Final Report for Go Refactoring Internship

William Frazier

08-12-2014

Abstract

During this internship I helped with the development of Refactoring tools for the use with Google's Go programming language. Go is a new programming language being designed by Google, and refactoring tools are tools that allow software engineers to retroactively change the internal structure of programs to better accommodate new features and bug fixes over time. While working on the project, I was able to learn the basics of Go and put those to use in creating 3 tools for the refactoring project. I was also able to create a website for our Go version of the project (which I had to learn HTML and CSS for, and applied it to making my website for the internship as well). I was able to learn how to work with a group of programmers, and how to ask for help when I had a problem or how to help another group member when they had a problem. I was also able to gather valuable work experience from the group environment (working with a group, working in a group space) whereas in school I have only worked on projects alone at home or in a school lab by myself.

Introduction

The main thing I wanted to do with this internship was to gain work experience for my major. I didn't really have a particular research question, only that I had the desire to learn a new language and learn what refactoring tools are and how to make them using the new language. If I had to create questions for the project, they would be along the lines of what is Go and refactoring tools, and how do I implement the tools using Go. I chose this research project because I wanted to get more experience in software development, and didn't think creating a video game this summer would be as good, experience wise, as learning Go and how to create refactoring tools. I was also interested in learning what Google's coding standards are, since Google is one of the top software development companies, and most programmers set their goal to working at Google, because I would like to work there one day.

Project Description

This project was designed to create refactoring tools for Go. For my part, I was tasked with first learning what the similarities and differences Google's Go language has to other programming languages I already know (like C++ and Java). After learning about Go, I was next tasked with learning what an abstract syntax tree (ast) is and how Go uses it to create programs.

Once I learned ast's and made simple examples to examine them, I was tasked with my first refactoring, creating a refactoring that would search through a program file, and find all the functions, structures, and interfaces that are exportable (meaning other programs outside of the file can use the particular function, structure, or interface), and to check that each one has documentation for it. By documentation I mean a comment that is right above the function, structure, or interface and can be read on a website that gives info about each function, structure, or interface of a program. (For example, if you go to http://golang.org/pkg/go/ast/ you will see different functions, methods, and interfaces, along with a bunch of nodes, and info below each. The documentation that the program looks for is the sentences or paragraphs that describe what the particular function/interface/structure is, and how it works.) An example would be:

```
func Apple() {
}
```

In this case, there is no documentation above it, so the refactoring would create something to let the creator know it needs documentation:

```
// Apple TODO: NEEDS COMMENT INFO
func Apple() {
}
```

This will allow the program creator or anyone else who uses this program to know they need to add comments in detailing what the function does and what it's for.

The next task I was given was to create a refactoring that would extract local variables and would place a new variable in their place. The way this refactoring works can be better explained with a simple example: say you have a file for a program, and in this file you have the following.

```
a := 1
b := -2
if a + b < a {
        a = 5
        fmt.Println(a)
}
```

let's say you want to change the a + b to a new variable, then you would highlight a + b, then click on the extract local option, then enter a name for the new variable. Let's say the variable will be newVar, then extract local will extract the highlighted part, and place it in newVar, and replace a + b with newVar like this:

```
a := 1
b := -2
newVar := a + b
if newVar < a {
        a = 5
        fmt.Println(a)
}
```

This can be useful when you have a huge program and you want to input a new variable without figuring out where it goes, and there are error checks in the refactoring telling you if it's legal or not to replace a particular highlighted section. As of now only the single extraction option works, as the multi-extraction option requires a lot of work and also learning how to use something different from ast's (the professor explained that if I wanted to create the multi-extraction, I could use it as a master's thesis

based on how much more work and time I'd have to spend on it compared to the single extraction option).

After creating the basic algorithm, I got to the midpoint of my internship, and was told to create a website by dreu. Before I could create the website, I had to learn HTML and CSS since I don't have the money to pay for a website creation program, and the free ones would have taken as long to learn as it would take to learn HTML and CSS (since the timing would be the same, I figured learning how to make it programming wise would be more beneficial than just relying on a program to make it for me). I went to this website to learn HTML and CSS: http://www.codecademy.com/tracks/web. After learning how to create a web-page, and creating one, the professor asked me to create a website for our research project, so I created one based off of a parent site (mainly because we were making the refactoring tools for Google he suggested I design it similar to the site they had for their Go language: http://golang.org/).

After creating the template for the website, I was given my third and final refactoring to create. It was a refactoring that would find a particular interface based on the name the user inputs, and would create all the method functions for it in the program file, even if it imported the interface from another file. Again a simple example will help explain how it works:

Say we have a file with the following in it:

```
func main() {
	fmt.Println("works")
}
type Queue interface {
	Enqueue(x int)
	Dequeue() int
}
```

Now let's say we want to implement the queue interface. First we click on the interface refactoring,

then enter the name of the interface we want to refactor, and finally input the name we want the implemented structure for the methods of the interface to be. Afterward, the refactoring will make the file look like this:

```
func main() {
	fmt.Println("works")
}
type Queue interface {
	Enqueue(x int)
	Dequeue() int
}
type MyQueueStruct struct {
}
func (s *MyQueueStruct) Enqueue(x int) {
}
func (s *MyQueueStruct) Dequeue() int {
	return 0
}
```

This can be a useful tool if you want to create methods from the interface and then change what they have in them. This refactoring creates a bare-bones interface method implementation.

Results

The professor said I learned Go language faster than he had anticipated, which is one of the reasons why I was able to create 3 refactoring tools. Each refactoring algorithm only took a few days to a week and a half to create, but the rest of the programs took longer. (Making the inputs requires strings, and creating the strings in just the right way took time. Also, after creating the strings you have to figure out exactly where to input the strings into the file, which took some work to figure out each time. Also, the last month after finishing the internship I have been working with the professor using a grant Google had given him, and have been doing nothing but error checking, bug fixing, and also constantly updating our website since we are going to release it “into the wild”, as the professor states which means we are going to open source it to the public, this fall.)

Extract local has a bunch of times were errors or bugs occur, so the past 2 weeks alone have been spent on fixing it. So far about 100 lines of extra coding have gone into error checks and errors to print out for certain times. The error and bug checks for the documentation refactoring didn't take long, since it was a short refactoring. Error coding and bug checking haven't been started for the implement interface refactoring yet.

I believe I have gained a good deal of experience by working with this project, and am glad I chose it as my summer internship. Now that I've learned Go, I think Go is my new favorite language, with my previous one being Java. I will also be able to actually put job experience on my resume beside s my previous summer of creating a video game, which will hopefully help me get a job (which will be hard for me to get with me being older than most students who graduate with a bachelor's degree and also because I have learning disabilities, so the job experience and possible references will definitely help).

References

1. The Go Programming Language. Package ast page http://golang.org/pkg/go/ast/
2. Code cademy (2014). http://www.codecademy.com/tracks/web
3. The Go Programming Language. http://golang.org/