Stepwise Refinement

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.
THE COLLEGE OF HIGHER LEARNING.

Objectives

- Learn a way to solve problems

Problem

- How to solve a problem?
- Scenario: presented with a spec
- Where do you start?

Temptation

- Find a piece that you understand
  - Build it
  - Build up from there
- Problem
  - You build that piece
  - Then what?
  - Repeat!
- End up with:
  - A bunch of pieces that don't fit together
    - Lots of superficial work… then a crunch to put it together
  - Poor program flow
  - Spaghetti
- Because: weren't looking at the big picture

Why?

- The Magical Number Seven, Plus or Minus Two
  - Miller, G. A. (1956). "The magical number seven, plus or minus two: Some limits on our capacity for processing information". Psychological Review. 63 (2): 81–97. doi:10.1037/h0043158. PMID 13310704.
- An average person can hold 7 ± 2 objects in working memory.
- Frequently referred to as Miller's Law.

Top-Down Design

- **Stepwise Refinement**. A way of developing a computer program by first describing general functions, then breaking each function down into details which are refined in successive steps until the whole program is fully defined. Also called *top-down design*. [1]
  - A method of task division.
  - A series of layers of decreasing abstraction
  - A step-by-step Separations Of Concerns
  - Niklaus Wirth. 1971. Program development by stepwise refinement. *Commun. ACM* 14, 4 (April 1971), 221-227. DOI=http://dx.doi.org/10.1145/362575.362577
  - Dijkstra, E.W., 1969. Notes on structured programming. EWD 249, Technical U. Einhoven, The Netherlands.

Stepwise Refinement

- Start with the initial problem statement
- Break it into a few general steps
- Take each "step", and break it further into more detailed steps
- Keep repeating the process on each "step", until you get a breakdown that is pretty specific, ~~and can be written more or less in pseudocode~~
- ~~Translate the pseudocode into~~ real code
- [2]

- Brush Teeth
  - find toothbrush
  - find toothpaste tube
  - open toothpaste tube
    - put thumb and pointer finger on cap
    - turn fingers counter-clockwise
    - repeat prior step until cap falls off
  - squeeze tube onto toothbrush
    - (details omitted)
  - clean teeth
    - put brush on teeth
    - move back and fourth vigorously
    - repeat above step 100 times
  - clean up
    - rinse brush
      - turn on water
      - put head of brush under running water for 30 seconds
      - turn off water
    - put cap back on toothpaste
    - put all items back in cabinet

Advantages

- Encompasses YAGNI and KISS
  - You Arent Gonna Need It
    - Don’t make things until you need them
    - Not when you foresee needing them
  - Keep It Simple Stupid
- Incremental development (spiral model)
- Claims:
  - Easy to test
  - Can rapidly make working prototypes
  - Works for large systems

Problems

- Often leads to a “wide” hierarchy
- Often, “steps” are chronological
  - we end up with a data driven program
    - Not Object-Oriented
    - High coupling (modules know the details of the others)
      - In previous example: all parts know about
        - Toothbrush
        - Toothpaste
- May have issues with non-toy systems
  - For me, gets dicey @ ~ 2000-4000 lines
- Often ignores the creative steps
  - The idea that software engineering is like other engineering, where you can mechanically apply techniques to get a solution

BMI Example