Programming Best Practices

Karthik Dantu
Ethan Blanton
Computer Science and Engineering
University at Buffalo
kdantu@buffalo.edu

Administrivia

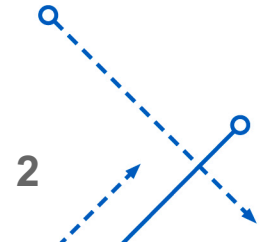
- Lab Exam 1 this week

  No books, no Internet, no cheat sheets

  Man pages, man pages, man pages

  Please stop by a restroom, eat and hydrate before !

- PA1 out – Conway's Game of Life



Effective Programming

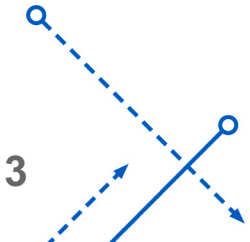
- Like every skill, the gap between a programmer and a good programmer is LARGE
- Good reasons for this difference

  Talent

  Knowledge

  Experience

  Practice
- You can get better on all these by adopting good programming practices
- Wonderful and lucrative skill to acquire!



PP 101 – Work Habits

- Best working habits, already stressed these before
- Start Early
- Work diligently – hard and smart
- Think before you code
- Comment and Document
- Write a second draft

Work Habits - Start Early

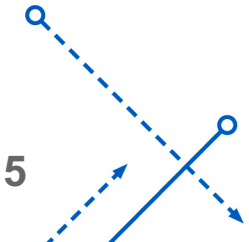
- Start programming early
- You'll have more time – but this is more than that
- Think about where and when you have programming insights

  Shower?

  Driving?

  Walking?

  In your sleep?
- Your sub-conscious works for you if you give it time

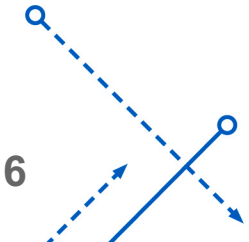


Work Habits – Comment and Document

- Comment your code judiciously
  - Include insightful comments
  - Avoid useless/redundant ones
  - Comment should tell you more than what code tells you at a glance
- Bad example

```
i++;    /* Increment i */
```

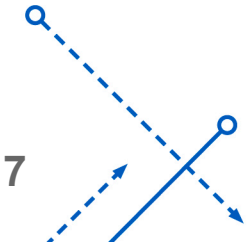
- Document when writing the code
- Helps crystallize your ideas and identify logical errors



Work Habits – Write a Second Draft

Plan to throw one away; you will anyhow.
- Fred Brooks, The Mythical Man Month.

- If you think your approach is getting unwieldy:
  - Stop and reconsider what you have learnt
  - Rewrite as necessary
  - Don't try to fix your solution based on the code



Work Habits – Getting Started

- Hardest thing sometimes is getting started
- Find something you know how to do and do it
- Do routine processing

  Process program arguments

  Perform simple calculations

  Define the data structures
- Once you start, it usually seems more tractable

Work Habits – Read and Write

- Read documentation
  - Man pages
  - API specifications
  - Standards
- Read programming texts
  - Several excellent texts (like K&R)
- Read code (future)
  - Learn from good programmers
  - Open-Source FTW!
- Write code
- Write documentation

Top Down and Bottom Up

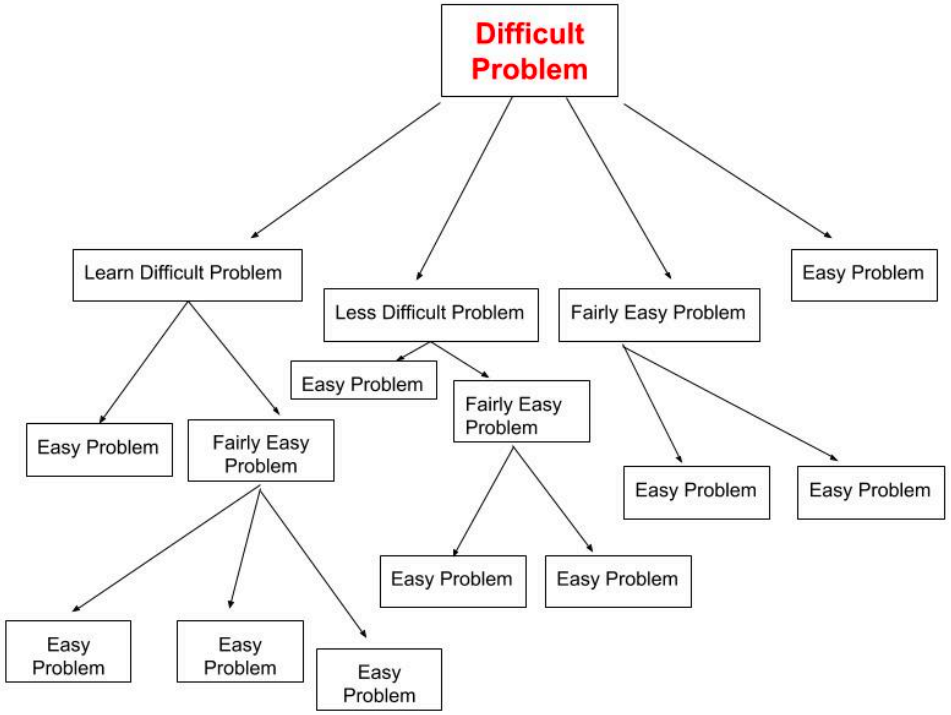
- For most big projects, we recommend a two-pass process:
- Divide the task Top-Down – recursively

  Identify the problem to be solved

  Determine what you need to solve it

  Define function/data structure to get what you need

  Identify common functionality when you do this



Top Down and Bottom Up

- For most big projects, we recommend a two-pass process:
- Implement Bottom-Up

  Identify sub-tasks you know how to solve

  Solve them

  Identify sub-tasks that can now be solved

Example of Top-Down: Stepwise Refinement

FIGURE 1.3
Structure Chart for Telephone Directory Problem

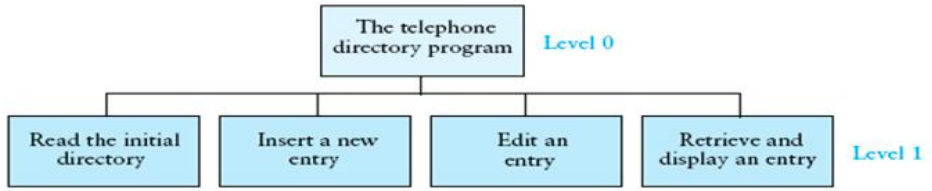


FIGURE 1.4
Refinement of "Read the initial directory"

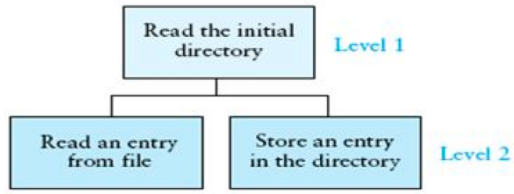
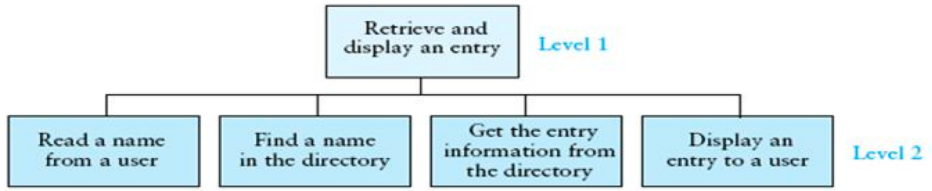


FIGURE 1.5
Refinement of "Retrieve and display an entry"



Chapter 1: Introduction to Software Design 16

Tackling Complexity

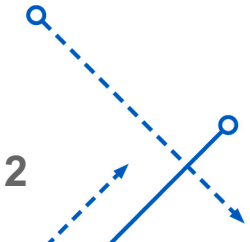
- Real-world projects are complex
- You manage them by

  Identifying modules and abstracting them into functions

  Defining and using constants

  Creating data structures to simplify computation

  Using standard library functions



Improving Programming Efficiency

- Using tools effectively is critical to efficient programming
- Tools include
  - Editor
  - Compiler
  - System tools such as `make`
  - A good debugger (`gdb`)
  - Version Control tools (`git`)
  - Text and data processing tools
  - Test suites you develop

In the long run, it is worth learning essential tools. They will pay back BIG TIME!

Useful tools - Editor

- Find a good editor, and **TRUST** it
- If you think something's weird, figure out why
- For example

  It is indenting funny → You misplaced braces

  Colored a variable named unexpectedly → You are shadowing a keyword/system variable

  It can't find a completion → You mistyped the command

  …

Compiler/Debugger

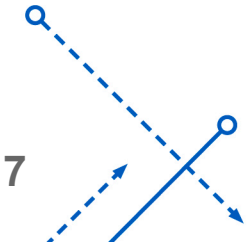
- Compiler
  - Can help in producing correct code
  - Can also help debug (`-Wall`, maybe `–Wextra`)
  - Silence warnings
  - Use the preprocessor for debugging
- Debugger
  - Can't afford NOT to learn `gdb`
  - Know when to `printf()` and when to `gdb`
  - Explore `gdb` features

Work Habits – Code Format

- Format your code precisely
- Just pick a style – and stick to it
- Badly indented code should bother you
- Code formatting helps spot logical errors

Work Habits – Program With a Purpose

- Don’t change code without forethought
- A change should address an issue – know what the issue is before changing
- It is always better to take longer and understand the problem
- Programming by Brownian motion – bad short term and long term
- Quick fixes cover up problems instead of fixing them



Summary

- Cultivate good work habits
- Design programs purposefully
- Learn/use your tools!
- Practice good style and form
- Debug/edit with a plan

Only way to become a better programmer is by programming !

Optional Reading

No required readings this class – all are optional

- https://www.topcoder.com/blog/coding-best-practices/
- https://www.doc.ic.ac.uk/lab/cplus/cstyle.html
- Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, 1999.
- Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering.* 20th Anniversary Edition. Addison-Wesley, 1995.
- Brian W. Kernighan and Rob Pike. *The Practice of Programming.* Addison-Wesley, 1999.