COMP 303 Study guide

Francis Piche

September 17, 2018

Contents

Part I
Introduction

1 Disclaimer

These notes are curated from Professor Joseph Vybihal's COMP303 lectures at McGill University. They are for study purposes only. They are not to be used for monetary gain.

2 About This Guide

I make my notes freely available for other students as a way to stay accountable for taking good notes. If you spot anything incorrect or unclear, don't hesitate to contact me via Facebook or e-mail at `http://francispiche.ca/contact/`

Part II
Writing Good Code

Good code is:

- Optimal (Time complexity and memory)
- Simple
- Correct
- Robust (Few crashes, good error handling etc.)
- Easy to read
- Is well documented
- Uses accepted engineering techniques.

3 Strategies for Writing Good Code

3.1 Optimality

To minimize memory usage we could:

- Encode long strings of data (compressing)
- Re-calulate instead of store values (trade-off with speed)
- Check library overhead!

- Data structure overhead (graph has a lot more pointers than a linked list!)

To minimize time complexity:

- Come up with a better algorithm (Improve big O)
- Store more things in RAM (rather than in a DB or over a network)
- Avoid nesting of pointers
- Avoid deep or unnecessary recursion

3.2 Simplicity

Strategies for simplicity:

- Good variable names (avoid single letters except for array index's)
- Don't use lots of variables when an array is more appropriate
- Use simpler data structures if possible
- Reduce the volume of code
- Limit line length
- Modularize the program
- Use algorithms that are well known if possible

3.3 Correctness

The only way to be somewhat sure that your program is correct is through rigorous testing. Some things are easier to test than others, and theres entire QA departments that specialize in testing. But with the rise of Agile, more and more standard Devs need to be good at testing.

Some general ideas on testing:

- 1 test per 1 function
- Test valid inputs
- Test invalid inputs
- Test edge cases
- Validate that the output is correct

There's a ton of material online about testing and I definitely recommend taking some time to learn about more in-depth testing methods than what is covered in this course. It's required in almost all dev jobs and it's a very valuable skill to have.

3.4 Readability

This comes from good indentation, spacing and general style of code. Most people know this implicitly so I won't go into detail. For examples see the lecture slides from Lecture 2.

3.5 Comments as documentation

The first place documentation happens is on the code level. This is where you can remove any ambiguity or sources of confusion from your code for other developers (or even future you). While real projects require real documentation, and excessive comments can be a detriment to readability, more comments are generally better than not enough. Break down your complex algorithms into comment separated steps, or add side-notes on anything that could be confusing to another developer.

4 Well Designed Objects

A well designed object should be one that does not cause any "wut?" moments amongst a team of developers. Specifically:

- Single class per file
- Single purpose
- Expose only essential information
- Support an API structure
- Follow appropriate inheritance methods

4.1 Single Purpose

A class should be used to represent an idea, concept or object (pun-intended), and nothing more. A student should only contain ideas directly relevant to a student. A `Car` class should not have information about busses or trucks.

This ensures that someone using your class can quickly find out everything there is to know about your object in an intuitive way. If your Student class has information about apples buried inside somewhere, someone on your team would have a hard time finding that out.

4.2 Restriction of Information

This is done to ensure that the class will be used correctly. There should be a decent amount of thought put into which parts of the code are "internal" and "external" to the class, ie: what parts of the class does the program need easy access to?

Any time you use the `public` keyword, you should be thinking twice about if it's truly needed.

4.3 Encapsulation

When you take the concepts of single purpose and restriction of information and put them together, you get *encapsulation*. Essentially what this means is that your class should be it's own complete bubble. No code outside the class should do the same things as inside your class, nor should things outside affect the class in an uncontrolled way.

4.4 Inheritance

To avoid duplication of code, we use inheritance to relate objects. This way classes can share common elements and simplify our lives. For details on inheritance see my COMP250 guide.

Note that private information from a parent is not visible t the child.

To design inheritance well:

- A parent is more general
- A child is more specific
-

4.5

Part III
Tools That Aid Good Design

Part IV
Design Patterns