Writing a Program

Software Engineering
CS 130

Donald J. Patterson

Content adapted from Essentials of Software Engineering 3rd edition by Tsui, Karam, Bernal
Jones and Bartlett Learning

Introduction

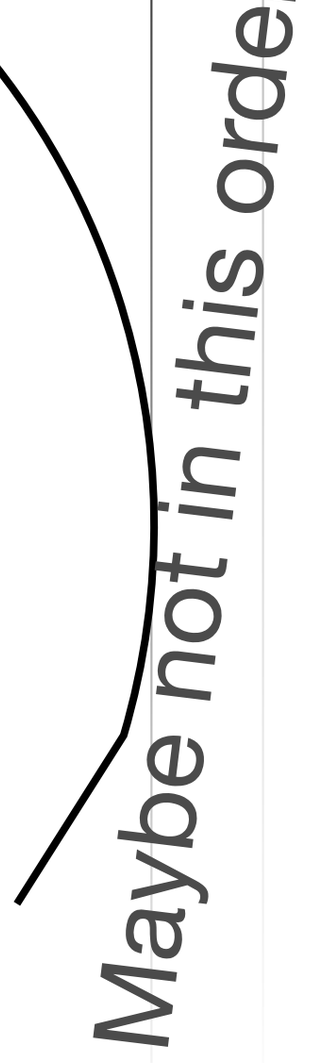
We all “start” by learning how to code in some programming language.

- With a small, hypothetical, and fairly well defined problem
- Usually the code is within one module

Introduction

We then learn that the program usually does not work on the first try, second try ------ may be even 5th or 6th try!

- We learn about “testing” the program
- We learn about re-reading and re-thinking the (problem) requirements more carefully --- then find that we may not have all the answers
- We learn about tracing and “debugging” the program
- Then ---- somehow magically ---- we decide that it’s “good enough !”



“Simple” Set of Steps

1. Understand the problem

- Contextual Inquiry
  - Study the environment in which the intervention will be deployed
- Requirements Engineering
  - Work with the client to figure out
    - Functionalities
    - Non-functionalities
      - performance, security, modifiability, marketability, etc

“Simple” Set of Steps

2. Perform Some Design

- Organizing the functionalities in some sequence;
  - possibly using some diagrams
- Focus on input/output (data, formats, organization)
- Think about some constraints (non-functionalities) such as speed, UI looks, programming language, dependencies, etc.
- any specific algorithm and improvements on sequence of functionalities.

“Simple” Set of Steps

3. Code/Implement

- Turning the design into actual code
  - Depending on how much design is completed, one may either directly engage in conversion to code (language dependent) or do some more designing.
    - Converting input/output to specific UI Interface or I/O format
    - Sequencing the processing in the desired order
    - Ensuring and converting the processing “algorithm” correctly to the target language construct.
    - figure out how to use language library (properly)

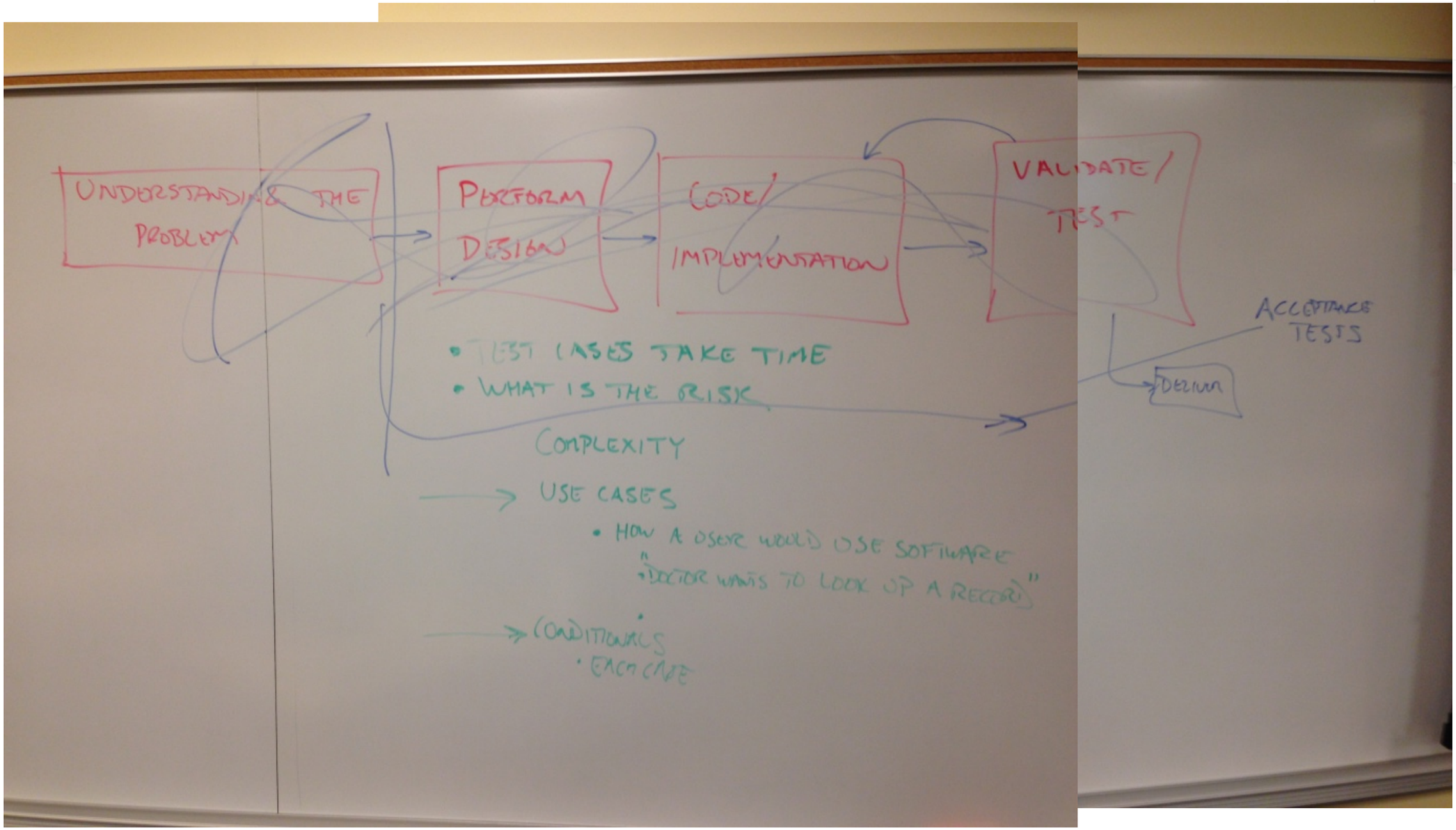
“Simple” Set of Steps

4. Validate/Test the program

- check the program results (via output) with some predetermined set of inputs.
  - The pre-determined inputs are “test cases” and requires some thinking.
- If the results do not match what is expected then:
  - “Debug”
  - Fix
  - Retest ---- revalidate
- Stop when all test cases produce the expected results.

How many test cases should we develop and run?

Narrative vs. Reality

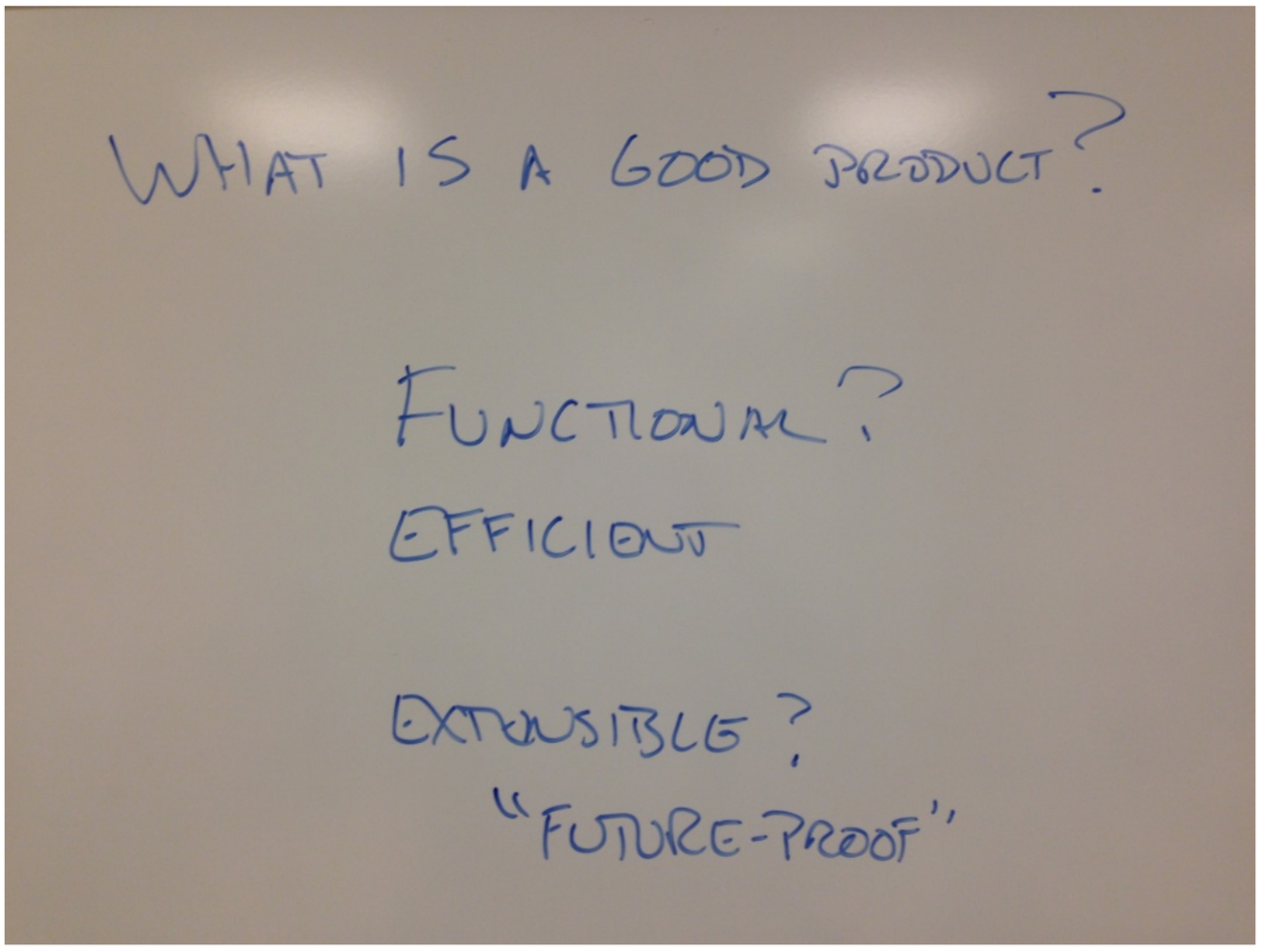


- The real process is a messy mix of the idealized process
- At the end, acceptance tests are contractual obligations

Code is “Done!” What Else Matters?

- How Long (elapsed time) did it take to complete the work?
- How much effort (total person hours) is expended to do the work?
- Does the solution solve the complete problem?
- How “good” is the work – (code, design, documentation, testing, etc.)?

How “good” is the work?



Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”

- Think about it for a second, then let’s discuss missing requirements

- 'THE FILE'
  - TYPE
  - LOCATION
- SORT BY ALPHA?
  - WORDS
  - LINES
  - NON-ALPHABETIC CHAR.
- WHAT IS A 'LINE'?

Decisions

Specifying the problem

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”

- It turns out that this task is under specified

- We need to clarify
  - the program requirements
  - the design constraints

Decisions

Complete specification

- The program requirements
  - State what the program does
  - Qualify what the program does
- Design constraints
  - Provide boundaries for ways in which the program can be created

Decisions

Requirements

- Not exactly the same as in common English
- Requirements cost money
  - Many are negotiable. Which ones?

Good

Fast

Cheap

Choose 2

Decisions

Requirements

- Functional Requirements
  - What the program does
- Non-functional Requirements
  - How the program behaves

Decisions

Requirements

- Functional Requirements
  - What the program does
    - “Sort a file”
- Non-functional Requirements
  - How the program behaves
    - Performance
    - Usability
    - Maintainability

Decisions

Design Constraints

- What languages can you use?
- What frameworks can you use?
- On what platforms must it run?

Decisions

Requirements vs Design Constraints

- The categories are not always clear cut
  - Functional Requirements
  - Non-functional Requirements
  - Design Constraints
- Some requirements can be de facto design constraints

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”

- What are the function requirements?
  - Brainstorm some

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Function requirements:
  - What is the format of the input data?
  - How is the data stored?
  - What is “a character”?
  - What is “a line”?

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Function requirements:
  - What order should the sort be?
  - How should sort react to non-alphabetic characters?
  - How should sort react to numbers?
  - Upper-case vs lower-case

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Function requirements:
  - Special cases:
    - Empty file?
    - Empty line?

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Non-Functional requirements:
  - Performance Requirements:
    - How long should it take?
  - Real-time Requirements
    - What about the variability in performance? Worst-case vs. Average-case
  - Modifiability in the future?

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Design Constraints
- What’s the user interface like? GUI vs CLI? web-based?
- Typical input size?
  - How much should we worry about algorithm?
  - Will it fit in memory? on disk? on one rack?

Simple Problem

Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Platforms
- What OS?
  - Often a business decision based on licenses, other systems
  - The computational world is fragmented, but each new platform incurs additional cost

Simple Problem

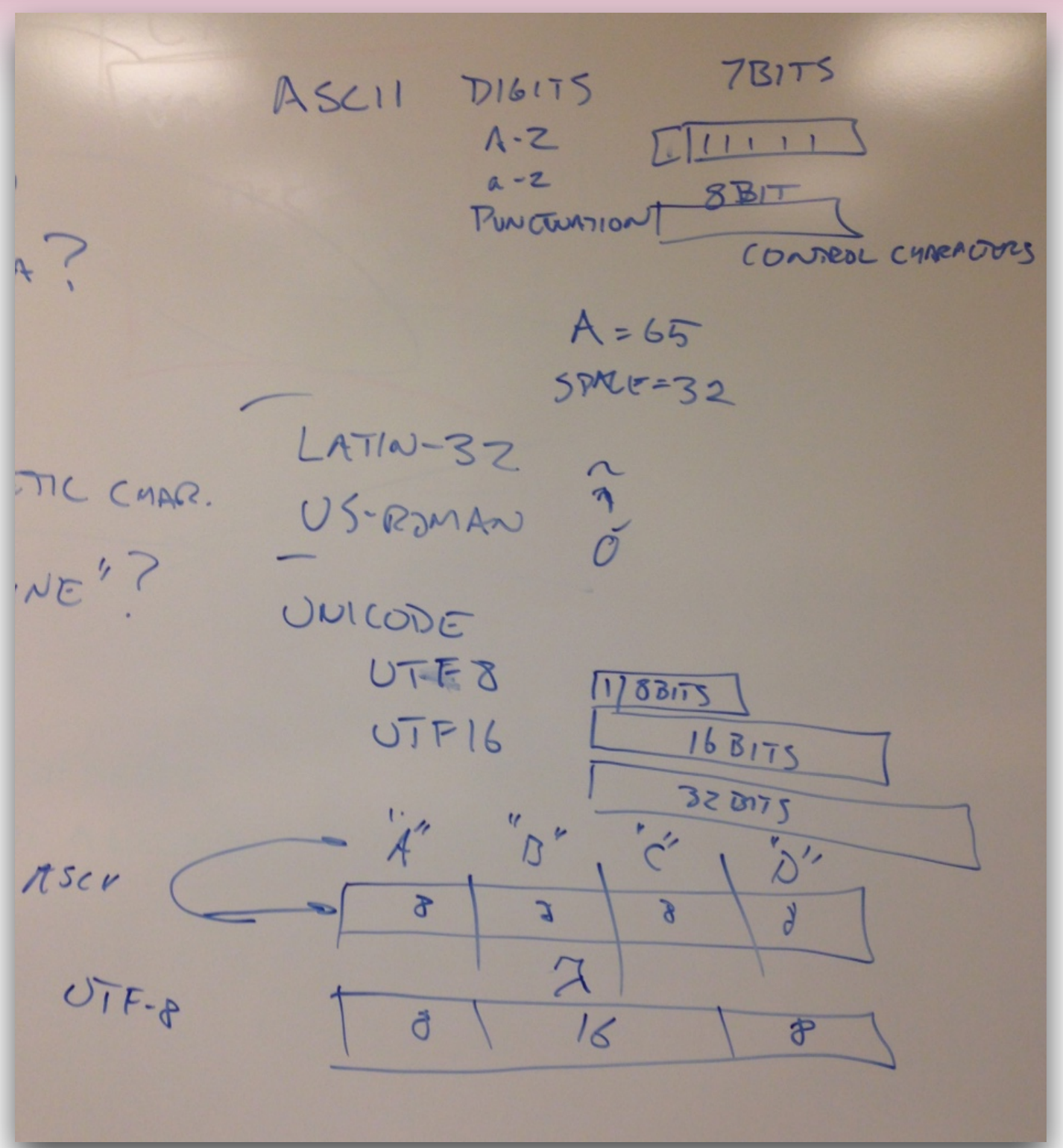
Thought Experiment

- “Given a collection of lines of text stored in a file, sort them in alphabetical order, and write them to another file”
- Schedule
- When does it have to be delivered?
  - Speed costs $$
  - $$ can’t make everything possible however

Thought Experiment

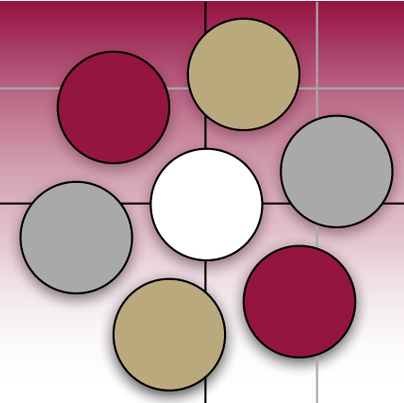
- 1 byte characters (UTF-8)
- Sort ascending, treating digits as characters, upper and lowercase differently in Unicode order
- Empty lines not special, empty files make empty files
- 1 minute to sort 100 lines of 100 characters
- no real-time requirements
- no modifiability requirements
- We should have a GUI, run on Mac, in Java
- typical input size will be 100 lines
- For Prof. Patterson: Due ASAP

Side-bar on character encoding



When in doubt use UTF-8

Prof. Patterson Experiment



I estimated 30 minutes to do the task

- 10:00 - started
- 10:15 - Eclipse crashed
- 10:30 - Decide initial design was bad
- 11:00 - laptop battery died - no charger - stop
- 13:00 - restart
- 13:21 - done debugging
- 13:36 - done writing tests
  - effort ~ 1.6 person hours
  - elapsed time ~ 3.5 hours

Decisions

Testing

- Acceptance Tests
  - High-level evaluations by a client, contractually bound
  - Do a sorting walk-through of examples
- UI Tests
  - Automated tests that simulate UI input
- Functional tests
  - Automated tests that simulate scenarios
- JUnit tests
  - Automated tests that test method and class specs

Decisions

Testing

- “Extreme Programming” methodology
  - Write the tests before you write the code

Decisions

Estimating Effort (aka Project Management)

- Breaking down a problem into sub-tasks
- Estimating the time for each
- Assigning a cost to the project

Decisions

Exercise

Write a “program” in your favorite language that will accept numerical numbers as inputs, compute the average, and output the answer.

Provide an estimate within one minute:

How long (in elapsed-time) would it take you to implement this solution?

How much overall effort (in person-hours) will this take?

How well will your solution match the problem?

How good is your code/design/documentation/testing?

Decisions

Estimating Effort

- Did you include meal breaks, rest breaks, bathroom breaks?
- Did you break down the task?
- Did you include GUI? Testing? Bug fixing?

Decisions

Estimating Effort

- This is one of the toughest problems in software project management.
- Accurate estimates are very hard to make
- Estimates should be made by the person assigned to the task
  - Hopefully after some reflection and data on performance.

- Pivotal Tracker

Previous Class Answers

- How long (in elapsed-time) would it take you to implement this solution?
  - Class Answer: 10 min. (A); 15 min (B); 1 hr. (C);
- How much overall effort (in person-hours) will this take?
  - Class Answer: 10 person min. ; 15 person min. ; 1 person hour; 3 person-hrs
- Will your solution match the problem?
  - Class Answer: YES!
- How “good” will your solution be?
  - Class Answer: Awesome!

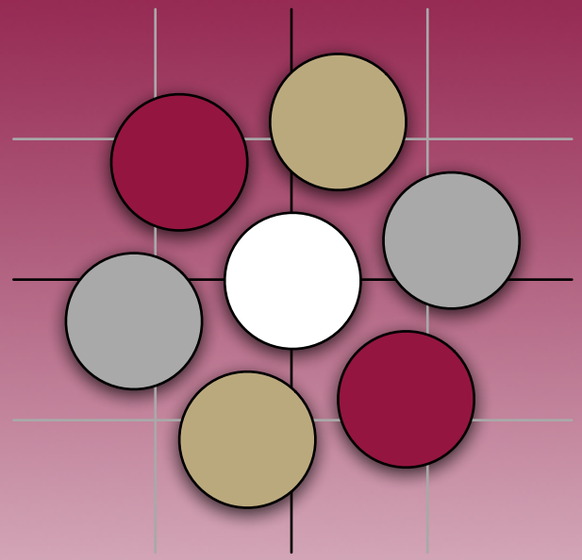
Some “Previous Class” Inputs

- How long do you think assignment #1 would take?
  - 1 hr --- 7 people
  - 2 hrs --- 6 people
  - 3 hrs --- 2 people
  - 10 hrs --- 3 people
- Real data from class:
  - Elapsed time: range was 5 days to 46 minutes – mostly between 1 to 3 hours
  - Effort: range was 8 person-hours to 40 person-minutes – mostly between 1 person-hour to 3 person-hours

Implementation

Conventions help teams move faster

- Teams need to agree on syntax conventions and stick to them
- When do you use capital letters?
- How are things named?
- How are comment utilized?
- Testing before giving code to others
- Having someone else review your code before sharing



WESTMONT COMPUTER SCIENCE