CSE 503 Homework 1

Beibin Li (ID: 1723402)

Jan/7/2018

I had visa issue and absent the first two classes; so, I spent some extra time on doing this homework (4 hours on reading lecture notes, 5 hours on doing the homework).

Reuse Code

Reusing code is beneficial: (1) when you make a change to a function, you just need to make the change in one place rather than many places. (2) Reusing code can also avoid writing bugs while reimplementing the same functionality. (3) Reusing code can improve the readability and reduce complexity of code.

However, reuse other's code is hard unless the code is well documented. Unfortunately, most code in CS research are not well-documented or well-commented because of time limitation: researchers have to spent time on writing grants, writing papers, presenting projects, and attending conferences. Even if a program might be well-described in published papers, its code can be hard to understand due to lack of comments and undocumented assumptions. It's also hard to locate a function/module in a large project (e.g. more than 100k lines of code) if you are not familiar enough to the project. Sometimes, you do not even know a functionality is already implemented by others in the past.

Sometimes programmers can hardly reuse their own code. Often, programmers have to try different function calls for a single line of code (particularly true for scripting languages), and they usually choose to just comment redundant code out (or even leave it as is) rather than deleting it for reproducibility (e.g. to reproduce testing accuracy, plots, etc). However, if programmer doesn't delete duplicate or "dirty" code, the script would become long, unreadable, and not efficient to run.

If code is well-documented, reusability would be largely increased. One solution to the problem is to create "markdown" or "html" version of the code (e.g. R Markdown, iPython Notebook) so that comments can be easily made and viewed. However, installing and using these tools might be time consuming for programmers (even if it just take few minutes), and those systems usually require programmers to run code in browser. Another possible solution is to create better "comment system" in IDEs (including Vim and Emacs), which can display comments nicely and show LaTeX formulas. So, programmers do not need to install new software nor use browser to view/edit codes.

From my personal experience, creating UML graph or logic graph would increase readability of code significantly. By reading a UML graph, new teammates (labmates, grad students) can understand large projects easily, and they can find bugs and reusable codes much quicker. However, creating and maintaining these graphs are not easy, and PIs usually do not create such documentation because it's not helpful to get funding. A better UML ploting software (as a plug-in in Comment System or Markdown language) can be useful.

Break large component of code to smaller pieces might be a solution to reuse code, because large component of code often have assumptions and hence hard to be reused. However, breaking large component is time consuming. A possible alternative solution is to create tools to break a large function into smaller ones. Sometimes, programmer can hardly break a large component because there are too many assumptions made in the code, and I often encountered this problem while adding features or performing maintenance.

Test Suites

I have created and used test cases while creating software and video games, but I rarely created test cases in research (for data analysis). However, creating test cases can ensure soundness and robustness for research.

I encountered the following problems while creating test cases for my data analysis codes: (1) Create test cases by hand is almost impossible in many domains. e.g. create brain fMRI signal data, which contains millions lines of data for one

Summary of Comments on CSE 503 Homework 1

Page: 1

Number: 1 Author: mernst Subject: Highlight Date: 1/9/2018 5:08:48 PM
There are many ways to reuse code. Please clarify. Here, you seem to talking about making multiple calls to a function.

Number: 2 Author: mernst Subject: Highlight Date: 1/9/2018 5:10:23 PM
Minor: I had to read this sentence three times to parse it correctly. You mean that reusing code prevents the need to reimplement functionality.

Number: 3 Author: mernst Subject: Highlight Date: 1/9/2018 5:11:34 PM
I'm not sure this is true. I see that it would reduce the size of the code, but adding abstractions generally increases complexity.

The same goes for readability. If the obstructions are well chosen, they should improve readability.

Number: 4 Author: mernst Subject: Highlight Date: 1/9/2018 5:12:36 PM
Typically you're looking for functionality. You know the function or module, it is easy to find.

Number: 5 Author: mernst Subject: Highlight Date: 1/9/2018 5:14:18 PM
I don't see what this adds. Also, the first sentence of the paragraph should be a topic sentence: the whole paragraph should be about that. Here, you've introduced a new paragraph with a throwaway sentence that is not related to the rest of the paragraph.

Number: 6 Author: mernst Subject: Highlight Date: 1/9/2018 5:15:14 PM
Why is commenting any safer than deleting? I don't see how this choice affects reproducibility.

Number: 7 Author: mernst Subject: Highlight Date: 1/9/2018 5:16:00 PM
If the code isn't being run, why does that affect efficiency?

Number: 8 Author: mernst Subject: Highlight Date: 1/9/2018 5:17:33 PM
This doesn't solve the problem. The problem is programmers not writing documentation. Converting the documentation from ASCII to HTML is unrelated to the root problem.

Number: 9 Author: mernst Subject: Highlight Date: 1/9/2018 5:16:33 PM
This doesn't seem like a serious problem, and it's not conceptual either.

Number: 10 Author: mernst Subject: Highlight Date: 1/9/2018 5:16:46 PM
I would think that learning the tool is the more serious issue.

Number: 11 Author: mernst Subject: Highlight Date: 1/9/2018 5:17:57 PM
Is this a significant part of the problem?

Number: 12 Author: mernst Subject: Highlight Date: 1/9/2018 5:18:35 PM
These graphs are separate from the code. I don't think it affects readability of the code. It might make the system as a whole easier to understand.

Number: 13 Author: mernst Subject: Highlight Date: 1/9/2018 5:20:10 PM
This might be helpful, but you've not described how it is helpful. Your document is touching on a number of possibly interesting issues, Billy better to narrow your focus to fewer of them and explain each one more thoroughly. High-level, vague descriptions do not inspire brainstorming to solve concrete problems.

Number: 14 Author: mernst Subject: Highlight Date: 1/9/2018 5:20:52 PM
Another disadvantage is that if there are very many components, it may be hard for people to know about all of them and reuse them.

Number: 15 Author: mernst Subject: Highlight Date: 1/9/2018 5:21:16 PM
Be specific about what the problem was and how you solved it. That can inspire solutions.

Number: 16 Author: mernst Subject: Highlight Date: 1/9/2018 5:24:15 PM
Why? Was this the right thing to do?

There multiple possible goals. One is to get a paper published and have impact that way. Another is to create a system that many other people use, and have impact that way. Neither one is correct or incorrect; each one can produce impact.

In other words, it is not obvious to me that this is a problem. Can you motivate that it is?

More generally, a software engineer should trade off the costs and benefits of the actions that he or she takes. The goal is to have impact. The goal is not to satisfy some arbitrary coding guidelines.

How would you decide whether it is a good idea to create test cases for particular research project?

experiment session. (2) There are many atypical cases (e.g. missing data, strange noise, etc) in the real world which you might not think about. Creating test cases to emulate these real world situations is hard and needs lots of thoughts. (3) Create test cases by generating random numbers (even with some smart restrictions) is not applicable because generated data might not follow real world logic. e.g. brain fMRi signals, heart rate, etc. (4) Even if we randomly generated test cases, we do not know what the correct output should be for that test case. (5) Actually, researchers are writing programs (and functionalities) that never existed before, and nobody knows what the correct output would be.

One possible solution is to learn patterns form existing data, and then synthesis fake data by infusing the learnt patterns. However, this solution doesn't solve the problem very well: how do you know your data generating program is bug-free? Which test cases do you use to test your test-case-generating program?

Another possible solution is to label your test cases (even if it needs lots of human labor), but this solution doesn't solve the whole problem either. Getting large amount of test cases becomes popular in computer vision and machine learning recently. Instead of lacking testing suites, lots of testing datasets (e.g. MNIST, ImageNet) are public available online. Different from traditional software development, machine learning use these testing suites not only as "testing for algorithm" but also as data for training. Nowadays, computer vision and machine learning researchers rely too much on these testing suites, and many research groups are just competing the "testing performance" on those data sets. However, these public testing suites are not sound. For instance, some papers claimed that they can detect vehicles from ImageNet with more than 90% accuracies (also sensitivity and specificity), but their models cannot achieve such performance in real world which contains lots of noises from weather, sunshine/light, shadow/dark, traffic, region/country differences, etc. Hence, even if the program can achieve good result from the testing suites, it does not necessary guarantee the soundness of the algorithm because it cannot be generalized to predict future data.
An obvious possible solution is to "get more data and more variable data", but this solution is not scalable because the real world is so complex. I cannot come up with better solutions for the software development and the machine learning testing problems at this point. However, solving the testing problems becomes crucial in CS research.

Maintenance

Making changes, adding features, resolving bugs for existing code are common tasks in maintenance. Maintenance is unavoidable for both researchers and programmers. At the same time, maintenance is time consuming, dirty, and not intellectually rewarding.

The maintenance problem is also related to "reuse code problem", because good documentation and enough comments can help both "reuse code" and "maintenance". I heard a joke about maintenance, "If you do not comment your code, then nobody can maintain it for you, and you are trapped to maintain your own code for your lifetime until the project is abandoned or comments are made". It's true for some software that the maintenance even takes more time than development.

Maintenance often break previously made assumptions. For instance, I wrote a "Fruit Ninja"-alike mobile game in the past, and now I want to reuse the class Target and class Background, etc in my "AngryBird"-alike video game. However, these class have assumptions that doesn't fit the new game: targets are not round shaped (physical engine change) fruits any more, and the background is not a static image (media codec change) now. If I choose to change existing classes to fit both games, I have to change the class (module) significantly: the logic will become more complex, the code will be longer, it sometimes will break my design pattern, and the UML graph should also be updated. During maintenance, I often found that I hadn't considered lots of things (e.g. multi-language support in game) in the original development. Discussing and thinking deeper about design pattern might help future maintenance, but it cannot avoid maintenance.

Moreover, debugging could cause (generate) more bugs, because debugging process often breaks local logics. Similarly, updating libraries and tools will enforce you to modify your code because of compatibility, which might cause new problems and bugs. Maintain code in local level will also destroy the style of code (e.g. I have to add embedded for loop inside other two for loops). The easiest way to avoid generating new bugs is to use test case, but this method caused testing problem as we discussed above. Read the version change notes before updating libraries is useful to handle library updates, but updating will still causes some unnoticed bugs. Rewrite the whole function might solve the problem of style, but it is time consuming and bug-prone.

In conclusion, in order to maintain code more efficiently and effectively, programmers should think deeper about design before writing, write well-commented code, and read library documentations carefully. New tools should also be developed to help programmers to complete those tasks.

Page: 2

Number: 1 Author: mernst Subject: Highlight Date: 1/9/2018 5:26:22 PM
I'm not sure what this means.

Number: 2 Author: mernst Subject: Highlight Date: 1/9/2018 5:27:14 PM
True, but the oracle does not have to be an exact answer. It is still possible to create a predicate that you know the program must satisfy. Same comment applies to point 4.

Number: 3 Author: mernst Subject: Highlight Date: 1/9/2018 5:27:35 PM
Label how?

Number: 4 Author: mernst Subject: Highlight Date: 1/9/2018 5:29:32 PM
This seems to be a problem with the data set. One obvious (and trivial) solution is to use better data sets.

But, is there any way to predict whether a given dataset or test suite is adequate?

(I now see that you come to a similar conclusion later in the section.)

Number: 5 Author: mernst Subject: Highlight Date: 1/9/2018 5:30:52 PM
I like this section, and your document as a whole, because it is raising a lot of interesting, juicy, important questions.
However, a weakness is that you mentioned these many questions but don't investigate any of them in detail. Therefore, you've deprive yourself of the opportunity to brainstorm deeply about them. You'll need to focus in on a smaller number in order to devise interesting questions and solutions.

Number: 6 Author: mernst Subject: Highlight Date: 1/9/2018 5:31:17 PM
Choose one of these and focus on it.

Number: 7 Author: mernst Subject: Highlight Date: 1/9/2018 5:34:51 PM
There is truth behind this joke: that is white is funny. So, can you figure out what the underlying problem is, and can you think about how to solve it?

Maybe the underlying problem is that programmers don't write comments. Then, you could think of ways to remind them or help them. Or, May the problem is that undocumented code is hard to maintain. You could think of ways to automatically create comments for undocumented code, or ways to perform common maintenance tasks even when documentation is lacking.

These are examples of ways to take an idea and make it more concrete. You would want to go into much more depth than a half here, but this is a way to start.

Currently, although there is a germ of an idea in this paragraph, the paragraph really doesn't add much to the document.

Number: 8 Author: mernst Subject: Highlight Date: 1/9/2018 5:32:54 PM
This is true for **all** successful software.

Number: 9 Author: mernst Subject: Highlight Date: 1/9/2018 5:35:38 PM
This note can example. But an even better example would be something that you really did in the past. You don't have to make up problems. The world in your life is already full enough of them!

Number: 10 Author: mernst Subject: Highlight Date: 1/9/2018 5:36:24 PM
Again, give a concrete example. If you have a concrete example, you can come up with a solution for that. It might not solve all problems, but at least would solve one, and it might lead you to useful generalizations in the future.

Number: 11 Author: mernst Subject: Highlight Date: 1/9/2018 5:36:53 PM
Programmers are not likely to change the behavior just because you urge them to. How can you provide tools to help them?