interchange formats that are well-specified and widely supported. However, while well supported by tools, they often

are less friendly for human authoring.

JSON [23] is perhaps the most commonly used interchange format at present. It is language-independent and uses JavaScript syntax. It is based on two types of data structures, a collection of key/value pairs known as a hash, map, dictionary, or struct, and an ordered sequence of values known as an array, vector, or list.

YAML [33] is another well-known data interchange format that is commonly used for writing configuration files. It uses python style indentation to represent nesting. More recently, it has incorporated JSON as a subset.

XML [32] is an older interchange format that evolved from SGML. It has similarities with HTML, albeit with stricter rules and higher verbosity. However, XML is less popular in modern applications because of its verbosity and the more cumbersome nature of its parsing and validation libraries, compared to newer alternatives.

Unfortunately, these existing formats are not writer-friendly enough for descriptions that contain large amounts of free-form text. Programming activities usually require sizeable chunks of multi-line text to describe their properties, whether it is the specification for a program, or starter code to provide to a student, or reference tests to check a solution, or a sample/reference solution to provide for other instructors to look at. In most cases, describing assignments is not focused on simple key/value pairs where values are small pieces of data, or about deeply structured nested object descriptions. It is about writer-friendly input of structured text where most of the values are multi-line text written by humans.

As a result, the most common data interchange formats in use today are not syntax-friendly enough to present a minimal-effort entry path for new authors. Of course, this is not an obstacle for programming-oriented instructors who are already familiar with YAML or JSON. However, those formats do require a learning curve that we hope to minimize further, while

also leading to syntax errors in markup. In particular, YAML's reliance on whitespace and indentation to indicate nested structure can be an obstacle for free-form text input. JSON's use of JavaScript quoting and lack of true multi-line values makes it challenging in similar ways.

Instead, PEML takes significant inspiration from ArchieML [3], an interchange format that originated at the New York Times for non-programmers to write and edit structured text. One of their goals was to make it easy to write structured text "without having [to] type a lot of special characters" in a form that "makes sense to non-programmers". At the same time, however, ArchieML can express exactly the same nested, structured data built from dictionaries and arrays as other notations.

Chapter 3

Design of PEML

The Programming Exercise Markup Language (PEML) is intended to be a simple and easy format for instructors of CS courses of all kinds (college, community college, high school etc.) to describe programming assignments. The goal for the format is to be so obvious to use that instructors will not see it as a technological or notational challenge to learn and express their assignments in.

We also intend for this format to be easy for authors and developers of automated grading tools to adopt so they can, in turn, provide instructors a low-energy onboarding path to get programming activities into said tools. PEML has been designed to support assignment authors by streamlining common use cases to provide a low-energy onboarding path for authors.

3.1 Design Goals

PEML has been designed with the following goals in mind:

1. **Minimal learning curve:** The goal here was to design a format that the average CS instructor can learn in an hour. With a bit of practice, any instructor should be able to encode existing assignments they might have in other data interchange formats, directly into PEML. We want the synatx to be author-friendly so instructors can focus

more on the assignment itself rather than the details of the notation.

2. **Plain-text file representation:** PEML uses a plain-text file notation and can be written or edited in simple text editor software. The goal is to allow instructors to be able to simply copy-paste parts of the assignment directly into a PEML template. For assignments that do not require external files, libraries or other resources, the entire description can fit in a file. Only more complex exercises will need to use PEML's capabilities to import and connect with external resources.

3. **Supports references to external resources:** As discussed above, in most cases, the representation of an assignment will be a simple text file but there might be cases where instructors need to refer to one or more external resources (such as custom data files, a special library, existing PDF documents, etc.). This can be achieved by using relative or absolute URLs to refer to files online or stored locally.

4. **Directory-structured organization of associated assets:** Although it's simple to manage and process exercises that are saved in a single text file, there are instances where referring to external resources stored on a local device is necessary. To accomplish this, the exercise description can include relative URLs that point to other files within the same subdirectory or subdirectory tree where the PEML file is located. This approach treats the subdirectory containing the PEML file as the root of the exercise description. By doing this, it's possible to manage a single PEML file and its associated local resources as a single entity on the local device.

5. **Zip file packaging of multi-file assets with description:** Following up on the last point, for complex exercises, instructors might have to package resources with the PEML text description itself. This can be done through zip-file packaging with the PEML file being in the root of the zip file's internal structure. Relative URLs inside

the PEML file will be treated as paths within the zip file, relative to location of the PEML file. Example of PEML exercises packaged in directories can be found on our repository. These examples can be zipped and shared and unpacked at the source location to produce a subdirectory representation.

6. **Programming language neutral:** The PEML format should allow for the description of programming activities in any programming language, rather than being limited to one language. Although certain parts of an exercise description will be specific to a particular programming language, the notation used to describe the exercise should not be language-specific.

7. **Minimal technology support:** PEML's basic descriptions do not mandate the use of any particular supporting technologies to manage build environments, execution environments, or external dependencies. This approach aligns with the aim of using a simple text file representation, without the need for external resources, for basic programming activities that are similar to those in many textbooks. It is believed that simple, low-overhead assignments will be the most common use case. However, it is acknowledged that some exercise authors may prefer to use specific tools or technologies to package or compartmentalize features related to the execution environment, supporting libraries, runtime dependencies, build environments, custom testing tools, and so on. Therefore, PEML is designed to enable exercise authors who desire such services to use them, but they are not required for mainstream (or simple) exercise descriptions. In other words, instructors who utilize "vanilla" assignments without any special tooling or infrastructure should be able to use PEML to describe their existing exercises without the need to learn about new tools or technologies.

We shall go into more details on how we went about achieving each of these listed goals in

the following subsections.

3.2 Why not YAML? or JSON?

PEML, using the peml parser ruby gem, the web microservice, or the web application can be directly converted into YAML or JSON for instructors who prefer that notation or tools that utilize those notations. This allows for instructors and automated grading tools to utilize the PEML data model. This begs the question, why not use JSON or YAML directly instead?

Firstly, these existing data interchange formats are not very writer-friendly. They do not work well with large amounts of free-form texts. Most programming assignments have large chunks of multi-line text informing students what needs to be solved, what is the starter code provided, what are the boundary conditions, what is the format that the solution needs to be submitted in, what are the test cases that the solution should pass, etc. PEML on the other hand is all about simple collections of key and value pairs where values can be a simple as a small piece of data (string, integer etc) to complex multi-line instructions with embedded markdown or even deeply nested object descriptions. This is how PEML comes out as more writer friendly when compared to other data interchange formats like JSON or YAML.

Secondly, the other data interchange formats discussed are not the most syntax-friendly and do not provide a minimal-effort entry path to instructors. While most instructors of CS courses will likely already have used these data-interchange formats, the constraints of these formats such as indentation, no simple way to import files, limited data types etc.

We realize that instructors will most likely have collections of assignments in these data-interchange formats, but with a simple PEML template, moving to PEML becomes a simple

procedure of copy and pasting content from these existing files and aligning them with the correct keys in the description.

3.3 Influences

PEML has drawn significant inspiration from YAML and its related formats, such as HAML, as well as various other notations created to describe structured textual data. One format that has greatly influenced PEML is ArchieML, which is used by the New York Times for specific types of online content and shares some of PEML's goals. PEML has borrowed extensively from ArchieML's design.

The Awesome JSON page contains a useful compilation of extensions and alternatives to JSON that address some of its limitations as a human-authored notation, such as CSON, MSON, and HOCON. The development of other languages, such as TOML and YAML variants, has also impacted PEML's design.

3.4 Key/Value Pairs

PEML, like YAML and JSON, uses a key-value structure to describe programming assignments. The keys used are alphanumeric identifiers with a more restrictive naming convention compared to YAML, since the use of complex identifiers requires more careful parsing and quoting rules, which decreases writability and increases the potential learning curve. PEML also supports the use of the period character to represent nested keys, as in ArchieML. Keys start at the beginning of the line, and, for keys with single values, end with a colon. Keys that map to a collection have values either surrounded by a pair of square brackets (lists) or a pair of curly braces (hashes or dictionaries). Values follow after the colon, are poten-

tially multi-lined, and can extend to the beginning of the next property. All leading/trailing whitespaces are trimmed and multi-line values are automatically terminated with a single new line. As a result blank lines can appear immediately before any key (or before any unquoted value) for visual spacing/chunking as desired without affecting the meaning.

3.5 Comments

Like YAML, PEML uses the '#' character for comments, which must appear on lines by themselves. The '#' character should be the first not-whitespace character on the line for the corresponding line to be considered a comment. PEML also uses a specific comment line, using '#—' to signal the start of a PEML description. While this is optional for the first PEML description, it serves as a delimiter for subsequent PEML descriptions when multiple exercises are included in the same file or stream.

```
exercise_id: edu.vt.cs.cs1114.palindromes

# Single-line comments start with #
# Comments must be on lines by themselves

title: Palindromes (A Simple PEML Example)
```

Figure 3.1: Comments in PEML

3.6 Quoting

Occasionally, one may have some text in the value that can be considered the start of a key, leading to the instructor having to quote the entire value. PEML supports quoting

using a variant of the HereDoc style, similar to triple quoting in Python and Scala. Any key that is followed by three or more repetitions of the same printing character (forming a custom delimiter) is treated as a HereDoc style quoted value. This is more flexible than triple quoting which itself might appear in assignment descriptions with code fragments. The quoted value is terminated by the first subsequent occurrence of the custom delimiter pattern.

It is worth noting that the '#' character is also commonly used as a comment symbol in many programming languages. However, in PEML, '#' does not have any special significance inside a quoted value. Therefore, to avoid situations where comment lines from a program are interpreted as PEML comments, we advise using HereDoc-quoting for values that contain source code from such programming languages.

```
some_key:"""
You can put any multi-line text inside
here and it is treated as if it is
quoted: even when it contains things
that: look
like: keys and values.
"""
```

Figure 3.2: Quoting in PEML

3.7 Embedding Markdown (and HTML)

PEML also supports textual descriptions written in markdown as well as embedding HTML directly into exercise descriptions. Optional use of other text-based markup formats is also possible, although library users providing PEML implementations may not support all pos-

sible formats. The PEML reference implementation and web service API also support automatic conversion (rendering) of markdown text embedded in exercise descriptions into HTML for client-level use.

3.8 External Resources

PEML allows referencing of external resources in two ways: first, any value can be provided using an external reference rather than inlining it by using the **url(...)** construct (similar to its use in CSS). The URL can be absolute, referring to a resource on the web, or relative, referring to local files that are bundled with the PEML description, such as, in a single zip file. The second way is based on the idea of convention-over-configuration, where external resources like images and libraries are put under the `public_html/` folder that is located alongside the PEML description in the same folder, zip file, or repository. Within Markdown or HTML keys, relative URLs that start with `public_html/...` will then be correctly resolved to these resources.

```
# An example of an external resource
instructions: url(some/directory/assignment.pdf)
```

Figure 3.3: Referencing External Resources in PEML

3.9 Convention Over Configuration

Convention over configuration paradigm aims to decrease the number of design decisions a developer (or an assignment author) has to make while ensuring no lack in flexibility. With PEML's design, we aim to promote choices that reduce the number of design decisions required to encode assignments while also providing flexibility in case the author chooses

another implementation. Conventions on how to add external resources, reference files inside descriptions, describe test suites and other elements that are provided as files can help authors spend less effort on the encoding of an assignment and more time on the contents of the assignment itself.

Although all settings and resources associated with an exercise can be included within the PEML file itself, it's often simpler to provide file-like content as separate files. While it's possible to list external files explicitly using the url(...) operator, it's often easier to just have the files located alongside the PEML description in separate files themselves. This is an example of the convention-over-configuration paradigm where, while the authors have the choice to provide external resources in multiple ways, there is a simple approach to do the same by following the provided directory structure which can reduce encoding efforts. PEML uses a naming convention for sub-directories to locate sets of files to support this approach (which can always be overridden using an explicit url(...) expression as the value for the file set).

For instance, instructions in an exercise may need to reference external images. To address this, the PEML format has a public_html key that refers to a set of files meant to be public web resources referenced in the instructions. Relative path names for images and links in the instructions refer to file resources in the public_html file set. If no public_html key is provided in the PEML description, then the subdirectory with the same name as this key (public_html/) that is located adjacent to the PEML description (e.g., packaged in the same zip file or directory/folder) is assumed to contain the files in the public_html file set. This, again exemplifies the convention-over-configuration paradigm where resources on the web can be provided in multiple ways such as in-lining or using complete urls but the usage of the public_html folder acts as a convention, which, if followed, reduces encoding effort. It also ensures that the files are always available when packaged with the description, which

may not always be true with urls.

In a similar vein, instead of specifying an src.starter.files file set, the author can just place files in the src/starter subdirectory next to the PEML source. For any key that represents a file set with a name ending in .files, that suffix is omitted from the corresponding directory name. In most cases, authors will likely provide external resources by convention rather than explicitly specifying them. This convention also ensures a cleaner PEML description with large files such as code fragments being chunked into external folders rather than in-lined directly into the PEML description. If the same set of physical files will be shared across multiple exercises, explicit url(...) locations can be used to refer to shared file sets without significant effort which ensures that authors always have the flexibility to pick an approach that best suits their requirements. It may also be helpful to place external files in a PEML fragment and use the :include directive, however, this is better suited when splitting up PEML descriptions. This is discussed in more detail in the next section.

3.10 Splitting Up PEML Descriptions

PEML also has support for authors to provide parts of PEML description itself from external files using the ':include' directive, which fetches PEML descriptions from external locations. While not required, it can be used to factor out repeated key/value pairs to aid reusability. This feature also allows authors to separate out test cases and test environments from the main exercise description, placing them in separate files as desired, in order to control access or visbility while allowing the main PEML description to be publicly available.

```
# An example of an external resource
:include url(some/directory/cc-sa-license.peml)
```

Figure 3.4: Importing a PEML Description into Another

3.11 String Interpolation with Variable Values

PEML also allows authors to write parameterized exercise descriptions where many instances of the exercise can be produced using different parameter values for different students. This type of string interpolation is provided using the mustache-compatible notation (‘{{}}’)which is analogous to Ruby’s ‘#{...}’ interpolation syntax. Authors can provide any number of user-defined variables and any occurrence of variable_name is replaced by provided value.

PEML does not support escaping literals ‘{{’ and ‘}}’, so if ‘{{’ ... ‘}}’ exists in the code or description, authors are advised to not use the same variable names as those being used for substitution. A workaround to this is to specify a different set of delimiters to mark parameterized values using the `options.interpolation.delimiter` key or specify variable names that should not be interpolated in the `options.interpolation.exclude` field.

```
instructions:
Draw a square that is {{width}} pixels wide by {{height}} pixels tall.
```

Figure 3.5: Mustache Style Variable Interpolation in PEML

3.12 Nested Structure

Beyond single-value keys, PEML supports a set of nested structures based on ArchieML’s convention for dotted keys, object blocks, and arrays. The key distinctions from ArchieML in PEML include the default use of multi-line values, the adoption of a hybrid HereDoc/triple-quote mechanism instead of a distinct end marker, and the incorporation of comment support. Additionally, special characters are escaped when a delimiter is necessary in PEML.

As with ArchieML, arrays are defined with keys enclosed in ‘[...]’ brackets and are terminated with an empty pair of brackets ‘[]‘. Any trailing empty brackets or braces at the end of the

file are omitted. PEML uses repeated occurrence of the first array item to mark the start of the new item so the first key should be consistent to correctly signify the start of a new item. Data structured in PEML can be arbitrarily nested with the use of dotted keys as discussed above.

Similarly, for hashes, the dictionary is wrapped in curly braces '{}'. As mentioned above, this value (dictionary) can be arbitrarily nested with single values, arrays or nested hashes.

```
[suites]
[.cases]
stdin: racecar
stdout: "racecar" is a palindrome.

stdin: Flintstone
stdout: "Flintstone" is not a palindrome.

stdin: url(some/local/input.txt)
stdout: url(some/local/output.txt)

stdin: url(http://my.school.edu/some/local/generator/input)
stdout: url(http://my.school.edu/some/local/generator/output)
[]
[]
```

Figure 3.6: Nested Array Example

Chapter 4

PEML's Implementation

Having discussed the design of PEML in the previous chapter, we now move on to the different implementations of PEML that allow instructors to test and use it. The implementation of a new tool or system is a crucial next step in ensuring its adoption by the community as well as its success. The implementation process of PEML has been designed with flexibility and ease of use for instructors in mind. Authors who want to start out with PEML or have a suite of existing assignment descriptions and want to encode them in PEML have access to several implementations of PEML including a Ruby gem, a web application, and a microservice using REST API, which allows them to choose the method that best suits their needs and preferences.

Instructors that want to get a feel of PEML and see it in use have access to a rich set of reference materials through the GitHub repository that stores 60 examples of PEML programming assignment descriptions of different sizes and difficulty, making it easier for them to learn and use the language effectively.

Instructors who are well-versed with the use of automated grading tools and want to test out or use existing integrations of PEML with automated grading tools like Web-CAT and CodeWorkout can benefit from the integration of PEML into these platforms, allowing them to create and grade programming exercises more efficiently and effectively.

This chapter explores the different options for implementing PEML, as well as the features

and benefits of each approach, providing a comprehensive overview of how PEML can be used to improve the quality and consistency of programming education. The following subsections correspond to the implementation approaches discussed above. These implementation approaches cover the PEML parser, the PEML REST service, the PEML Rails application, the github repository, and tool integrations, respectively.

4.1 PEML Parser

PEML has been designed with the goal of providing authors of programming assignments with a human-friendly way to describe assignments while also supporting automatic grading based on PEML descriptions. We built a parser for PEML which first converts the supplied textual description into a PEML object following PEML's internal data model. From there, tool developers can convert this PEML internal object into common data-interchange formats like JSON, YAML, or XML for easy consumption by educational tools. Similarly, instructors who are more comfortable with those data interchange formats can convert their PEML descriptions of assignments for easy management and categorization. This also allows for easy sharing and reuse of assignments with instructors or institutions who have not shifted to the use of PEML yet.

The PEML parser is written in Ruby and is publicly available as a gem [21]. The parser can be directly used by instructors who have some familiarity with ruby or automated grading tools written in Ruby on Rails. The main entry point for the gem is the 'Peml.parse()' function, which takes a PEML representation in string form or as a file. The function returns a hash of two key/value pairs: a nested hash representation of the parsed PEML contents, together with an array of any generated diagnostics.

The parser supports optional arguments to control additional processing of the parsed PEML

content. The “interpolate” flag can be set to signal the parser to substitute variables embedded in PEML field values following the mustache-style notation discussed earlier. The “render_to_html” flag can be set to convert descriptions written in markdown into HTML in the returned data representation.

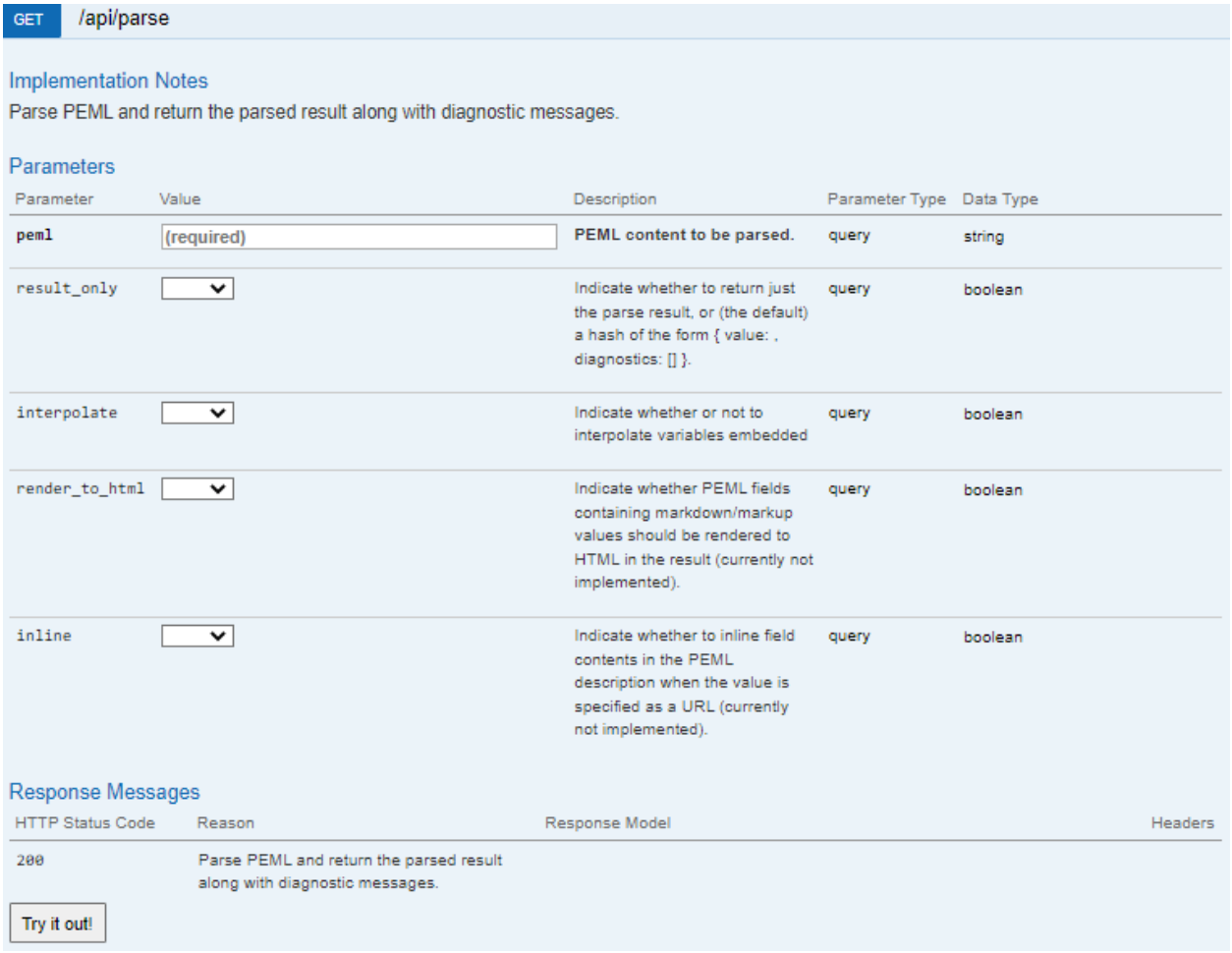
Furthermore, the PEML parser is designed to be flexible and extensible, allowing authors to customize and extend the language as needed for their specific programming assignments. The parser supports the use of custom fields, which can be used to define additional attributes and metadata for each programming assignment. This allows authors to include information such as learning objectives, assessment criteria, and grading rubrics directly in the PEML descriptions, making it easier for grading tools that require those fields to work with and ‘understand’ the assignment. Custom fields might not have any meaning to the PEML data model but can be used as a way to package any extra information required for the programming assignment.

Finally, the PEML parser is continuously updated and improved based on user feedback and contributions from the community. The parser is open source and available on GitHub, allowing authors and developers to contribute to the development and maintenance of the parser and supporting tools. This ensures that PEML remains a flexible and adaptable tool for programming education, able to evolve and grow along with the changing needs and requirements of instructors, students, and educational systems.

4.2 PEML Webservice

We understand the not all instructors work with a ruby gem or the tools they use might not be written using the Ruby on Rails framework, and, as such, may not be able to directly use the PEML parser gem. Another approach can be to develop the parser capabilities in

multiple different programming languages and package them as libraries but this requires substantial development effort.



GET /api/parse

Implementation Notes

Parse PEML and return the parsed result along with diagnostic messages.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
peml	(required)	**PEML content to be parsed.**	query	string
result_only		Indicate whether to return just the parse result, or (the default) a hash of the form { value: , diagnostics: [] }.	query	boolean
interpolate		Indicate whether or not to interpolate variables embedded	query	boolean
render_to_html		Indicate whether PEML fields containing markdown/markup values should be rendered to HTML in the result (currently not implemented).	query	boolean
inline		Indicate whether to inline field contents in the PEML description when the value is specified as a URL (currently not implemented).	query	boolean

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	Parse PEML and return the parsed result along with diagnostic messages.		

Try it out!

Figure 4.1: PEML GET API

Instead, we also provide the parser's features through a web service using a REST API [20] implemented using a Ruby on Rails application. Clients (instructors or automated grading tools) can make REST API calls to the webservice through HTTP POST (or GET) requests, providing a PEML description as a file or as a query parameter along with any additional flags they might require and get the parsed results back in the data interchange format of their choice such as JSON, XML, or YAML according to their needs or based on which data

interchange format the tool they are employing uses. Since the web service is built around the parser, it offers all features provided by the gem.

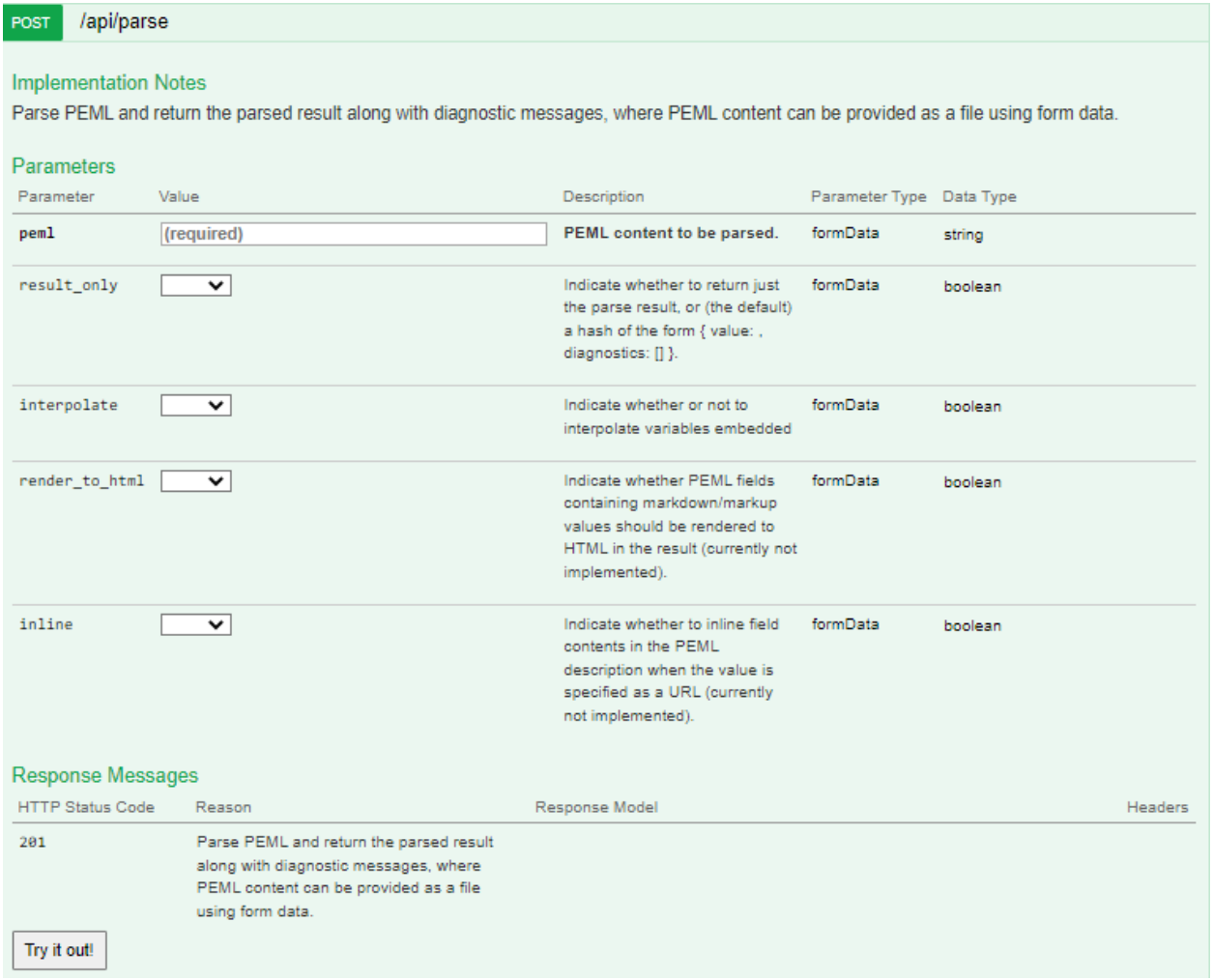


Figure 4.2: PEML POST API

The PEML webservice is hosted on the Virginia Tech Rancher cluster which ensures that it is scalable and performant, allowing it to handle large volumes of requests and parse PEML descriptions quickly and efficiently. The webservice includes a comprehensive set of APIs and endpoints, as well as a detailed documentation and support resources, to help authors and developers get started with using PEML in their programming assignments and automated grading tools.

4.3 PEML Live!

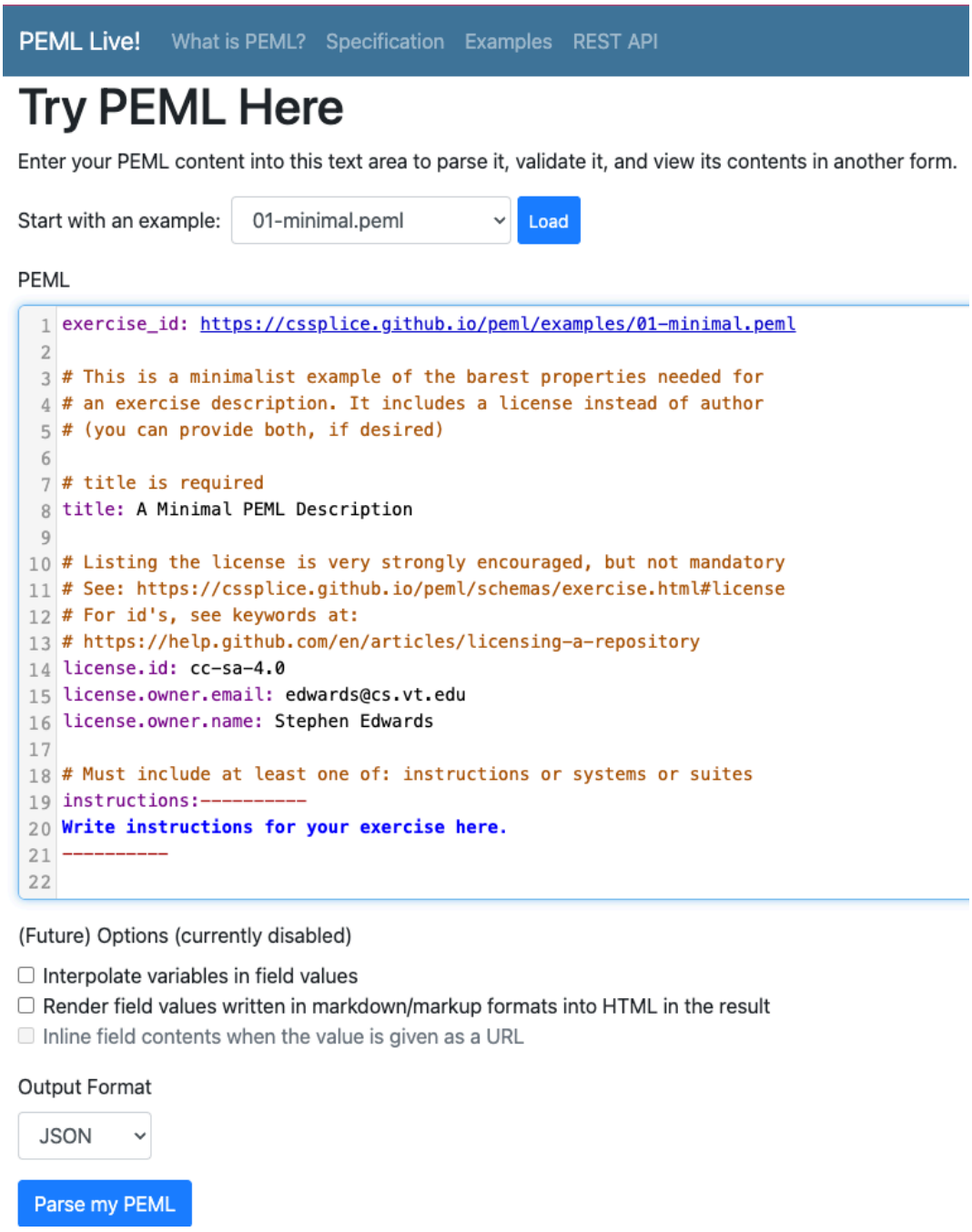


Figure 4.3: The PEML Live! Landing Page

In addition to the Ruby gem and the PEML webservice, we also offer a web application built

on top of the webservice called 'PEML Live!'. This web application provides a user-firendly interface for creating and editing PEML descriptions directly in the user's browser. PEML Live! allows instructors to easily create, modify, and test PEML descriptions as files in a live editing environment, without needing to write any code or have expertise in the underlying technology.

PEML Live! includes a comprehensive suite of documentation, providing instructors with everything they need to get started with using PEML in their programming assignments. Users can select from a set of additional features as well as a preferred response format when parsing their PEML descriptions. Users can see the results of the parser live right in the browser.

The application also provides additional details about PEML such as its goals and design decisions, its specifications, documentation of the REST API, and a diverse set of PEML examples for getting started with writing PEML descriptions. The examples cover a wide range of programming topics and concepts, including loops, arrays, functions, and object-oriented programming. The documentation provides detailed explanations of the PEML syntax, as well as tips and best practices for working with the language.

4.4 GitHub Repository

We also provide a GitHub repository containing a diverse set of PEML examples for instructors to use as a basis for their own programming assignments that was developed as part of our feasibility study discussed later in this work. The repository can be found here. Currently, this repository consists of 60 examples of varying sizes and difficulty levels, covering a wide range of programming concepts and topics, including loops, arrays, functions, object-oriented programming, and more. The repository contains assignments ranging from

simple console stdin/stdout problems to complex GUI focused interactive problems.

Each example in the repository includes a PEML representation of the assignment that can be easily parsed and used with automated grading tools that have PEML integrated into them. Instructors can use these examples as a starting point for creating their own programming assignments, or they can modify and customize the examples to suit their specific needs and requirements, with these examples working almost as a template for different categories of programming assignments.

The PEML examples repository allows instructors to save time and effort when creating programming assignments. Rather than starting from scratch, instructors can leverage the existing examples to create assignments that are tailored to their specific needs and goals. This can be especially useful for instructors who are new to PEML or who are looking for inspiration and guidance when creating programming assignments.

The examples in the repository cover a wide range of programming languages and environments, including Python, Java, C++, and more. This makes it easy for instructors to find examples that are relevant to their specific teaching contexts and requirements. In addition, the examples cover different types of test suites, including rendered unit tests, and data-driven tests allowing instructors to create assignments that are tailored to their specific assessment needs and goals.

The PEML examples repository provides instructors with a consistent and standardized way to represent programming assignments as well as an overview of PEML descriptions used as part of our feasibility study. By using PEML, instructors can ensure that their assignments are easy to encode and understand, while also being machine-readable and compatible with a wide range of automated grading tools that have PEML integration.

4.5 Tool Integration

PEML's integration into two automated grading tools, Web-CAT and CodeWorkout, provides instructors with a powerful and easy-to-use platform for creating and automatically grading programming assignments using PEML. Through these integrations, instructors can create assignments using a human-friendly and easy to use format which will aid sharing and reuse, while also taking advantage of the rich features and capabilities of these tools for automated grading, student feedback, and more. The exact procedure of how PEML has been integrated into these tools is provided in Chapter 6.

The integration of PEML into Web-CAT and CodeWorkout also provides a valuable case study for other tool developers who are interested in implementing PEML into their own tools. By examining the implementation details of these tools, developers can gain insights into best practices for integrating PEML, as well as the benefits and challenges of using this standard notation for programming assignments. To make it easier for developers of automated grading tools, we also provide an outline of sorts on how to go about integrating PEML into existing tools in Chapter 7.

The integration of PEML into these tools provides instructors with a "best of both world" approach where they can use an author-friendly format to describe their assignments, allowing a consistent and standardized way of representing programming assignments, while also being able to use the full suite of features that come with automated grading tools such as providing consistent grades, generating feedback, as well as easily maintain and share assignments across courses.

Chapter 5

Walkthrough for Creating PEML Descriptions

Having gone through the design and implementation of PEML, we now provide an instructor-oriented walkthrough of creating a PEML description. The reader should have a basic familiarity with PEML syntax as well as a thorough idea of the programming assignment that needs to be encoded.

We will begin by introducing the basic syntax and structure of a PEML description, including the use of some optional features that the parser provides. We will also explore the use of various keys and parameters that can be specified in a PEML description including options for specifying programming languages, test suites for automated grading, and environment variables.

Through a series of examples and tutorials, we demonstrate how to use PEML to create and grade programming assignments in a range of contexts and scenarios. Whether you are a seasoned instructor with years of experience in programming education or a new teacher looking to create your first programming assignment, this chapter will provide you with the tools and knowledge you need to get started with PEML and unlock the full potential of this powerful and innovative tool for programming education.

Required Keys	Recommended Keys	Optional Keys
exercise_id	instructions	tag
title	systems	src
author	version	suites
	license	difficulty
		vendor
		tag

Table 5.1: A collection of required, recommended, and optional PEML keys.

5.1 Required Keys

The PEML data model has a set of three required keys that are pretty basic and are commonplace with most other specification formats. Firstly, we have the ‘exercise_id‘ key which is used to uniquely identify the exercise. This identifier can be any sequence of non-whitespace characters that is globally unique. We suggest that instructors stick with existing naming conventions used in other domains, for example, a unique URL that points to the exercise's home definition can be used. Other examples can be a dotted name, perhaps including the institution's domain name as a prefix, much akin to Java packages, or an email combined with a uniqie exercise suffix name. Other standard naming conventions can also be used so long as they lie within the constraints of names values in PEML.

The next required key is the 'title' key which is used to provide a descriptive human-readable name for the exercise. The title key should be identifiable by students in various contexts either when viewing a single exercise or when viewing lists of exercises. While there is no specific length limit, ideally titles should be no more than "one line" in size, because of the various contexts where they might be displayed.

The 'author' or 'authors' key is used to identify the creator(s) of the programming assign-

ment. The singular version of this key can be a simple string, indicating the name or the email. It can also be a dictionary of two keys, 'name' and 'email', which can be used to identify the creator. The plural version of the key is essentially a list of authors, and can either be a list of strings or a list of dictionaries with each then having the 'name' and 'email' keys. The 'author' key can even be omitted provided that a 'license' is provided and the dictionary has an 'owner' key. This is touched upon in the next section.

```
1  exercise_id: <insert-your-globally-unique-id-here>
2  title: <what title will you use?>
3  author.name: <your name>
4  author.email: <your email>
```

Figure 5.1: PEML Required Keys

5.2 Recommended Keys

The PEML data model also has a set of recommended keys and we discuss some of them in this section. The 'instructions' key is where the assignment's main tasks and objectives are outlined for the student to complete. This is a crucial component of the assignment and often comprises the bulk of the content. The value associated with this key is typically a lengthy string, and as with all key/value pairs in PEML, quotes can be used. Markdown is a preferred format for writing instructions, although other markup formats may be supported by certain tools. If there is a situation where the assignment's instructions are meant to be accessed through a course management system, an instructor-provided website, or another method, a URL can be used. Although it is possible to separate assignments into external files or web pages, using PDF assignment descriptions is not recommended as it can limit the usefulness and effectiveness of a PEML resource. However, there may be instances where using PDFs may be the most efficient way for an author to begin, with the intention of

incorporating markup directly into PEML descriptions for future assignments when time permits.

The 'systems' key maps to an array of nested dictionaries that describe the details of the programming language(s) and system(s) the exercise needs to be solved in. This includes details of the programming language(s) and its version(s). Special consideration should be placed on capitalization of the programming language name since some tools processing the description might not treat the name case-insensitively. Optionally, you can also specify a version, if your supporting files are version-dependent. The 'systems' key also supports gem-style version dependency constraints but this needs to be validated with the tool being used.

The 'version' key is an object or dictionary that identifies the version of this exercise that is described. Authors may utilize various forms of version control to manage their sources, which is highly recommended. Therefore, the "version" field can be used to record the access pathways to a version history of an exercise description. Talking about the different nested keys for version, we start with 'type' which captures the type of version control system used for for the PEML description's version history such as git, mercurial, CVS. The 'id' key is used to identify the commit within the repository holding the description. It could be a commit hash, a tag, a version umber etc. The exact meaning of this key depends on the version control system employed. The 'timestamp' key is a human readable timestamp indicating the time at which a version of the description was modified. The 'repository' key is a string or object intended to provide an access path to the repository containing this PEML description's version history. This is most likely a URL, although relative URLs that are resolved relative to the location of this PEML description can be used. Again, some tools may be able to deduce the type, repository, id, and location all from a single URL, such as with direct URLs to files on github.com. In such a situation, only the full location

URL needs to be specified.

The final recommended key is ‘license‘. The “license“ key is associated with a nested dictionary or object that specifies the license applicable to the exercise. The minimum requirement is to include an ”id” that identifies the license governing the use of the exercise. Optional fields may be used to provide additional information. However, if a license is provided, both the “license.id“ and “license.owner“ fields are mandatory. The ‘id‘ key is used to identify the license used which can be a URL to the license or a commonly used name such as any of the license keywords used by github. The ‘owner‘ key is used to identify the person who owns the exercise, in the sense of intellectual property. This can be an individual, a publisher, a corporation etc. Individual authors can use ‘name‘ and ‘email‘ as described in the ‘author‘ key. The ‘book‘ key can be used to identify a book in case the exercise is derived from it. In this scenario, it is assumed that the exercise has the same license as the book. The ‘attribution‘ key, if provided, can be used to store an acknowledgement string that the license owner wishes for users of the exercise to include when using the work. The ‘acknowledgement‘ key, if provided, can be used to identify all the attributions this exercise makes for licensed use of other (separate) content. While the license.attribution contains content for the users of this exercise to include in derived works, the license.acknowledgements contains attributions acknowledging other content this exercise uses. Lastly, the ‘permission‘ key can be used to specify access permissions for the exercise. This key used a set of enums which are: “none“, “read“, “fork“, “fork-with-tests“, “contribute“, and “all“.

5.3 Optional Keys

Optional keys can be specified to add more context to the assignments when being used with tools or when being worked upon by students. The first of these keys is ’tag’ which is

```
instructions:----------
Write instructions for your exercise here.
----------

[systems]
language: Java
version: >= 1.9
[]

version.type: <your version type here>
version.repository: <link to your repository>

license.id: <id for your license>
license.owner: <the license's owner>
```

Figure 5.2: PEML Recomended Keys

a dictionary that can be used to tag exercises with different topics such as object oriented programming, loops, conditionals etc. Next, 'src' is another dictionary that holds the source code definition for starter and wrapper code for the exercise. 'suites' is another dictionary that holds information about the testing criteria as well as test cases that solutions are tested and validated against. 'difficulty' key is an integer and is used to rate how difficult an assignment is. Finally, the 'vendor' key is another dictionary that holds any tool-specific keys or extension properties that individual educational tools might support which are not explicitly part of the PEML data model. This can be used to build out keys/properties that identify grading schemes, late policies, submission constraints, options for processing pipelines that are specific to a tool.

```
tags.topics: <topic(s) this exercise focuses on practicing>
tags.prerequisites: <should already know these, can specify: exposure, familiarity, mastery>
tags.style: code-writing <or other choice>
tags.course: <name your course, if relevant>
tags.book: <name your textbook, if relevant>
tags.personal: <optional personal/custom values that aren't topics>

[suites]
[.cases]

stdin: url(some/local/input.txt)
stdout: url(some/local/output.txt)

stdin: url(http://my.school.edu/some/local/generator/input)
stdout: url(http://my.school.edu/some/local/generator/output)

[]
[]

[.src.files]

# Example of file included inline:
name: Answer.java
content:----------
public class Answer
{
    // Insert your answer here
}
----------

# Example of file included inline, URL relative to PEML description:
name: AnswerTest.java
content: url(src/AnswerTest.java)

[]

difficulty: <a numerical measure of difficulty>
vendor: <vendor specific keys go in this hash>
```

Figure 5.3: PEML Optional Keys

5.4 Writing a PEML description

To start writing a PEML description, you need to know the details of the assignment you're encoding. PEML is a plain-text representation of the programming assignment and can be written and modified in any text editing tool. To start, the three required keys, 'exercise_id',

'title', and 'author' need to be specified. If there are multiple authors, you can use the 'authors' key. The 'exercise_id', as discussed above, should be globally unique and the 'title' should help identify the assignment clearly.

```
60  # Multiple authors with names:
61  [authors]
62  name: author 1
63  email: email 1
64  name: author 2
65  email: email 2
66  []
```

Figure 5.4: Representing Multiple Authors

Once the required keys are set, you can then move to the recommended keys. Starting with 'instructions', which is the bulk of the assignment, this section should hold the requirements and the end goals of the assignments. Here-doc style quoting can be used and the instruction itself can have markup to specify sections, headings, or code-blocks. Once the instructions have been specified, you should list down all the supported programming languages under the 'systems' key. Optionally, list down the version of the supported languages if the programming assignment is language dependent. While specifying these keys will set you up to publish the assignment (without test cases, but those can be supplied externally too), we recommend adding the version key in case the assignment has a github repository (or some other version control process) as well as the license key to ensure proper access permissions as well as clarifying sharing and resue criteria of the assignment.

Having specified all the core-keys to make the PEML description human-readable, you can now move onto embedding external files as well as specifying auto-grader inputs. PEML offers a comprehensive model for organizing this information to facilitate tool usage. However,

PEML predominantly relies on convention over configuration to streamline the management of these aspects and to simplify the learning process for authors. This approach enables authors to begin with the minimal amount of knowledge necessary and gradually expand upon that knowledge over time as more complex scenarios arise.

To provide students with some starter code to start working on their solution, files can be added under 'src/starter' directly, next to the PEML description itself. It is advisable to create a separate directory for each exercise that utilizes supplementary resources. This method is applicable regardless of whether the PEML description is stored in a local folder, packaged in an archive such as a zip file, or hosted in a repository. Alternatively, the [src.starter.files] array key may be utilized to specify this information within the PEML description itself. However, it is often more straightforward to provide these resources via co-location. This also holds true for any wrapper code (classes in java etc.) which can be similarly provided under 'src/wrapper' or inlined directly in the PEML description under the [src.wrapper.files] array.

```
# Providing starter files for the user
dir
|-- exercise.peml
+-- src
    +-- starter
        |-- file1.ext
        |-- file2.ext
        +-- file3.ext
```

Figure 5.5: Providing Starter Code in a PEML File Directory

Assume that you wish to provide images that can be used in the exercise instructions. The instructions are usually written in Markdown or vanilla HTML and may refer to supplementary files like images, separate pages that describe APIs, examples for students to download,

and so on. Instead of using absolute URLs to host these resources on your own website, you may prefer to package them with the exercise. You can use the [public_html] key to specify these explicitly, or simply place them in a 'public_html' folder which, again, should be next to the PEML description, whether it is package as a directory, a repository, or a zip file.

```
# Providing images files for the instructions
dir
|-- exercise.peml
+-- public_html
|   |-- image1.png
|   |-- image2.png
|   +-- download_file.dat
+-- src
    +-- starter
        |-- file1.ext
        |-- file2.ext
        +-- file3.ext
```

Figure 5.6: Providing External Images in PEML

Assume that the resources utilized by your automated grading tests require additional data files or other resources that must be accessible when executing the tests. Not all grading tools can accommodate such resources, but for those that can, you can include them in the 'environment/test' directory, which corresponds to the [environment.test.files] key. The top-level keys, such as src., environment., or [suites], have an impact on the entire exercise, which means that they apply to all supported programming systems. This is usually acceptable when the exercise targets only one programming language. However, if the exercise supports multiple programming systems, it may be necessary to provide different resources for each system. To accomplish this, the same directory structure for 'src/', 'environment/', and 'suites/' can be used, but placed underneath 'systems/<language>' (using the language name as specified in the PEML file. If a MIME type is used, replace the '/' in the MIME type with a '-'). It should be noted, however, that most automated grading tools support grading for one programming language but even with multiple languages specified, the tool

should be able to work with the first one.

```
# Providing system-specific resources
dir
|-- exercise.peml
+-- systems
    +-- Java
    |   +-- src
    |   |   +-- starter
    |   |       |-- Class1.java
    |   |       +-- Class2.java
    |   +-- suites
    |   |   |-- TestClass1.java
    |   |   +-- TestClass2.java
    |   +-- environment
    |       +-- test
    |           |-- file3.ext
    |           +-- file4.ext
    +-- python
        +-- src
        |   +-- starter
        |       |-- class1.py
        |       +-- class2.py
        +-- suites
        |   |-- test_class1.py
        |   +-- test_class2.py
        +-- environment
            +-- test
                |-- file3.ext
                +-- file4.ext
```

Figure 5.7: System Specific Resources in PEML

5.5 Incorporating Tests

Having followed the steps from the previous section, you should have a pretty in-depth and well written assignment. But for an assignment to be automatically graded you need to

add test cases that candidate solutions will be tested and validated against. Test cases, in PEML, can be specified in multiple ways. If you want to include files that define the tests for the exercise, such as scripts, compilable program code, or data files, you can place them in the './suites' folder. The suites folder corresponds to the [suites] array key. You can include multiple files and they can be in any notation or format used by the auto-grading tool that will read the PEML description.

```
# Providing test case files
dir
|-- exercise.peml
+-- public_html
|   |-- image1.png
|   |-- image2.png
|   +-- download_file.dat
+-- src
|   +-- starter
|       |-- file1.ext
|       |-- file2.ext
|       +-- file3.ext
+-- suites
    |-- TestClass1.java
    +-- TestClass2.java
```

Figure 5.8: Defining Tests With PEML's Directory Structure

If the tests for an exercise are relatively straightforward and follow a similar format with minor variations, it may be more convenient to include the test data directly within the PEML description rather than providing separate test cases. The test cases can also be written in other formats such as CSV, YAML, JSON etc. depending on what the auto-grading tool employed supports. Due to the potential for errors in manually quoting data in CSV format, PEML offers an alternative format known as "text/x-unquoted-csv." This format allows values to be written in the same notation as the target programming system, enabling the use of expressions, literal constructs, escapes, and other native features in a more natural way.

```
name: csv_stdio_tests
type: text/x-unquoted-csv
# patterns are user-defined strings built by variable interpolation from the
# "columns"/fields in a test case.
pattern.description: sumNumbers() ->
pattern.actual: subject.sumNumbers()
# Some tools may use a template, populated by variable substitution using
# the columns/fields/patterns for one test cases, to produce an executable test.
# whether and how this is supported is tool-dependent.
template:
    assertEquals(, );
content:----------
str,expected,description
"abc123xyz",123,example
"aa11b33",44,example
"7 11",18,example
"Chocolate",0
"5hoco1a1e",7
"5$$1;;1!!",7
"a1234bb11",1245
"",0
"a22bbb3",25
"FS3453g36fs25",3514,hidden
"dfg64g21ge743",828,hidden
"2sdf4523sdfsd7",4532,hidden
"sdffherbwm",0,hidden
----------
```

Figure 5.9: Data Driven Tests in PEML

Although some tools can interpret the test cases written in PEML's data-driven format, not all tools can utilize them effectively. To address this, the PEML parser offers an automatic conversion feature that generates executable test files based on the programming languages specified under the 'systems' key. These data-driven test cases are still included in the description, but the newly created executable test cases are appended along with any generated metadata, such as the number of tests, language, versions, and imported external libraries.

```
exercise_id: edu.vt.cs.cs1114.palindromes

# Single-line comments start with #
# Comments must be on lines by themselves

title: Palindromes (A Simple PEML Example)

license.id: cc-sa-4.0
license.owner.email: edwards@cs.vt.edu
license.owner.name: Stephen Edwards

topics: Strings, loops, conditions
prerequisites: variables, assignment, boolean operators

instructions:----------
Write a program that reads a single string (in the form of one line
of text) from its standard input, and determines whether the string is
a _palindrome_. A palindrome is a string that reads the same way
backward as it does forward, such as "racecar" or "madam". Your
program does not need to prompt for its input, and should only generate
one line of output, in the following
format:

```
"racecar" is a palindrome.
```


Or:

```
"Flintstone" is not a palindrome.
```

```

Figure 5.10: PEML Description for an Exercise to Find Palindromes (1)

```
assets.test_format: stdin-stdout

[systems]
language: java
version: >= 1.5
[]

# Test data/files/classes are probably located in separate files, but this
# is a simple example of how to do something directly inline
[suites]
[.cases]

stdin: racecar
stdout: "racecar" is a palindrome.


stdin: Flintstone
stdout: "Flintstone" is not a palindrome.


stdin: url(some/local/input.txt)
stdout: url(some/local/output.txt)


stdin: url(http://my.school.edu/some/local/generator/input)
stdout: url(http://my.school.edu/some/local/generator/output)

[]
[]
# The [] ends/closes a list of items, but they can be omitted at the
# end of the file, since EOF auto-closes any open lists. The first []
# closes the list of cases in one suite, while the second [] closes
# the list of suites, which here includes only one suite.
```

Figure 5.11: PEML Description for an Exercise to Find Palindromes (2)

Chapter 6

Integrating PEML into Tools

The integration of PEML into existing automated grading tools will allow instructors to integrate PEML into their existing workflows, but it requires careful consideration of technical and design aspects. In this chapter, we describe our experiences integrating PEML into two automated grading tools commonly used at Virginia Tech: CodeWorkout and Web-CAT. We outline the steps we took to integrate PEML into these tools, discuss the design decisions we made, and evaluate the benefits and challenges of using PEML in these contexts.

CodeWorkout is a web-based system for creating, sharing, and grading programming exercises. It is used in a range of courses at Virginia Tech, including introductory programming courses. Our integration of PEML into CodeWorkout involved modifying the existing infrastructure to allow instructors to create and upload PEML descriptions for exercises. This required designing a new interface that allowed instructors to easily write and test their PEML descriptions, as well as modifying the system's backend to support parsing and execution of the descriptions.

Similarly, Web-CAT is an open-source automated grading system used at Virginia Tech and other institutions. It provides a range of features for grading programming assignments, including support for data-driven test cases, automated feedback, and plagiarism detection. Our integration of PEML into Web-CAT involved developing a new plugin that allowed instructors to upload PEML descriptions for their assignments. This required adding elements to the existing Web-CAT internal assignment data model to support parsing and execution

of the descriptions, as well as designing a new user interface that allowed instructors to easily write and test their PEML descriptions.

In the following sections, we describe in detail the technical aspects of our integrations of PEML into CodeWorkout and Web-CAT. We discuss the design decisions we made, the challenges we encountered, and the benefits we observed. We also provide recommendations for future improvements and enhancements to the integration process. Overall, our experience demonstrates the potential of using PEML in automated grading tools to improve the efficiency and consistency of programming education.

6.1 Integration with CodeWorkout

CodeWorkout is an open source drill and practice system that supports short programming exercises and multiple-choice questions. CodeWorkout utilizes a combination of open, gradual-enagement policy where students can solve exercises without needing an account or being enrolled in a course, together with strong course management features that support graded quizzes and assignments. Moreover, CodeWorkout provides a rich data collection and evaluation infrastructure which facilitates CS education research.

CodeWorkout is written using Ruby-on-Rails and uses a MySQL database to store data. CodeWorkout also supports programming assignments in a wide range of programming languages such as Java and C++. In addition, it provides tools for automatically grading these programming assignments, including support for test cases in multiple forms. CodeWorkout also provides a number of features to help students learn programming, such as online code editors, syntax highlighting, and the ability to run code and see its output. In this section, we describe PEML's integration with CodeWorkout.

CodeWorkout has been used in a number of different programming courses, including introductory courses, data structures courses, and software engineering courses at the CS1 level. It has been shown to be effective in helping students learn programming, and has been well-received by both students and instructors.

6.1.1 Motivation for PEML Integration

Before moving on to the technical details of PEML's integration with CodeWorkout, it is important to understand the motivation behind doing so. CodeWorkout is a widely used automated grading tool which allows instructors to create, modify, and grade assignments efficiently. However, the usage of the tool is not without its own challenges. Instructors wanting to use CodeWorkout have to describe their assignments in YAML, which is the data interchange format utilized by this tool. While it is easy for the tool to parse and work with YAML, owing to built in library as well as many YAML focused open-source gems, it is actually quite challenging for authors to write complex assignment descriptions in YAML because of the strict indentation rules as well as poor support for large, free-form multi-line texts.

This problem is only exacerbated when it comes to writing test cases. CodeWorkout supports CSV-like tests with function templates as well as large x-unit style executable tests. For the former, ensuring proper quotation for CSV-like test is a challenge while the latter again suffers from being a large multi-line text value which itself might have indentations required for the proper execution of the tests.

PEML solves both of these problems by providing an instructor-oriented authoring format with less stringent rules when it comes to indentations and quotations. PEML is much easier to describe programming assignments in when compared to YAML and PEML's special

CSV-like format for data-driven test cases allows proper quotation for any values without interfering with the assignment's structure. Moreover, PEML's ability to generate executable test cases from data on the fly allows instructors to write concise assignment descriptions which are easy to maintain and share. By integrating PEML into Code Workout, we hoped to provide a more efficient and streamlined way for instructors to create and manage programming assignments, while also maintaining consistency and quality across assignments.

6.1.2 Design Decisions

When planning PEML's integration into CodeWorkout, we had to make several decisions. One of the major design decisions was to use the PEML parser's gem instead of the REST API. This decision was made based on the fact that CodeWorkout is written in Ruby, which made it easier to use the gem without incurring additional network overhead from using the REST API. By using the gem, we were able to integrate PEML parsing capabilities directly into CodeWorkout's existing Ruby codebase.

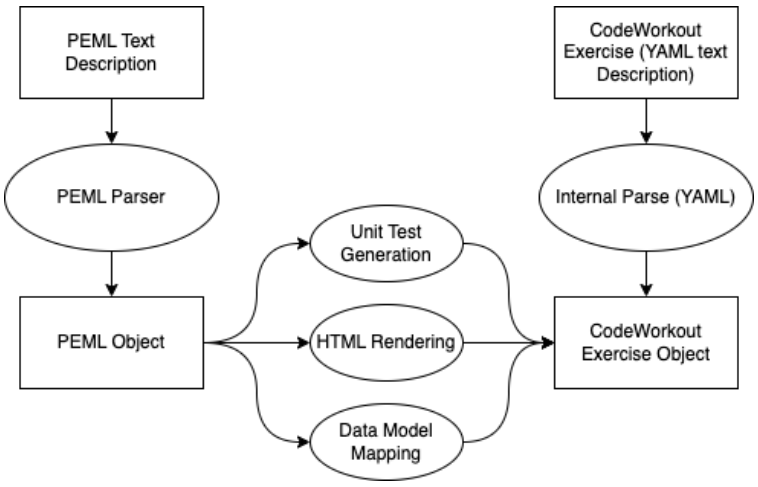


Figure 6.1: CodeWorkout's PEML Integration

Another important design decision we made was to keep new PEML assignments consistent

with the exsiting CodeWorkout exercises, which were originally written in YAML following CodeWorkout's internal data model. We decided to do this because PEML's data model is similar to CodeWorkout's internal data model. By not having two different models for the two types of exercises, we were able to keep the codebase simpler and more manageable.

We also made the decision to keep UI changes minimal in order to make it easier for existing CodeWorkout users to use PEML. To achieve this, we made it possible for users to simply write PEML descriptions into the existing place for YAML descriptions, as the backend could automatically detect what kind of description had been passed to it. By minimizing the UI changes, we were able to avoid confusion and make it easier for users to transition to using PEML without having to relearn the interface of the tool.

6.1.3 Implementation Walkthrough

In this section, we provide a walkthrough of sorts on what happens when a PEML description is provided to CodeWorkout. As discussed earlier, PEML is integrated into the tool as a gem since CodeWorkout itself is written in Ruby on Rails. The prompt provided to instructors remains unchanged between YAML and PEML description with added UI elements that provide a quick guide to PEML for any instructor who wants to make the switch.

Once the description is submitted, it is parsed into a plain ruby hash. This hash is then passed to a middle-layer of sorts which is responsible of converting this hash into one amenable to CodeWorkout's internal data model. This is also where any executable test cases generated are added to the hash. After this step, if the description is validated, it is recorded as a new workout which appears on the instructors screen. If any errors occur during the processing, the instructor is immediately notified of issues with the descriptions. PEML's generated diagnostics prove to be very useful here.

6.1.4 Limitations

PEML's integration with CodeWorkout isn't without its own challenges however. We believe that a discussion of these limitations can be of value to developers planning to integrate PEML into their own tools. Firstly, there isn't a one-to-one map between PEML's data model and CodeWorkout's internal data model for assignments. This makes the process of transforming a PEML description into a CodeWorkout assignment not entirely lossless. While we have made attempts to preserve as much information as possible, ultimately, this mismatch can lead to certain fields being omitted. It should, however, be noted that none of the fields omitted such as 'public_html', 'url', etc. are required by CodeWorkout to utilize these assignments as any of those not written in PEML.

Another limitation of this integration is the UI to enter the PEML description being geared towards YAML. This is not a huge challenge, as discussed in the chapter on evaluation, since PEML's structure does not require a lot of syntax highlighting and the development of a custom edit mode for PEML is not a huge challenge in itself, but for instructors using CodeWorkout, making the shift from YAML to PEML may be initially visually jarring.

6.2 Integration with Web-CAT

Web-CAT is an open-source automated grading system that provides support for a wide range of programming languages and assessment strategies. The system is highly customizable and extensible, making it a popular choice among educators who are looking for a flexible and adaptive grading solution. Web-CAT's most notable feature is its ability to grade students on how well they test their own code. This approach has been found to offer greater learning benefits compared to traditional output-comparison grading methods, as it

encourages students to write more effective test cases and improve their understanding of the software development process.

To store assignments, Web-CAT uses a directory structure similar to GitHub. Assignments can be linked to a specific course or offered across multiple courses, even in different semesters. Additionally, Web-CAT has a range of analytics features that enable instructors to obtain an overview of student performance in their courses. These features provide valuable insights into the effectiveness of assignments and allow for course improvement based on student feedback and performance. In this section, we describe PEML's integration with web-CAT.

Setting up courses, preparing reference tests, adding testing plug-ins, and configuring assignments are all essential components of using Web-CAT effectively. The system provides a user-friendly interface for instructors to perform these tasks, making it easy for them to get started with automated grading. Additionally, Web-CAT allows graders to manually grade assignments for design, which provides a valuable opportunity for students to receive feedback on their code and improve their understanding of software engineering concepts.

6.2.1 Motivation for PEML Integration

Integrating PEML into Web-CAT presents several benefits that motivated our efforts to create this integration. Firstly, Web-CAT's github-like directory structure for assignments can be overwhelming for new users who need to create complex assignments. PEML provides a simplified and human-readable format that allows users to provide all the necessary information for an assignment, including access to external sources, in a single text file. This ease of use and readability of PEML descriptions makes it easier for instructors to create engaging and challenging programming assignments that are easy to read and understand

as well as share across courses.

Secondly, the automatic test rendering capabilities of PEML from data-driven tests provide a more efficient way of coupling tests with an assignment than the traditional approaches used by Web-CAT. PEML allows for the automatic generation of executable software tests from tabular descriptions, making it easier for instructors to create and grade assignments that have complex input/output requirements. This streamlined process saves time and effort for instructors, while ensuring that students are evaluated based on rigorous and well-defined standards.

Finally, integrating PEML into a widely-adopted tool like Web-CAT has the potential to increase the adoption of PEML itself. This would, in turn, help to provide feedback and enable improvements to the format, leading to better standardization and improved usability. By leveraging the strengths of both Web-CAT and PEML, we hope to improve the programming education experience for both instructors and students.

6.2.2 Design Decisions

Several key design decisions were made when integrating PEML with Web-CAT. Firstly, the decision was made to use the PEML REST API instead of the PEML parser gem. This was because Web-CAT is written in Java using the WebObjects framework, and is not interoperable with the PEML parser gem written in Ruby. The REST API provided a simple way to integrate PEML parsing capabilities into the system as well as access to the entire suite of PEML parser features.

The second design decision was to record PEML exercises as separate entities from the exercises described using Web-CAT's process. This was done to avoid affecting any existing exercise and future-proof any development efforts for new features for PEML exercises. It

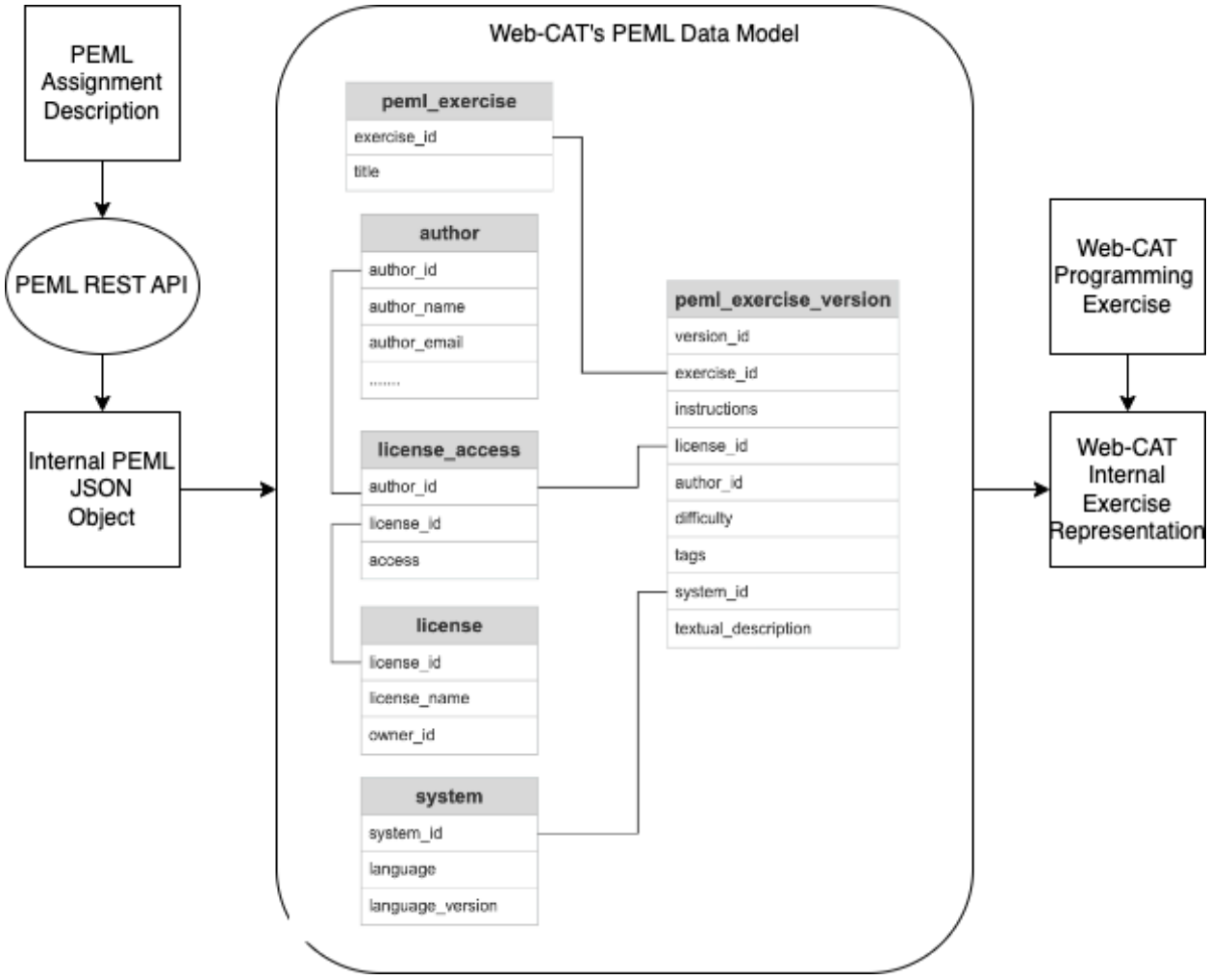


Figure 6.2: PEML Data Model for Web-CAT

also allowed for easier tracking and management of PEML assignments separately from other types of assignments in the system.

Lastly, the UI changes were kept minimal, and the prompt to add a PEML exercise was kept similar to the one for adding other exercises. This was done to provide existing users of Web-CAT an identifiable way to make and edit PEML exercises, without having to re-learn the system or significantly alter their existing workflows. This design decision aimed to minimize disruption for users and encourage the adoption of PEML within the system.

6.2.3 Technical Implementation

To create a PEML assignment in Web-CAT, the user first selects the option to create a new assignment and then chooses to create a PEML-based assignment. This should create a new assignment in the system that is identified as being written in PEML. The user then enters the PEML description into the UI, which is sent to the backend for processing. Unlike traditional Web-CAT assignments, there is a clear distinction between assignments written in PEML and those that are not since they follow two different data models.

Once the PEML description is received on the backend, it is parsed into a JSON object using a REST API call. The JSON object returned is then mapped to a PEML assignment object using the Jackson library's ObjectMapper. Since not all aspects of the PEML data model are relevant to Web-CAT, only those aspects that are important for Web-CAT are included in the mapped object. This includes any files such as test cases, libraries, starter code, and wrapper code that are required for the assignment.

After the assignment object is created, the system generates the files required for the assignment based on the directory path structure specified in the PEML's data model. Any external files that are referenced in the PEML description are also fetched and recorded in the assignment's git repository. This ensures that all the required files are available in the appropriate locations for the assignment to function correctly.

Once the processing is complete, the generated files are committed to the main branch of the assignment's git repository. Any modifications made to the assignment, such as changes to the description or updates to the files, are recorded as new commits on the main branch. If the assignment is used in a different course or semester, a new branch is created to track the usage of the assignment in that context.

If the processing completes successfully, the user is notified and the assignment is made

available for use in the system. If there are any failures during parsing, the user is notified through diagnostic messages that are generated by the parsing step. These messages indicate the specific issues with the PEML description and provide guidance on how to resolve them.

6.2.4 Limitations

The integration of PEML into Web-CAT has also highlighted some limitations. One such limitation, just like with CodeWorkout, is that the internal data model for an assignment in Web-CAT does not have a one-to-one mapping with PEML's data model. This can result in the loss of some information, mainly those related to version control in Web-CAT, when writing assignment descriptions in PEML. To mitigate this loss, we have preserved the textual description of the assignment written in PEML as part of the assignment itself. However, this means that users will need to modify the assignment descriptions written in PEML when trying to make changes which then prompts another parse of the description.

Another limitation that we faced during the integration process is the internal difference between an assignment described with Web-CAT's process and one written using PEML, which exists in the Assignment data model. While CodeWorkout does not make this distinction, we had to make a new UI screen for adding a PEML assignment in Web-CAT. This limitation is a direct result of Web-CAT's architecture and the fact that Web-CAT was not initially designed with PEML in mind.

Chapter 7

PEML Integration Approached for Tools: A Framework

We have demonstrated in the previous chapters that PEML provides a human-friendly way to describe programming assignments in a format that supports automated grading and is apt for tool integration. This chapter provides a framework for integrating PEML into existing automated grading tools or developing new tools around PEML. It covers some ideas for integrating PEML into existing tools as well as some decisions and challenges that developers should be aware of when integrating PEML. By the end of this chapter, readers should have a better understanding of how to incorporate PEML into their own tools and how to navigate the challenges that may arise.

7.1 Integration Strategies

When it comes to integrating PEML into an automated grading tool, there are several strategies that developers can employ. The first strategy is to create a PEML parsing library in the language your tool is written in for native support. However, we do not recommend this approach as it can be time-consuming and requires a significant amount of resources to maintain. Instead, we recommend using the REST API provided by the PEML webservice. This allows for easier integration with existing tools, as developers can simply make HTTP

requests to the parser service to parse PEML descriptions.

Another strategy is to use the parser gem itself which is only feasible if your tool is written in Ruby. If for some reason the tool cannot employ external REST API calls either due to infrastructure challenges or security concern or to ensure zero down time, a Rails wrapper can be written around the parser and hosted on the cluster the tool lives on. Again, this strategy is not recommended and we still suggest utilizing the REST API provided.

A third strategy is to build a new tool from scratch that is specifically designed to support PEML. This approach allows developers to fully leverage the power and flexibility of PEML, as well as tailor the tool to the needs of instructors and students. However, this approach can be time-consuming and requires a significant investment of resources.

Ultimately, the choice of integration strategy will depend on a variety of factors, including the existing infrastructure of the automated grading tool, the needs of instructors and students, and the resources available for development. Developers should carefully consider these factors before selecting an integration strategy, and should be prepared to adapt and iterate their approach as needed.

7.2 Decisions for Integrations

We use this section to provide a flowchart for the decisions that developers will have to make when integrating PEML into a tool. We then describe the flowchart and provide some solutions to common problems that might occur during integration. The first decision is whether to integrate PEML into an existing tool or develop a new tool from scratch. The next decision is to choose which implementation of PEML to use based on the language in which the tool is written. The PEML ruby gem should be used for tools written in Ruby,

while the PEML REST API should be used for tools written in other languages.

The third decision involves modeling the assignment data for the tool. For an existing tool, attempts should be made to match the internal assignment data model to the PEML data model as closely as possible. However, for a tool under development, the PEML data model should be used for assignments. The fourth decision involves choosing which kind of test cases to use with the tool. Data-driven tests, executable tests, or a combination of both can be used.

The fifth decision is whether to use the tool's existing UI or to develop a new UI specifically for PEML. This decision will depend on the complexity of the tool's current UI and the desired level of integration with PEML. Overall, careful consideration of these decisions will be crucial to successfully integrating PEML into a tool or developing a tool around PEML.

Now, moving on to the the common challenges and their solutions:

1. **Handling different data models:** One of the main challenges of integrating PEML into existing tools is reconciling the differences between PEML's data model and the data model used by the target tool. One possible solution is to create a mapping between the two models to ensure that all relevant data is properly transferred.

2. **Dealing with edge cases:** Another challenge of integrating PEML is that there may be certain edge cases or unusual scenarios that are not covered by the standard PEML specification. Edge cases can revolve around encoding non-standard programming assignments, the need to test individual submissions for an assignment differently, or other tool-specific requirements that need to be adhered to. Developers will need to anticipate these cases and build in flexibility to handle them appropriately.

3. **Adapting to tool-specific requirements:** Different automated grading tools may have different requirements for how assignments are structured and formatted. In-

tegrating PEML into these tools may require some additional work to ensure that assignments created in PEML meet these requirements. These requirements can span from assignment description size to how different vendor specific keys are read.

4. **Testing and debugging:** As with any integration project, testing and debugging are critical components of ensuring that the integration is successful. Developers will need to have a robust testing plan in place to catch any errors or issues that arise during the integration process.

5. **User adoption:** Finally, a major challenge of integrating PEML into existing tools is ensuring that users are able to easily adopt the new format. Developers should focus on creating a user-friendly interface and providing clear documentation to help users understand the benefits of using PEML and how to work with it effectively.

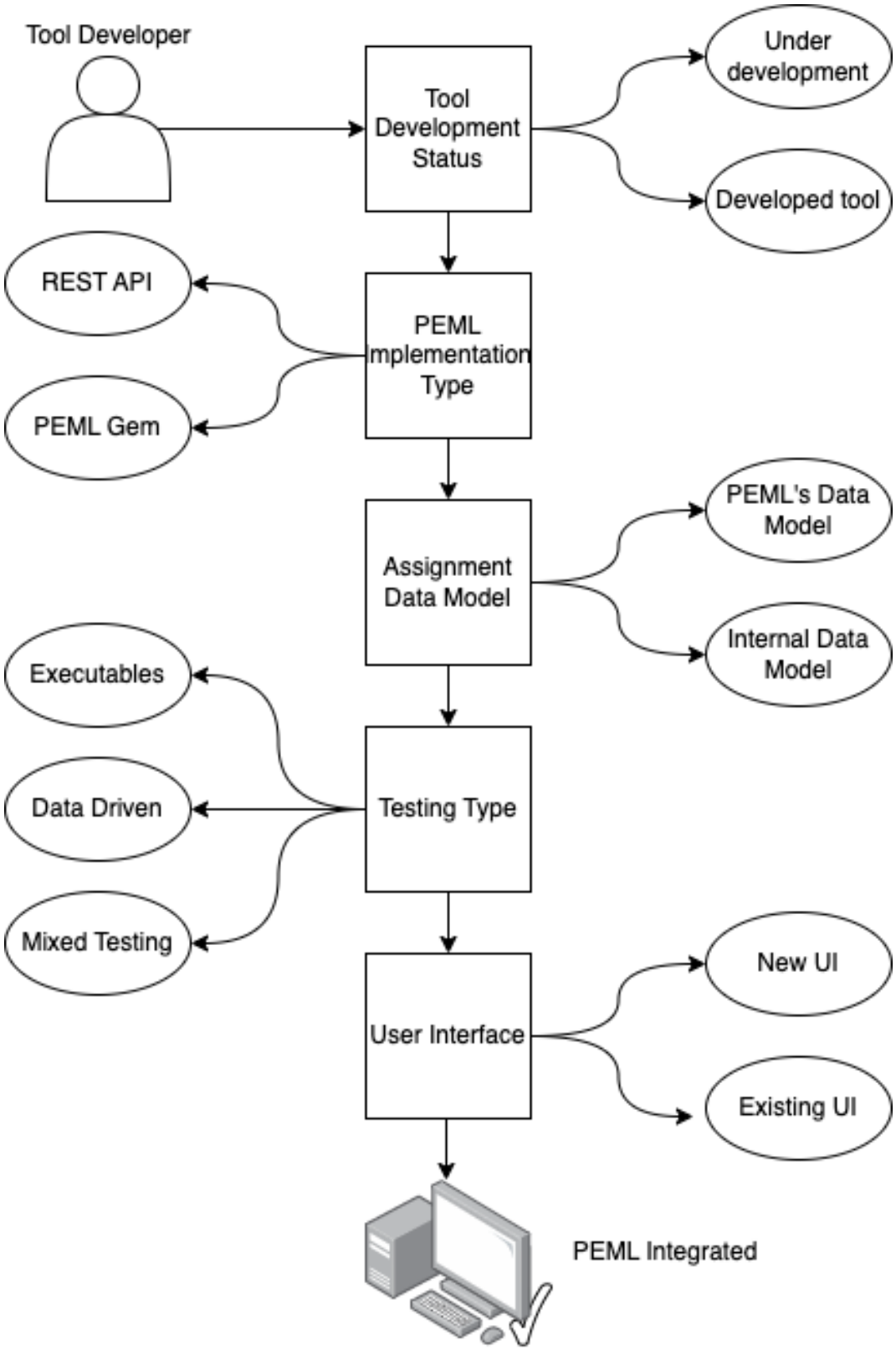


Figure 7.1: Decisions That Need to be Made When Integrating PEML

Chapter 8

PEML's Evaluation

With the rapid adoption of automated grading tools in computer science education, the need for a standardized format for describing programming assignments and making them available to other instructors as well as diverse set of automated grading tools is becoming increasingly important. In this chapter, we describe our evaluation of the suitability and feasibility of PEML to be this standardized notation format.

We will discuss a set of approaches, starting out with describing our methods, following up with a feasibility study of encoding 60 PEML descriptions, an analysis of the diagnostic generation capabilities of PEML, and finally, the results of our parsed tests being able to validate candidate solutions. The goal of this evaluation is to find out if PEML can be feasibly used to encode a wide selection of programming assignments in an instructor-friendly manner as well as how PEML's programming language-independent nature affects test generation.

We first discuss our methods as well as examine the research questions posed in Chapter 1 in more detail. We then developed a set of criteria for evaluating the suitability and feasibility of PEML and applied them to our evaluation.

Next, we focus on the feasibility study of encoding 60 PEML descriptions. Our objective was to evaluate whether authors could use PEML to describe a diverse range of programming assignments, including assignments that require the use of external resources, such as data files and libraries. We provide a detailed analysis of the feasibility study, including the

challenges we encountered and the solutions we developed.

In this section, we discuss our experience working with PEML, the PEML parser, as well as the PEML web Service. In the following subsections, we touch upon our method for this part of the paper, our experience converting existing assignment descriptions to PEML, our experiences working with the parser, and finally, the web service, particularly its ability to generate executable test cases for use in automated grading tools.

The third section discusses the diagnostic generation capabilities of PEML. We examine the error messages generated by PEML when encoding the 60 PEML descriptions and analyze the types of errors that were encountered. We also discuss the effectiveness of the error messages generated by PEML in helping authors identify and resolve errors in their descriptions.

The fourth section presents the results of our parsed tests being able to validate candidate solutions. We used a script to parse all the encoded assignment descriptions and ran them through the parser to generate executable tests from data-driven tests to evaluate whether the generated tests were able to validate candidate solutions. We describe the test suites used in our evaluation and provide a detailed analysis of the results.

Finally, in the last section of this chapter, we evaluate whether we were able to answer all the research questions that we initially asked. We examine whether the results of our evaluation support the suitability and feasibility of PEML as a standardized notation for describing programming assignments.

8.1 Method

To evaluate the feasibility and suitability of using PEML, we began by selecting a representative sample of CS1 programming assignments, which spanned a range of programming

topics, difficulty levels, and classroom activities. We aimed to include a diverse set of assignments that covered different programming paradigms, programming languages, and types of programming problems. This allowed us to assess the effectiveness of PEML across a wide range of use cases, as well as to identify any limitations or areas for improvement.

We selected 60 problems from three kinds of activities. We chose 50 short-answer practice exercises, which most students answer in a few minutes each as part of homework. We also selected 5 lab assignments, which take a few hours each. Then we added 5 larger out-of-class programming assignments, which students work on for several days or a week. All were taken from the CS1 course at Virginia Tech.

After selecting the programming assignments, we proceeded to generate PEML descriptions for each of them. This involved a thorough review and analysis of the original assignment specifications and requirements, as well as an understanding of the underlying programming concepts and constructs. We then manually wrote and reviewed PEML descriptions for each assignment, trying our best to match the programming assignment description to the PEML data model.

Our evaluation process included a thorough examination of the feasibility of encoding 60 PEML descriptions, as well as an analysis of any errors or issues that arose during the encoding process. We also conducted tests to validate candidate solutions generated from the PEML descriptions, to ensure that they accurately reflected the desired programming tasks and could be successfully graded using automated tools.

Overall, our evaluation of PEML involved a comprehensive assessment of its effectiveness, efficiency, and accuracy in encoding programming assignment descriptions that are human-friendly to write and suitable for automated grading.

All PEML descriptions from this study are publicly available [19]. Table 8.2 shows examples

Name	Description	Type
Hello Cat	A "hello world" program that produces fixed output	Class Exercise
Modeling a Microwave Oven	Using inheritance to make multiple subclasses	Class Exercise
The Pig Game	A dice game with a text UI and computerized opponent	Class Exercise
Talk Like a Pirate	A text I/O program to turn input strings into pirate talk	Class Exercise
Shape Maker	A graphical drawing editor with a Java Swing UI	Class Exercise
Mine Sweeper	Using a 2D array as a game board for a graphical UI	Class Exercise
Time Table	A lab assignment practicing with 2D arrays	Lab Exercise
Counting Lines	A lab assignment practicing text input using scanners	Lab Exercise
Building a Gradebook	Practice with maps and lists	Class Exercise
Spelling Checker	A lab assignment practicing using maps and text input	Lab Exercise
Rot-13 Decryption	A lab assignment practicing string manipulation	Lab Exercise
arrayListCodingPractice	small exercise practicing with lists	CodeWorkout Exercise
arraysFindOnes	small exercise practicing with 1D arrays	CodeWorkout Exercise
binarySearchNonRecursive	small exercise practicing binary search	CodeWorkout Exercise
callingSuperPractice2	small exercise practicing using 'super'	CodeWorkout Exercise
genericsComparableCage.peml	small exercise practicing using generics	CodeWorkout Exercise
implementAConstructor	small exercise practicing writing constructors	CodeWorkout Exercise
	[44 other small exercises not shown]	

Table 8.1: A subset of the 60 assignments used in the feasibility study.

from the problems selected.

8.2 Experience Writing PEML Descriptions

During the encoding phase, we encountered some initial difficulties when creating PEML descriptions using plain text editors without syntax highlighting. As many programmers are accustomed to using integrated development environments (IDEs) or other tools that provide syntax highlighting and suggestions, the lack of such support in the initial stages of PEML encoding can feel visually jarring. While other formats such as JSON and YAML readily provide this kind of support, it was straightforward to create custom editing mode definitions for tools like BBEdit and SublimeText to support PEML.

One of the notable aspects of PEML that emerged during the encoding phase was the reusability of templates. For CS1-level assignments in a our course, since they were written by a small number of people, there is a consistent structure comprising of assignment in-

structions, starter or wrapper code, test cases, and course-specific identifiers. These elements can be provided inline, as references to external file locations, or using URLs. PEML's convention over configuration approach facilitated the use of a generic PEML template, which enabled the creation of new assignments by simply plugging in new instructions, test cases, and starter and wrapper code inline or by using a predefined directory structure. This streamlined the generation of PEML descriptions so that, after completing a few, it became quick and easy to create many more. While other instructors might not use our precise conventions, they are likely to have their own which PEML templates will be able to support.

In addition, the PEML parser's ability to render markdown to HTML was a useful feature. This functionality proved particularly helpful when using PEML descriptions for online tutoring and grading tools that provide students with an IDE-like environment to solve questions and receive feedback. By using git-flavored markdown for instructions, the PEML parser automatically converts them to HTML, which can be used seamlessly in grading tools. On-the-fly variable interpolation was another helpful feature of PEML that allowed us to dynamically generate and use variables in our assignments.

Through our evaluation of PEML, we found that it provided a flexible and reusable structure for encoding CS1-level assignments. The use of templates and convention over configuration facilitated the creation of new PEML descriptions, which could be used across different programming topics and difficulty levels. Additionally, the rendering of markdown to HTML and on-the-fly variable interpolation features provided added benefits in online tutoring and grading environments.

Overall, our experience with PEML demonstrated its suitability as a tool for encoding assignments and making them accessible in different contexts, while also simplifying the creation and management of such assignments.

8.3 PEML Errors and Diagnostics

During our evaluation of PEML, we encountered the need to address syntax errors that arose when writing new documents. The PEML parser provided a helpful feature where it returned the parsed description along with a list of diagnostic messages generated. As with any notation, the occasional syntax error is to be expected. However, unlike typical programming languages, true syntax errors were found to be nearly absent. Instead, diagnostics resulted from validating the data representation and identifying required keys that were absent or improperly structured nested fields.

During the feasibility study, we recorded and evaluated all diagnostic messages received. The most common diagnostic message pointed to a missing author key. This was observed to be the case for a group of the short programming exercises used, where author names were omitted. Another frequent diagnostic message identified externally referenced files that were missing, due to their being out of place.

A less frequently occurring diagnostic message was about strings in test cases being incorrectly formatted when test input/output values were provided as traditional CSV files, where quoting values by hand can be error-prone. Fortunately, PEML provides an alternate tabular format for data-driven descriptions of test cases that is more human-friendly than CSV. This format allows full programming language expressions and quoting styles, which reduces the risk of syntax errors.

The PEML parser's ability to immediately provide diagnostic messages upon parsing was immensely beneficial. By receiving these messages in real-time, we were able to identify any errors pointed out and fix the descriptions before proceeding. This streamlined the process and made it possible for us to ensure that all PEML descriptions were free of syntax errors.

Diagnostics	Occurrence
“The document is missing the key: author“	5
“The document is missing the key: authors“	5
“The document is missing the key: license“	4
“Path ‘/license/id‘ is not a string“	3
“‘/systems‘ is not an array (has the wrong structure)“	1
“Path ‘/version‘ does not have sub-keys (has the wrong structure)“	1

Table 8.2: Commonly Occurring Diagnostics

8.4 Unit Testing Assignments

To evaluate the feasibility of using PEML descriptions to generate executable software tests for grading student submissions, we conducted tests using a variety of programming languages. In our feasibility study, we found that many CS1 assignments come with embedded unit tests that are often specified using input/output values in columns of CSV data or in software test frameworks like JUnit. However, many automated grading tools require that tests be specified in a different format that can be used to grade student submissions in a way that is programming-language-agnostic.

PEML provides a flexible and extensible column format for describing tabular forms of unit tests, which can be used to generate executable software tests in a programming-language-agnostic way. Our goal was to evaluate the feasibility of generating these executable software tests from PEML descriptions in a way that is compatible with a variety of programming languages. We tested this approach with Java, Python, and C++, and we found that the PEML parser was able to generate test cases that were able to compile and run successfully.

To validate the feasibility of this approach, we used a script to parse the PEML descriptions using the REST service and extracted the generated executable software tests from the result. These test cases are generated through variable interpolation of values from the inlined data table into language-specific Liquid templates. We then compiled the tests to ensure they

were rendered successfully. By measuring if all the test cases compile and run successfully and if we were able to validate solutions in a way that a manual grader would, we were able to determine the feasibility of using PEML to generate executable software tests for grading student submissions.

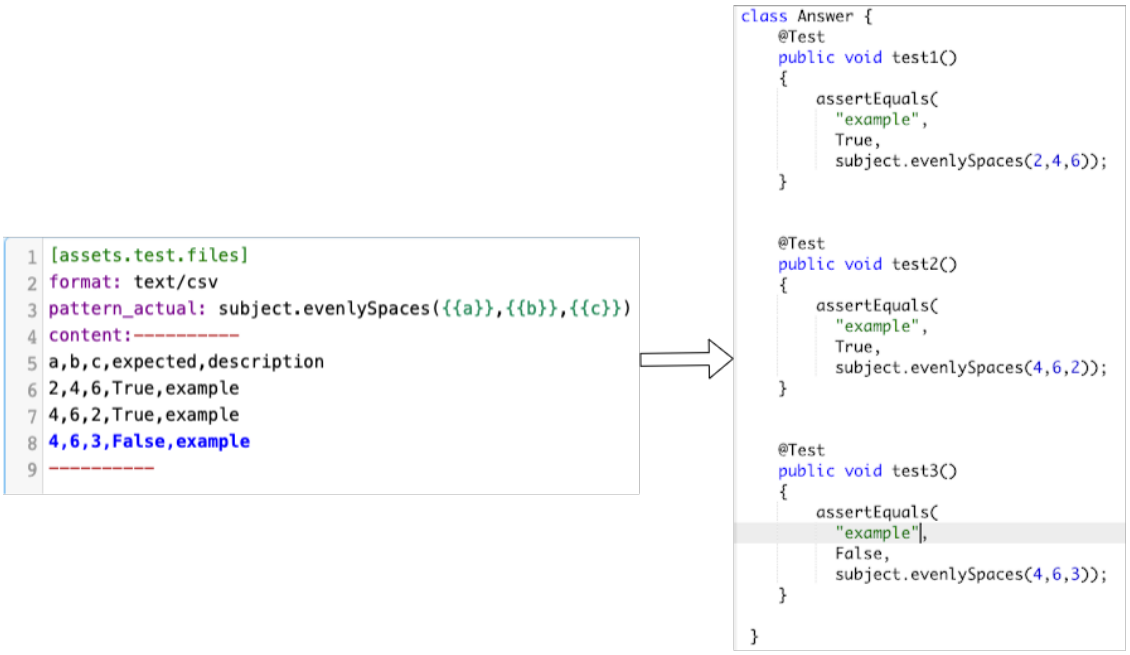


Figure 8.1: Data Driven to Executable Test Rendering

8.5 Answers to Research Questions

The evaluation performed in the previous sections provide the information necessary to answer the research questions we posed in Chapter 1. Each of these questions along with findings and analysis associated with their objectives are detailed below:

8.5.1 RQ1: Types of Describable Assignments

What types of programming assignments can be described using PEML?

PEML can be used to describe a wide variety of programming assignments. This can range from small programming exercises designed to be solved by a single student over the duration of a single class to those spanning complex assignments supposed to be worked on in groups and take weeks to finish. The assignments can range from requiring a single function to those requiring complex collections of classes and sub-classes. The assignments can be simple menu driven stdin/stdout programs to complex GUI driven programs too.

We show this through our feasibility study where we encode 60 assignments from CS-1 courses at Virginia Tech. We also discuss any challenges we faced along the way which can serve as a good list of things to look out for when encoding your own repository of exercises into PEML. Overall, PEML is a versatile programming assignment specification format and can serve to encode a wide variety of programming assignments.

8.5.2 RQ2: Feasibility of integrating PEML into tools

Is it feasible to integrate PEML into a variety of educational tools?

A format for writing programming assignment descriptions that is easy to integrate into a wide variety of educational tools implemented in different programming languages and application frameworks requires certain consideration. Firstly, it should be easy to learn and use and should not be a notation barrier. Secondly, it should have several implementations that allow it to be integrated into tools written in a variety of different programming languages and application frameworks. Lastly, it should cover most use cases and their variations for describing assignments that instructors perform when preparing them.

Through our discussion of PEML's integration into CodeWorkout and Web-CAT as well as our discussion about the integration framework, we have shown that it is indeed feasible to integrate PEML into auto-grading tools. Furthermore, we provide several implementa-

tions of PEML that allow it to be integrated into most automated grading tools. Lastly, the framework provides a discussion of any challenges that might come up during the integration process as well as how to overcome them. Thus, PEML is a feasible format for writing programming assignment descriptions that can be integrated into a wide variety of educational tools.

8.5.3 RQ3: Programming Language-Independent Nature of PEML

Is it feasible to generate software tests in different programming languages from the same PEML description leveraging PEML's programming-language-agnostic nature?

The programming language-independent nature of PEML greatly impacts the feasibility of using it as an assignment description language for automated grading and tool integration. Since PEML does not rely on any particular programming language, it can be used to describe assignments across multiple programming languages, making it an incredibly versatile tool for instructors and developers. Our automatic unit test generation for assignments in different programming languages from the same description during the feasibility study proves that PEML can indeed be used effectively in a language-agnostic manner for automated grading.

Additionally, the language-independent nature of PEML also makes it easier for instructors to use in different contexts, such as when teaching a course that focuses on the usage of different programming languages or when using different language-focused grading tools for different courses with a single version of a PEML description. Overall, the use of a PEML's programming language independent nature reduces the complexity of writing assignment descriptions, thus, streamlining the assignment development process for instructors.

Chapter 9

Conclusions

In this thesis, we have described PEML which has been specifically designed to provide instructors with a human-friendly way to write, maintain, and reuse assignment descriptions that can be directly used by auto-grading tools. We have also presented, in detail, the design and implementation of PEML. We have demonstrated how PEML can simplify the process of creating and sharing programming assignments while also providing a flexible and extensible structure for defining assignments that can be tailored to meet the needs of instructors as well as automated grading tools.

Based on our evaluation, we have established that PEML is a practical and efficient approach for outlining programming assignments that facilitate automated grading and tool integration. Our assessment highlights the potential advantages of PEML for instructors, educators, and students in terms of improved productivity, uniformity, and quality. Additionally, we have recognized some limitations and obstacles that need to be addressed to ensure widespread adoption.

Furthermore, we have shown PEML being integrated into two automated grading tools as well as described how we went about integrating it and the challenges we faced along the way. Building upon this, we have also provided an integration framework for PEML, which allows it to be integrated into existing tools and workflows used by instructors and educators for teaching programming.

In this concluding chapter, we summarize the contributions of this work and discuss potential avenues for future research and development of PEML. We believe that PEML has the potential to make a significant impact on programming education and we hope that our work will inspire others to continue exploring the potential of this approach.

9.1 Contributions

Our work has made several several contributions to the field of computer science education. One of the key contributions of our work is the development of PEML, a programming assignment specification format that is designed to support automated grading and tool integration. Through our evaluation of PEML, we have shown that it is a feasible and effective approach for describing programming assignments in a way that supports automated grading and tool integration.

We have also contributed a PEML parser library implemented in Ruby that is publicly available as an open-source resource. This library provides a comprehensive suite of features that can be used to parse and interpret PEML files, including support for variable interpolation, markdown rendering, and automatic test case generation. Additionally, we have developed a REST API that provides an easy and robust way to access PEML parsing capabilities from any tool or scripting language. This API supports JSON, YAML, or XML representations of parsed contents, making it easier for developers to integrate PEML into their existing tools and workflows.

Another contribution of our work is the development of a user-facing website that serves as a ”PEML playground” for authors to experiment with and learn PEML. This website provides direct user-level access to JSON, YAML, or XML results of parsed descriptions, allowing authors to quickly iterate on their assignments and view the results in real-time.

Additionally, we have created a repository of exercises ranging in size and difficulty encoded in PEML that can be used by instructors as examples when developing their own assignments in PEML.

Our work has also contributed to the field of computer science education by providing an experience report covering the challenges and benefits encoding 60 exercises into PEML as well as integrating PEML into two existing tools at Virginia Tech: Web-CAT and Code Workout. This report details the process of encoding exercises of varying sizes and difficulties, providing a run through for any instructor planning to switch to PEML. Furthermore, integrating PEML into the two tools mentioned above provides insights into the challenges and benefits of doing so. Additionally, we have developed a framework for integrating PEML into other automated grading tools, building upon our experiences from the previous point. This framework provides guidance for developers who are interested in integrating PEML into their own grading tools, helping to reduce the development effort required and ensure a smoother integration process.

Overall, our work has made several significant contributions to the field of computer science education, including the development of a new programming assignment specification format (PEML), a PEML parser library, a REST API for accessing PEML parsing capabilities, a user-facing website for experimenting with and learning PEML, a repository of example PEML-encoded exercises, and an experience report and integration framework for integrating PEML into automated grading tools. These contributions have the potential to significantly impact the way programming assignments are developed, shared, and graded in the future, making the process more efficient, consistent, and scalable for educators and students alike.

9.2 Future Work

As with any new technology or tool, there is always room for improvement and further development. In this section, we discuss potential future directions PEML can take, building on the work presented in this paper. We identify areas where further research and development could improve the functionality, flexibility, and accessibility of PEML, as well as its adoption by educators and instructors. These suggestions are not meant to be exhaustive or prescriptive, but rather to inspire and guide future work on PEML.

1. **PEMLtest:** While the current suite of available approaches to define tests covers most of the use-cases, in the future, we plan on developing PEMLtest, a DSL designed to express a wide variety of software tests in a more concise, and lighter weight manner than XUnit programming. This DSL will be able to support tests for different programming languages, and test specifications can be translated down to executable XUnit-style tests in the target language.

2. **Further refinement of the PEML specification:** While we have developed a comprehensive specification for PEML, there is always room for improvement and refinement. Future work could focus on expanding the capabilities of PEML to better support more complex programming assignments, such as those found in upper-level undergraduate and graduate courses or exercises in other areas such as data science, machine learning, and game development. Additionally, refinement of the PEML specification could focus on improving the user experience for instructors and educators by providing additional guidance and documentation to help them more easily create and use PEML descriptions.

3. **Improving the PEML parser's diagnostic capabilities:** During our evaluation, we observed that the PEML parser generates diagnostic messages for common errors,

but there is room for improvement in terms of identifying more specific errors and offering suggestions for how to correct them. Further research and development could be devoted to enhancing the parser's error detection capabilities to provide more informative feedback to authors when errors are detected.

4. **Integration with additional tools and platforms:** While we have demonstrated the feasibility of integrating PEML with a few existing tools, there are many other tools and platforms used in programming education that could benefit from PEML integration. Future work could focus on developing integration frameworks and guidelines for these tools, and providing additional support and resources to developers looking to integrate PEML into their own tools and platforms.

5. **Tool-specific diagnostics:** While the PEML parser can generate diagnostics for when the description fails to follow PEML's internal data model, there might be tools which have their own constraints. In this case, it would be helpful for instructors to receive diagnostics when they violate these tool-specific constraints. This can be achieved in the future by using XPATH (or JSONPath) to allow tools to define custom constraint-checking rules for presence/absence of keys. In fact, it would likely be possible to use existing libraries for JSONPath to interpret and apply such checks on a post-parsed PEML data model. We can also allow a collection of constraints from a specific tool to be provided as a JSON upload parameter to the REST API in the future.

6. **Hierarchically templated tests:** Another avenue for future work is the use of hierarchical testing templates for unit test rendering. Currently tests are rendered through substituting values for individual tests and then collecting these tests and putting them in a class but in the future we can have an implementation where templates exist at different levels (statement, function, class, etc.) and template, post substitution, is

fed into a template at the next higher level. This will allow for more granular control when rendering tests from a collection of values.

7. **Development of new tools and resources for PEML:** While we have developed a parser, web service, and user-facing website for PEML, there are many other tools and resources that could be developed to support the adoption and use of PEML in programming education. For example, a PEML authoring tool that provides a graphical interface for creating and editing PEML descriptions could help streamline the process of creating new assignments. Additionally, enhancements to the repository of pre-made PEML descriptions, such as templates for common programming tasks, could help new instructors get started with PEML more easily.

8. **Expansion of the PEML community:** While we have demonstrated the potential benefits of PEML for programming education, the success of PEML ultimately depends on the adoption and use of the format by instructors, educators, and students. Future work could focus on expanding the PEML community by providing additional training and support resources for instructors and educators, as well as promoting the use of PEML through conferences, workshops, and other events. Additionally, further research could be done to measure the impact of PEML on student learning outcomes and instructor efficiency, to better understand the potential benefits and limitations of the format.

Appendices

Appendix A

PEML's Data Model

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://CSSPLICE.github.io/peml/schemas/PEML.json",
  "title": "PEML",
  "description": "This schema defines the data model for PEML, the Program
      Exercise Markup Language (https://cssplice.github.io/peml/). The data
      model is intended to be represented in PEML notation, but the same data
      model can easily be expressed in JSON or YAML or any similar structured
      data format, so this serves as a JSON Schema for the common data model
      behind the content.",

  "definitions": {
    "nonempty_string": {
      "$id": "nonempty_string",
      "type": "string",
      "minLength": 1
    },
    "string_no_whitespace": {
      "$id": "string_no_whitespace",
      "$ref": "#/definitions/nonempty_string",
      "pattern": "^[^\\s]+$"
    },
    "boolean": {
      "$id": "boolean",
```

```
20        "oneOf": [
21          { "type": "boolean" },
22          { "type": "integer",  "minimum": "0", "maximum": "1" },
23          {
24            "type": "string",
25            "pattern": "^(true|True|TRUE|false|False|FALSE|yes|Yes|YES|no|No|NO|
                  on|On|ON|off|Off|OFF|0|1)$"
26          }
27        ]
28      },
29      "id": {
30        "$id": "id",
31        "description": "An id is a non-empty string containing no whitespace and
              no commas (which might be used as separators in strings denoting
             lists of ids).",
32        "$ref": "#/definitions/nonempty_string",
33        "pattern": "^[^\\s,]+$"
34      },
35      "id_list": {
36        "$id": "id_list",
37        "description": "Either a single id (no spaces or commas) or a comma-
             separated (or space-separated) list of multiple ids.",
38        "$ref": "#/definitions/nonempty_string",
39        "pattern": "^[^\\s,]+(\\s*[, ]\\s*[^\\s,]+)*$"
40      },
41      "timestamp": {
42        "$id": "timestamp",
43        "description": "A human-readable timestamp indicating the time at which
             this version of the exercise was last modified. For lack of a better
              option, at the moment this should be an RFC 3339/ISO 8601 UTC
             timestamp (if you know of something more user-friendly but equally
```

```
        unambiguous, let us know!). That format is: YYYY-MM-DDThh:mm:ss.
        nnn±hh:mm. See https://stackoverflow.com/questions/28020805/regex-
        validate-correct-iso8601-date-string-with-time for the regex info.",
44      "$ref": "#/definitions/nonempty_string",
45      "format": "date-time"
46    },
47    "email_address": {
48      "$id": "email_address",
49      "description": "An email_address is a non-empty string conforming to the
            idn-email format.",
50      "$ref": "#/definitions/nonempty_string",
51      "format": "idn-email"
52     },
53    "person": {
54      "$id": "person",
55      "description": "Represents an individual, including a unique email
          address and an optional name",
56      "oneOf": [
57        { "$ref": "#/definitions/email_address" },
58        {
59          "type": "object",
60          "required": ["email"],
61          "properties": {
62            "email": { "$ref": "#/definitions/email_address" },
63            "name": { "$ref": "#/definitions/nonempty_string" }
64          }
65        }
66      ]
67    },
68    "tag_list": {
69      "$id": "tag_list",
```

```
70        "description": "Can be either a single string representing a list of
              semi-colon-delimited or comma-delimited tags/elements, or an array
              of strings representing the same content.",
71        "oneOf": [
72          { "$ref": "#/definitions/nonempty_string" },
73          {
74            "type": "array",
75            "items": { "$ref": "#/definitions/nonempty_string" },
76            "minItems": 1
77          }
78        ]
79      },
80      "location": {
81        "$id": "location",
82        "description": "A string that may also be a url(...) defining an
              addressable location.",
83        "$ref": "#/definitions/nonempty_string"
84      },
85      "relative_location": {
86        "$id": "relative_location",
87        "description": "A string that intended to define a relative location,
              probably using a url(...).",
88        "$ref": "#/definitions/location"
89      },
90      "file": {
91        "$id": "file",
92        "oneOf": [
93          { "$ref": "#/definitions/location" },
94          {
95            "type": "object",
96            "required": ["content"],
```

```
97            "properties": {
98              "content": {
99                "oneOf": [
100                 { "type": "string" },
101                 { "type": "array" },
102                 { "type": "object" }
103               ]
104             },
105             "name": { "$ref": "#/definitions/nonempty_string" },
106             "type": { "$ref": "#/definitions/nonempty_string" },
107             "content_encoding": { "$ref": "#/definitions/nonempty_string" }
108           }
109         }
110       ]
111     },
112     "file_list": {
113       "$id": "file_list",
114       "oneOf": [
115         { "$ref": "#/definitions/location" },
116         {
117           "type": "array",
118           "items": { "$ref": "#/definitions/file" },
119           "minItems": 1
120         }
121       ]
122     },
123     "repository": {
124       "$id": "repository",
125       "type": "object",
126       "required": ["url"],
127       "properties": {
```

```
128         "url": { "$ref": "#/definitions/location" },
129         "path": { "$ref": "#/definitions/nonempty_string" },
130         "branch": { "$ref": "#/definitions/nonempty_string" },
131         "tag": { "$ref": "#/definitions/nonempty_string" }
132       }
133     },
134     "suite": {
135       "$id": "suite",
136       "oneOf": [
137         { "$ref": "#/definitions/location" },
138         {
139           "type": "object",
140           "anyOf": [
141             { "required": ["content"] },
142             { "required": ["cases"] }
143           ],
144           "properties": {
145             "content": {
146               "oneOf": [
147                 { "type": "string" },
148                 { "type": "array" },
149                 { "type": "object" }
150               ]
151             },
152             "name": { "$ref": "#/definitions/nonempty_string" },
153             "type": { "$ref": "#/definitions/nonempty_string" },
154             "content_encoding": { "$ref": "#/definitions/nonempty_string" },
155             "visibility": { "$ref": "#/definitions/nonempty_string" },
156             "pattern": { "type": "object" },
157             "template": { "$ref": "#/definitions/nonempty_string" },
158             "cases": { "type": "array" }
```

```
159             }
160           }
161         ]
162       },
163       "suite_list": {
164         "$id": "suite_list",
165         "oneOf": [
166           { "$ref": "#/definitions/location" },
167           {
168             "type": "array",
169             "items": { "$ref": "#/definitions/suite" },
170             "minItems": 1
171           }
172         ]
173       },
174       "environment": {
175         "$id": "environment",
176         "type": "object",
177         "properties": {
178           "inherits": { "enum": ["start", "build", "run"] },
179           "files": { "$ref": "#/definitions/file_list" },
180           "repository": { "$ref": "#/definitions/repository" },
181           "image": { "$ref": "#/definitions/nonempty_string" },
182           "registry": { "$ref": "#/definitions/location" }
183         }
184       },
185       "environment_list": {
186         "$id": "environment_list",
187         "type": "object",
188         "properties": {
189           "start": { "$ref": "#/definitions/environment" },
```

```
190         "build": { "$ref": "#/definitions/environment" },
191         "run":   { "$ref": "#/definitions/environment" },
192         "test":  { "$ref": "#/definitions/environment" }
193       }
194     },
195     "solution": {
196       "$id": "solution",
197       "oneOf": [
198         { "$ref": "#/definitions/location" },
199         {
200           "type": "object",
201           "properties": {
202             "name": { "$ref": "#/definitions/nonempty_string" },
203             "description": { "type": "string" },
204             "visibility": { "$ref": "#/definitions/nonempty_string" },
205             "correct": { "$ref": "#/definitions/boolean" },
206             "reference": { "$ref": "#/definitions/boolean" },
207             "files": { "$ref": "#/definitions/file_list" }
208           }
209         }
210       ]
211     },
212     "solution_list": {
213       "$id": "solution_list",
214       "oneOf": [
215         { "$ref": "#/definitions/location" },
216         {
217           "type": "array",
218           "items": { "$ref": "#/definitions/solution" },
219           "minItems": 1
220         }
```

```
      ]
    },
    "system": {
      "$id": "system",
      "type": "object",
      "properties": {
        "language": { "$ref": "#/definitions/nonempty_string" },
        "version": { "$ref": "#/definitions/nonempty_string" },
        "environment": { "$ref": "#/definitions/environment_list" },
        "suites": { "$ref": "#/definitions/suite_list" },
        "src": {
          "type": "object",
          "properties": {
            "files": { "$ref": "#/definitions/file_list" },
            "starter": {
              "type": "object",
              "required": ["files"],
              "properties": { "files": { "$ref": "#/definitions/file_list" } }
            },
            "frame": {
              "type": "object",
              "required": ["files"],
              "properties": { "files": { "$ref": "#/definitions/file_list" } }
            },
            "solutions": { "$ref": "#/definitions/solution_list" }
          }
        }
      }
    },
    "system_list": {
      "$id": "system_list",
```

```
252       "type": "array",
253       "items": { "$ref": "#/definitions/system" },
254       "minItems": 1
255     }
256   },
257
258   "type": "object",
259
260   "required": ["exercise_id", "title"],
261   "allOf": [
262     { "anyOf": [
263       { "required": ["instructions"] },
264       { "required": ["suites"] },
265       { "required": ["systems"] }
266     ] },
267     { "anyOf": [
268       { "required": ["author"] },
269       { "required": ["authors"] },
270       { "required": ["license"] }
271     ] }
272   ],
273
274   "properties": {
275     "exercise_id": { "$ref": "#/definitions/id" },
276     "title": { "type": "string" },
277     "author": { "$ref": "#/definitions/person" },
278     "authors": {
279       "description": "An array of multiple authors.",
280       "type": "array",
281       "items": { "$ref": "#/definitions/person" },
282       "minItems": 1
```

```
283     },
284     "tag": {
285       "type": "object",
286       "properties": {
287         "topics": { "$ref": "tag_list" },
288         "prerequisites": {
289           "oneOf": [
290             { "$ref": "tag_list" },
291             {
292               "type": "object",
293               "properties": {
294                 "exposure": { "$ref": "tag_list" },
295                 "familiarity": { "$ref": "tag_list" },
296                 "mastery": { "$ref": "tag_list" }
297               }
298             }
299           ]
300         },
301         "style": { "$ref": "tag_list" },
302         "course": { "$ref": "tag_list" },
303         "book": { "$ref": "tag_list" },
304         "personal": { "$ref": "tag_list" }
305       }
306     },
307     "version": {
308       "type": "object",
309       "properties": {
310         "timestamp": { "$ref": "#/definitions/timestamp" },
311         "type": { "$ref": "#/definitions/nonempty_string" },
312         "id": { "$ref": "#/definitions/nonempty_string" },
313         "repository": { "$ref": "#/definitions/repository" },
```

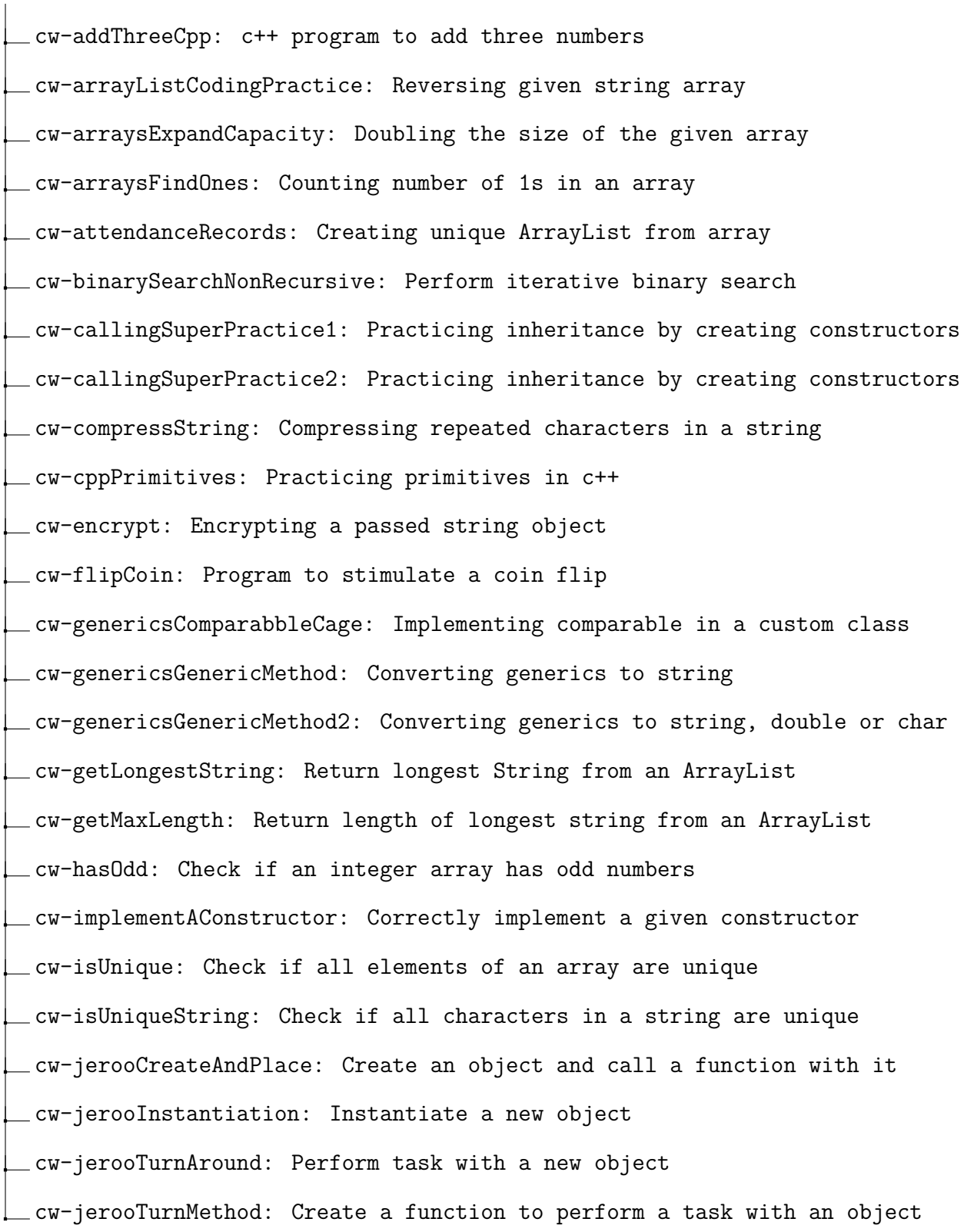
```
314         "location": { "$ref": "#/definitions/relative_location" }
315       }
316     },
317     "license": {
318       "type": "object",
319       "required": ["id", "owner"],
320       "properties": {
321         "id": { "$ref": "#/definitions/nonempty_string" },
322         "owner": { "$ref": "#/definitions/person" },
323         "book": { "$ref": "#/definitions/nonempty_string" },
324         "attribution": { "$ref": "#/definitions/nonempty_string" },
325         "acknowledgements": { "type": "string" },
326         "acknowledgments": { "type": "string" }
327       }
328     },
329     "difficulty": {
330       "type": "integer",
331       "minimum": 0,
332       "maximum": 100
333     },
334     "instructions": { "type": "string" },
335     "public_html": { "$ref": "#/definitions/file_list" },
336     "environment": { "$ref": "#/definitions/environment_list" },
337     "suites": { "$ref": "#/definitions/suite_list" },
338     "systems": { "$ref": "#/definitions/system_list" }
339   }
340 }
```

Appendix B

List of PEML Descriptions from the Feasibility Study

In this chapter, we provide the complete list of PEML descriptions written as part of our feasibility study in the form of a directory tree representing the repository. This was shown in brief in the table from our Evaluation chapter under section 8.1. The github repository for this collection can be found here.

```
laboratory-exercises
    Modelling a Microwave Oven: Inheritance to make multiple subclasses
    Time Table: Practicing with 2D arrays
    Counting Lines: Practicing text input using scanners
    Building a Gradebook: Practicing maps and lists
    Spelling Checker: Practicing maps and text inputs
    Rot-13 Decryption: Performing string manipulation
project-exercises
    The Pig Game: 2-player dice game
    Talk Like a Pirate: I/O program to perform string manipulation
    Hello Cat!: Basic "hello world" like program
    Mine Sweeper: Using 2D array as a game board
small-exercises
```


cw-addThreeCpp: c++ program to add three numbers
cw-arrayListCodingPractice: Reversing given string array
cw-arraysExpandCapacity: Doubling the size of the given array
cw-arraysFindOnes: Counting number of 1s in an array
cw-attendanceRecords: Creating unique ArrayList from array
cw-binarySearchNonRecursive: Perform iterative binary search
cw-callingSuperPractice1: Practicing inheritance by creating constructors
cw-callingSuperPractice2: Practicing inheritance by creating constructors
cw-compressString: Compressing repeated characters in a string
cw-cppPrimitives: Practicing primitives in c++
cw-encrypt: Encrypting a passed string object
cw-flipCoin: Program to stimulate a coin flip
cw-genericsComparabbleCage: Implementing comparable in a custom class
cw-genericsGenericMethod: Converting generics to string
cw-genericsGenericMethod2: Converting generics to string, double or char
cw-getLongestString: Return longest String from an ArrayList
cw-getMaxLength: Return length of longest string from an ArrayList
cw-hasOdd: Check if an integer array has odd numbers
cw-implementAConstructor: Correctly implement a given constructor
cw-isUnique: Check if all elements of an array are unique
cw-isUniqueString: Check if all characters in a string are unique
cw-jerooCreateAndPlace: Create an object and call a function with it
cw-jerooInstantiation: Instantiate a new object
cw-jerooTurnAround: Perform task with a new object
cw-jerooTurnMethod: Create a function to perform a task with an object

- cw-lightBotMethodCalls1: Editing function calls for desired effects
- cw-lightBotMethodCalls10: Editing function calls for desired effects
- cw-lightBotMethodCalls9: Editing function calls for desired effects
- cw-mergingTwoIntegerArrays: Merge two given integer arrays
- cw-minDiff: Return minimum difference of adjacent elements in an array
- cw-parameterPractice1: Practicing passing parameters to functions
- cw-parameterPractice4: Practicing passing parameters to functions
- cw-recursionAddOddPosition: Recursive function to find sum of numbers
- cw-recursionCumulativeSum: Recursive function to find cumulative sum
- cw-recursionGCD: Finding GCD through recursion
- cw-recursionLargest: Finding larget number through recursion
- cw-recursionLog: Finding log through recursion
- cw-recursionMultiply: Finding product through recursion
- cw-recursionSumOfDigits: Adding digits through recursion
- cw-removeOdd: Removing odd numbers from an ArrayList
- cw-rotateRight: Move each element to the right in an array
- cw-simpleRemainderInCpp: c++ program to find remainder upon division
- cw-sortingCreateBubbleSort: Sorting an array with bubble sort
- cw-sortingFixSelectionSort: Sorting an array with selection sort
- cw-sortingInsertionSort: Sorting an array with insertion sort
- cw-sortingQuickSort: Sorting an array with quick sort
- cw-usingParameters1: Setting attributes for a custom object
- cw-whileLoopsWithRelationalOperators1: Changing while loop conditions
- cw-whileLoopsWithRelationalOperators2: Changing while loop conditions
- cw-whileLoopsWithRelationalOperators3: Changing while loop conditions

Bibliography

[1] A. Agrawal and B. Reed. 2022. A SURVEY ON GRADING FORMAT OF AUTOMATED GRADING TOOLS FOR PROGRAMMING ASSIGNMENTS. In *ICERI2022 Proceedings* (Seville, Spain) *(15th annual International Conference of Education, Research and Innovation)*. IATED, 7506–7514. https://doi.org/10.21125/iceri.2022.1912

[2] Kirsti Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15 (2005), 102 – 83.

[3] ArchieML. 2022. *Archie Markup Language*. Retrieved August 15, 2022 from http://archieml.org/

[4] Christopher Brown, Robert Pastel, Bill Siever, and John Earnest. 2012. JUG: A JUnit Generation, Time Complexity Analysis and Reporting Tool to Streamline Grading. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. Association for Computing Machinery, New York, NY, USA, 99–104. https://doi.org/10.1145/2325296.2325323

[5] Miquel Calvo, Artur Carnicer, Jordi Cuadros, Francesc Martori, Antonio Miñarro, and Vanessa Serrano. 2019. Computer-Assisted Assessment in Open-Ended Activities through the Analysis of Traces: A Proof of Concept in Statistics with R Commander. *EURASIA Journal of Mathematics, Science and Technology Education* (2019).

[6] Curtis Cohenour and Audra Lynn Hilterbran. 2016. Automated Grading of Excel Workbooks Using Matlab.

[7] Andrew J. Czaplewski. 2009. Computer-Assisted Grading Rubrics: Automating the Process of Providing Comments and Student Feedback. *Marketing Education Review* 19 (2009), 29 – 36.

[8] Stephen H. Edwards. 2021. Automated Feedback, the Next Generation: Designing Learning Experiences. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 610–611. https://doi.org/10.1145/3408877.3437225

[9] Stephen H. Edwards, Jürgen Börstler, Lillian N. Cassel, Mark S. Hall, and Joseph Hollingsworth. 2008. Developing a Common Format for Sharing Programming Assignments. *SIGCSE Bull.* 40, 4 (nov 2008), 167–182. https://doi.org/10.1145/1473195.1473240

[10] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. *SIGCSE Bull.* 40, 3 (jun 2008), 328. https://doi.org/10.1145/1597849.1384371

[11] Xiang Fu, Boris Peltsverger, Kai Qian, Lixin Tao, and Jigang Liu. 2008. APOGEE: Automated Project Grading and Instant Feedback System for Web Based Computing. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '08)*. Association for Computing Machinery, New York, NY, USA, 77–81. https://doi.org/10.1145/1352135.1352163

[12] Marcelo Guerra Hahn, Silvia Margarita Baldiris Navarro, Luis De La Fuente Valentín, and Daniel Burgos. 2021. A Systematic Review of the Effects of Automatic Scoring and Automatic Feedback in Educational Settings. *IEEE Access* 9 (2021), 108190–108198. https://doi.org/10.1109/ACCESS.2021.3100890

[13] Aliya Hameer and Brigitte Pientka. 2019. Teaching the Art of Functional Programming Using Automated Grading (Experience Report). *Proc. ACM Program. Lang.* 3, ICFP, Article 115 (jul 2019), 15 pages. https://doi.org/10.1145/3341719

[14] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '10)*. Association for Computing Machinery, New York, NY, USA, 86–93. https://doi.org/10.1145/1930464.1930480

[15] Westley Weimer James Perretta and Andrew Deorio. [n.d.].

[16] Amruth N. Kumar. 2005. Generation of Problems, Answers, Grade, and Feedback—Case Study of a Fully Automated Tutor. *J. Educ. Resour. Comput.* 5, 3 (sep 2005), 3–es. https://doi.org/10.1145/1163405.1163408

[17] Richard Lobb and Jenny Harlow. 2016. Coderunner: A Tool for Assessing Computer Programming Skills. *ACM Inroads* 7, 1 (feb 2016), 47–51. https://doi.org/10.1145/2810041

[18] Alan Marchiori. 2022. Labtool: A Command-Line Interface Lab Assistant and Assessment Tool. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3478431.3499285

[19] Divyansh S. Mishra and Stephen H. Edwards. 2022. *PEML Examples*. Retrieved December 14, 2022 from https://github.com/CSSPLICE/peml-feasibility-examples

[20] Divyansh S. Mishra and Stephen H. Edwards. 2022. *PEML REST API*. Retrieved December 14, 2022 from https://cssplice.github.io/peml/peml-api.html

[21] Divyansh S. Mishra and Stephen H. Edwards. 2022. *PEML: The Programming Exercise Markup Language*. Retrieved December 14, 2022 from https://github.com/CSSPLICE/peml

[22] Sidhidatri Nayak, Reshu Agarwal, and Sunil Kumar Khatri. 2022. Automated Assessment Tools for grading of programming Assignments: A review. In *2022 International Conference on Computer Communication and Informatics (ICCCI)*. 1–4. https://doi.org/10.1109/ICCCI54379.2022.9740769

[23] JavaScript Object Notation. 2022. *Introducing JSON*. Retrieved August 15, 2022 from https://www.json.org/

[24] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 22, 3, Article 34 (jun 2022), 40 pages. https://doi.org/10.1145/3513140

[25] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling How Students Learn to Program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 153–160. https://doi.org/10.1145/2157136.2157182

[26] Abdulaziz Shehab, Mahmoud Faroun, and Magdi Zakria Rashad. 2018. An Automatic Arabic Essay Grading System based on Text Similarity Algorithms. *International Journal of Advanced Computer Science and Applications* 9 (2018).

[27] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. 2017. Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale* (Cambridge,

Massachusetts, USA) *(L@S '17)*. Association for Computing Machinery, New York, NY, USA, 81–88. https://doi.org/10.1145/3051457.3051466

[28] Jaime Spacco, William Pugh, Nathaniel Ayewah, and David Hovemeyer. 2006. The Marmoset project: an automated snapshot, submission, and testing system. 669–670. https://doi.org/10.1145/1176617.1176665

[29] Shashank Srikant and Varun Aggarwal. 2014. A system to grade computer programming skills using machine learning. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014).

[30] Andreas Stahlbauer, Marvin Kreis, and Gordon Fraser. 2019. Testing Scratch Programs Automatically. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 165–175. https://doi.org/10.1145/3338906.3338910

[31] Chris Wilcox. 2016. Testing Strategies for the Automated Grading of Student Programs. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 437–442. https://doi.org/10.1145/2839509.2844616

[32] XML. 2022. *Extensible Markup Language*. Retrieved August 15, 2022 from https://www.w3.org/XML/

[33] YAML 1.2. 2022. *YAML Ain't Markup Language*. Retrieved August 15, 2022 from https://yaml.org/

[34] Jeremy K. Zhang, Chao Hsu Lin, Melissa Hovik, and Lauren J. Bricker. 2020. GitGrade: A Scalable Platform Improving Grading Experiences. In *Proceedings of the 51st ACM*

Technical Symposium on Computer Science Education (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1284. https://doi.org/10.1145/3328778.3372634