Comp 601

Final Exam

Jason Bell 3078931

April 9, 2017

Professor: Dr. Huntrods

Section A: Reusable Learning Object (RLO)

Introduction

For my undergraduate major project, I implemented a custom programming language and compiler, which I called Pie as both a nod to Python as inspiration, and in a fit of whimsy. This language was dynamic type, built on my preferred platform: .NET, and supported unit testing and test coverage analysis as language features. I included unit testing to learn about, and assess the efficacy of, test driven development. Because all components of a compiler have clearly defined and predictable inputs and outputs, this project was the perfect opportunity to do so.

While this goal was achieved, the project up to that point was largely just an intellectual exercise and proof of concept. The next step with the language is moving it past that. The programming language graveyards are full of dead languages, and unless future versions of Pie do something different, it will join them. I cite the D programming language as one example of a language that is extremely clean, beautiful, and polished, but did not take off because while it does what it does very well, it doesn't do anything new.

My RLO, a proposed language spec for the next version of Pie, was the culmination of the first two papers that I wrote for this course. The first paper surveyed the history of successful languages and investigated what made them popular. It concluded that:

- Popular languages tend to be experimental: for example, LISP's blending of data and code.
- Popular languages do something new: for example, being tailored for specific needs, such as business computing for COBOL and scientific computing for Fortran.
- Popular languages can be regressive: C was a backlash against the first generation of high-level languages, which tended to have English syntax and abstracted the hardware. Systems programmers desired a language that was more concise and allowed access to the hardware

My second paper looked at current popular languages, discussed future needs, and investigated why Fortran is still widely used despite being from the first generation of high level languages. It concluded that:

- Fortran maintained it's popularity by adapting to the times, by providing an easy to use means of implementing numerical problems in code, and by having very high quality support for distributing computing and data.
- Given that processing speeds of individual CPUs is likely to level out until quantum computing is mainstream, any future programming language MUST have excellent support for distributed computing and data.

Rationale

I implemented my RLO as a proposed language specification for the next version of Pie, as this was the logical culmination of the other preceding two papers. I would never endorse using a language specification, by itself, as teaching material: this was one of my criticisms of the Introduction to Java course when I took it. However, as supplementary material, just one RLO among many in a well-designed repository? Absolutely. The peer review process afterwards also provided an opportunity for my peers to provide feedback and suggestions for the next version of the language.

While the feedback has been quite positive, a common criticism is that it is too large to be an RLO. I think this is probably a valid criticism, but I make the counter point that what precisely constitutes an RLO is poorly defined. Definitions range from being so vague as to be meaningless [1], to being a piece of learning material that can be consumed in 5-15 minutes [2]. I think my submission is a RLO, though I concede that if it was put in a repository, it would make sense to break it up. Each section could be its own RLO and presented along with related materials.

Finally, I have the goal, once I have a more detailed specification, of publishing a white paper about the language. This paper would be presented as a request for comment, and readers would expect a well written formal specification to be available for assessment.

Target Audience

My target audience is, simply, programmers. I assume that the reader is familiar with programming fundamentals and do not try to teach them, otherwise the RLO would have been enormous. This is an RLO that would be suitable for an someone who has already learned to use another language, a student who has taken introductory programming courses, or as training material in a company. For example, I make mention of C++ pointers and smart pointers, but do not explain what they are: if the reader doesn't know, he or she can Google their way to victory; though I do provide a link to reference materials about smart pointers.

Design

As this RLO is a proposed language specification, it only made sense to look at existing language specifications for inspiration. This specification is loosely modeled after the C# 5.0 language specification [3]. I did not choose C# for any special reason other than it being my preferred language, so it was the first specification I looked at.

Language specifications tend to be no-frills. They're to the point: "the language does this, here's how, and here's why". My language specification is the same: I was more interested in implementing a concise and clear specification than entertaining the reader, particularly since I am neither an educator, nor do I wish to be. The goal was a piece of reference material that makes it convenient to quickly find what you're looking for, without sifting through extraneous text.

Beyond looking at existing language specifications for inspiration in presentation and layout, not a great amount of formal design process went into the RLO. As I described in my design document, my "writing process" generally consists of spending a long time researching and thinking about things, then regurgitating the contents of my brain onto a document and spending a few days making it look good. I wouldn't recommend teaching this as good practice, but it works well for me. The process is further facilitated by the fact that I am always thinking about Pie, so I never have a shortage of ideas and things to talk about. Finally, my feeling is that modern word processors largely make writing outlines ahead of time unnecessary and redundant, when it is so fast and simple to rearrange a document. Simply, I had an outline: just one in my head, rather than on paper.

Implementation

Beyond using existing language specifications as inspiration, the implementation was entirely organic. I had a very clear idea of what I wanted to talk about, and how I wanted to present it. The writing process just consisted of formalizing those ideas in a document: I spent more time playing around with layouts, colorizing the example code, and proofreading, than I spent writing the text.

The RLO is just a document, so requires no specific technology other than a PDF viewer.

The final product was exactly what I had planned. The final design matched what I had in my head, for the most part: as I said, there is an organic aspect to my writing process. For example, I was initially going to write about the object-oriented and functional programing paradigms in the introduction, but quickly realized that it would result in the "introduction" taking up half the document. Thus, they were moved to their own sections. This is something easy to do with word processors, so wasn't a big deal.

Testing

This is a difficult section for me, because the process of writing this document was organic, and straight from my brain. I hope I don't sound like I'm trying skirt the question, but simply: I knew exactly what I wanted to write about, so it was obvious when I was done. I did not deviate in any meaningful way from what I had in my mind, as I had put a lot of research and thought in ahead of time. If I were to describe a point where I knew I was "done", it was when I had written down everything I wanted to write down, and I exceeded 7000 words.

I could have talked about more. But the things I could have talked about were things I need to research more, and 7000 words was already long enough. Two examples of subjects I would like to talk about after more research are:

- Design patterns, and whether it's worthwhile to somehow support them as language features, or just through the standard library. I don't use design patterns as much as I should: my code tends to be loosely coupled modules. This is a subject that I need to study more.
- I would have liked to talk more about functional programming. Until recently, I almost exclusively used the object-oriented paradigm, and found the functional paradigm unintuitive. I have learned a great deal about functional programming in the past few months, and have developed an appreciation for some of its advantages, but there are still arcane aspects of it that I need to learn more about.

Regardless, I hit the limit that I wanted to reach, both in terms of what I was confident enough to talk about, and word count.

The question of a reader approaching the document in a manner I didn't intend is a difficult one. Part of the challenge for me is that I'm not an educator, and I'm also someone who's driven to independent learning. I suppose my simple answer would be "then they're doing it wrong". This isn't intended to be flippant: for example, if someone doesn't know how to program, I would never suggest that the first place they should go is Oracle's Java documentation. The target audience is people who already understand programming fundamentals, and someone who does not have those fundamentals will gain nothing from reading it. I can mitigate this by putting in disclaimers that inform the reader of this fact, but ultimately, it is up to the instructor to decide when and if it is appropriate learning material. For independent learners, it's up to them to decide whether they're ready for it: that's just part of independent learning.

Documentation Standards

The documentation standard I followed was, broadly speaking, the one found in specifications for existing languages. The appearance and layout of my RLO is modeled after that of the C# 5.0 language specification. This seemed perfectly logical: presenting a language specification in a format that differs drastically from existing ones will only lead to confusion by readers. This is even more the case considering that I plan to publish a journal paper on this subject as a request for comment. Readers will expect a high-quality language specification to be available to comment on.

The standard I emulated, found in the C# language specification, was a reference book with chapters and subsections. Chapter headings were in large bolded text and on the right, while subsection headings were in smaller bolder text on the left. Each chapter is a distinct topic from each other, and each subsection is made up of related topics.

Interspersed throughout the document are code samples that demonstrate the current topic. Like the C# language specification, this RLO uses the Consolas font for code, as it is monospaced and frequently used in for code. Unlike the C# language specification, this RLO adds colors to the code: blue for keywords, green for comments, and orange for literals.

I provided a table of contents that I was quite pleased with, and citations at the end.

Section B: Reflection

Introduction

I'd like to preface my discussion with a disclaimer: I am skeptical about RLOs. I spoke about this a bit in my personal reflections for the RLO. My opinion has not evolved: I feel that if RLOs and their repository are carefully crafted, they can be an excellent *supplementary* learning aid. If they are a disparate collection of materials from a wide range of sources, they are not. The example I would give would be the Introduction to Java course when I took it. Much of the material for that course was made up of links to Wikipedia articles and Oracle documentation. First, I didn't pay 900 dollars for the results of a Google search. But more importantly, Wikipedia articles and Oracle documentation are reference materials, not educational materials. This would be like presenting a reader with my RLO alone, and expecting them to learn how to program with it. That idea horrifies me: I would expect this RLO to be in a repository with other learning objects, specifically ones that cover programming fundamentals.

I firmly believe two things. First, RLOs must be designed to a common standard. It is genuinely jarring to jump between websites written by different authors, in different contexts, with different formatting, and often with conflicting information; speaking again from my own experience. I understand that one of the current definitions of an RLO is that it is a learning object that can be consumed in 5-15 minutes. Frankly, I strongly disapprove of this idea. I am skeptical about how much useful information can be presented in 15 minutes, what effect that level of fragmentation will have on standardization between them, and whether bite-sized 15 minute chunks will make sense when combined into a larger lesson.

Second, they must be supplementary. The document outlining what's expected for this final report states:

"Learning must change from being the movement of information from teacher to student or from manager to subordinate."

I feel like this is a sweeping generalization, and an inaccurate one at that. I have seen no convincing evidence that this is a thing that "must change". I can see only one context where this may be true: poorer countries with shortages of teachers. This is a case where practically speaking, anything that can help to overcome the teacher shortage is useful. Otherwise, traditional teaching methods have worked fine for thousands of years, and I don't expect that to change. I earnestly believe that there is no real replacement for teacher/student interaction. A teacher works directly with the student and can tailor to that students needs. An RLO repository can not. A teacher can answer unexpected questions asked by the student. An RLO repository can not. A teacher can lead and inspire, while an RLO can not. I don't think there's a real substitute for a teacher's ability to present the learning material in a natural and incremental way, and tailored to the audience.

So, as I said in my initial disclaimer: well designed RLOs should be supplemental. Education should stay traditional, but there's no harm for there to be a repository of useful learning modules for students to reference. Perhaps it could be a system where students are able to submit learning objects: interesting websites, blogs, YouTube videos, something they wrote themselves, and so on. Further enhance this with a voting system that allows students to vote based on how valuable they found the object. I realize that this contradicts my assertion that they should be consistent and well designed: but I think this is a rule that would have to hold true for the core learning objects. Put another way: when the student searches the repository for a topic, they are first presented with the well designed, consistently standardized, RLOs provided by the course. Following that are a set of search results for student submitted RLOs. In fact, now that I've written this, a part of me wishes I had implemented a system like that for my RLO submission: that would have been a fun project.

Community

The exam document continues with:

"In this context, multi-tasking and mutually concurrent conversations play an interesting part in allowing people to construct, deconstruct, and then reconstruct new meaning from activities, events, and reflection."

I must confess that I don't entirely understand what this is trying to say. The context in question is that "learning must change from being teacher to student", which of course is a premise that I disagree with. But how does that context lead to this statement? And how does it apply to RLOs? A repository of digital resources promotes concurrent conversation and multi-tasking? Multi-tasking of what?

It may seem like I'm being pedantic, but my question is this: how does traditional teaching methods not accomplish this? While interaction with peers is woefully inadequate in distance learning courses, it is not so for classroom universities. I attended the University of British Columbia, and some of my fondest memories are of group study with other students. Discussing ideas, teaching each other, helping each other out. As is commonly said: if you can teach an idea, you understand it. If you understand it well enough to present it in an alternative way that makes sense to someone else, you really understand it. I entirely fail to see how traditional classroom universities are failing to promote "concurrent conversations" or "allowing people to construct, deconstruct, and reconstruct new meaning". I probably learned more from those study sessions than I did in the classroom.

Part of the difficulty I have with this question is that I attended UBC in 2001. Laptops weren't ubiquitous yet, nor was internet access. But at the same time, I don't feel that my experience with learning is that different 16 years later. How has technology changed learning? Put simply: students are typing on laptops instead of writing in a binder, and are Googling rather than going to a library. I feel that looking for ways that technology is forcing teaching methods to change, is looking for a problem that doesn't exist.

My learning experience has differed from when I went to UBC in some ways, but that's because of the difference between distance learning and classroom learning. Student interaction is woefully minimal in distance learning. I'm sure that's partly practical considerations: people attend distance learning universities because they are too busy with a job or busy family life to attend scheduled classes. This was the case with me: I did my first two years of courses while working 60 hours a week as a restaurant manager. So, it should not be surprising that many students are more interested in getting things done, than spending a lot of time talking to other students. This is unlikely to be something that can be completely countered: distance learning students are always going to interact less.

Therefore, it still be useful to look for ways to encourage interaction. I believe that it would have to be something that students see as practically useful to their learning process. For example, when I took Linear Algebra, I was initially excited to see that there was a chat room. Alas, I checked it often, and never once did I see another student in there. Clearly, students did not see value in it. Perhaps RLO repositories that provide useful supplementary learning, and an avenue for students to share information would be such a system. As I'm not an educator, I'm not sure that I'm qualified to say if it would work or not, but I think it's an interesting idea. However, the presence of such a repository should not be used as justification to not provide an excellent text book and study guide. It should supplement them.

Learning

Personalized learning is adapting learning materials for individual needs, whether cultural, linguistic, special needs, prior experience, and so on. [4]

Personalized learning was something that I very much enjoyed about this course. While I would have preferred to skip the RLO-heavy focus of the course as I'm not an educator, I found immense enjoyment in being able to research and discuss the topics that interest me, and in my own way.

The concept of personalized or individual learning is precisely why I am an advocate for traditional classroom teaching: a repository of digital resources can not replace a human teacher's ability to adapt to the needs of individual students, never mind inspire them. Perhaps as artificial intelligence becomes more powerful, systems will be able to adapt to student needs. But they won't inspire them.

The idea that different people learn differently, and remember things differently, is common sense. I have experienced this in my own life. When I attended UBC in my early 20s, I could cram the night before an exam and get an excellent grade. Now that I'm 39, I find that I just can't do that anymore. I must spend a lot more time studying a subject to retain it. This has had a big advantage: whereas at 20, I could cram the night before, I'd promptly forget it. Because I now must put more time and effort in, my understanding of the subjects is much deeper and complete, and I remember it longer.

In addition to the effects of age, people have different life experiences and knowledge that determine how they learn new things. I'm very much a programmer, so that's how I approach things. For example, high-level theories often bore me to tears, and I am much more interested in building interesting things. Therefore, when learning about something new, I am most interested in practical applications. I enjoyed linear algebra immensely, because I saw the practical applications of matrices and linear problems. I did not enjoy discrete math, because it taught no useful applications other than using modulo math to create a hash function…and that's something I can Google if I must.

This is my experience alone. What about other people? They are, of course, going to differ from me. More interest in high-level theories, educational theories like RLOs, less interest in programming languages, younger, older, cultural differences, language differences, and so on. All these things are going to affect how they approach learning new things.

Inquiries

Experiential inquiries are conceptualized based on prior experience [5]. For example, I had a tough time with discrete math. It is safe to assume that I would be further challenged if I had to study a subject that built upon that knowledge, because that new knowledge would be built upon a wobbly foundation.

Subject matter inquiries are defined by how the subject matter is presented [6]. Can different subjects be presented in the same format? For example, can discrete math and philosophy be taught through the same course structure?

Self-inquiry is how well one learns based on one's knowledge of self and ego, though I did not find a good citation for this.

Group inquiries study a problem as a group, where each individual studies the problem, shares their thoughts with the group, which then forms a consensus [7].

I'm not sure that I have much to say here, in terms of reflection and discussion. I'm not an educator and this is out of my area of expertise. As I said, past experiences influence how well I learn new things (experiential), as can my ego (self). For example, Java is not my preferred platform. I have no real complaints about the language itself, but find the tools frustrating to use, relative to .NET or Python. This bias isn't entirely fair, but still influences how I learn new Java technologies, and not for the better. I find great value in group learning, as I discussed earlier in regards to group study at UBC.

Reflection

I won't spend a lot of time on this topic, as this paper is entirely reflective. Reflection is a good and positive thing. It's how you learn from past experience, both good and bad. If you don't look back, how do you learn from the things in your past?

Citations

[1] IEEE Learning Technology Standards Committee (LTSC) (2001) Draft Standard for Learning Object Metadata Version 6.1. http://ltsc.ieee.org/doc/

[2] Beck, Robert J., "What Are Learning Objects?", Learning Objects, Center for International Education, University of Wisconsin-Milwaukee

[3] Microsoft. (2017). C# Language Specification [Online] Available: https://msdn.microsoft.com/en-us/library/ms228593.aspx

[4] The Glossary of Educational Reform. (2017). Personalized Learning [Online] Available: http://edglossary.org/personalized-learning/

[5] Yeo, R. K., & Marquardt, M. J. (2010). Problems as building blocks for organizational learning: A roadmap for experiential inquiry. *Group & Organization Management*, *35*(3), 243-275.

[6] Arbaugh, J. B., Bangert, A., & Cleveland-Innes, M. (2010). Subject matter effects and the community of inquiry (CoI) framework: An exploratory study. *The Internet and Higher Education*, *13*(1), 37-44.

[7] Kraft, R. G. (1985). Group-inquiry turns passive students active. *College Teaching*, *33*(4), 149-154.