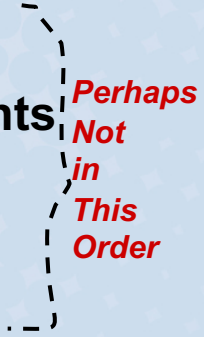
Introduction to Creating a Program

- **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.**
- **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 !"**



This is a common (popular) developer's "simple-reflex" approach.

Considerations and Decisions

Problem Statement

> “Given a collection of lines of text (strings) stored in a file, sort them in alphabetical order, and write them to another file.”

Do we have all the information we need?

Considerations and Decisions

Problem Statement

"Given a collection of lines of text (strings) stored in a file, sort them in alphabetical order, and write them to another file."

What Are the Program Requirements?

Input formats? Sorting?

Special cases, boundaries, and error conditions?

Performance? Real time? Security?

What Are the Design Constraints?

User interface? Typical and maximum input sizes?

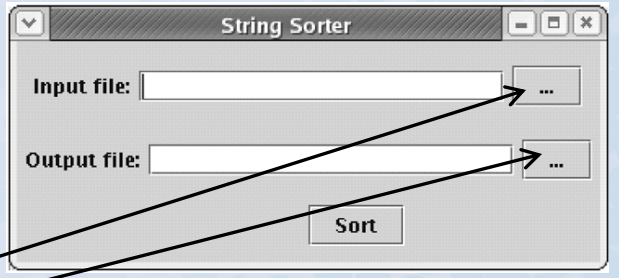
Platforms? Schedule?

More to Consider and Decide on

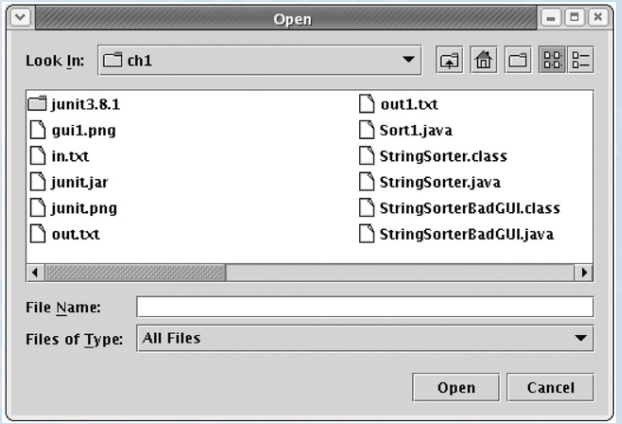
- Testing Time
  - While program is defined
  - While program is developed
  - After program is completed
- Kinds of Tests
  - Acceptance (validation)
  - Verification
    - Unit testing
    - Black box
    - White box

Version 1. Estimate Total Minutes

Write a program that reads lines from one file and writes the sorted lines to another file. Assume that you will be writing the sort routine yourself and will implement a simple GUI (first image).



Pressing one of the two buttons displays a File Open dialog (second image), where the user can navigate the computer's file system and choose a file.



Assume that you can work only on this one task, with no interruptions. Provide an estimate within 1 minute.

Step 1. Estimated ideal total time: _____

Version 2. Estimated Calendar Time

Is the assumption that you will be able to work straight through on this task with no interruptions realistic?

Won't you need to go to the restroom or drink some water?

Can you spend the time needed on this task?

If you were asked to do this task as soon as reasonably possible, starting right now, can you estimate when you would be finished?

Given you start now, estimate when you think you will have this program done to hand over to the client. Also give an estimation of the time you will not be on task (for example: eating, sleeping, other courses, etc.).

Step 2. Estimated calendar time started: _____ ended:_____ breaks:_____

Version 3. Estimation of Subtasks

Divide the entire program into separate developmental tasks; these tasks might be divided into several subtasks.

Your current task is a planning task, which has estimation as a subtask.

When thinking of the requirements for the project, assume you will create a class, called StringSorter, with three public methods: Read, Write, and Sort.

- For the sorting routine, assume that your algorithm involves finding the largest element, putting it at the end of the array, and then sorting the rest of the array using the same mechanism. Assume you will create a method called IndexOfBiggest that returns the index of the biggest element on the array.

Ideal Total Time	Calendar Time
Planning	
IndexOfBiggest	
Sort	
Read	
Write	
GUI	
Testing	
Total	

Actual Time Creating the Program

Now design and implement your solution while keeping track of the time.

	Started:	Ended:	Breaks:	Time
Planning (Estimation)				
IndexOfBiggest				
Sort				
Read				
Write				
GUI				
Testing				
TOTAL:				

A "Simple" Set of Steps

1) Understand the problem – requirements
   - Functionalities
   - Non-functionalities: performance, security, modifiability, marketability, etc.
2) Perform some design – based on requirements
   - Organize 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.

A "Simple" Set of Steps (cont.)

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.

     A) Convert input/output to specific UI interface or I/O format.
     B) Sequence the processing in the desired order.
     C) Ensure and convert the processing "algorithm" correctly to the target language construct.
     D) Figure out how to use language library (properly).

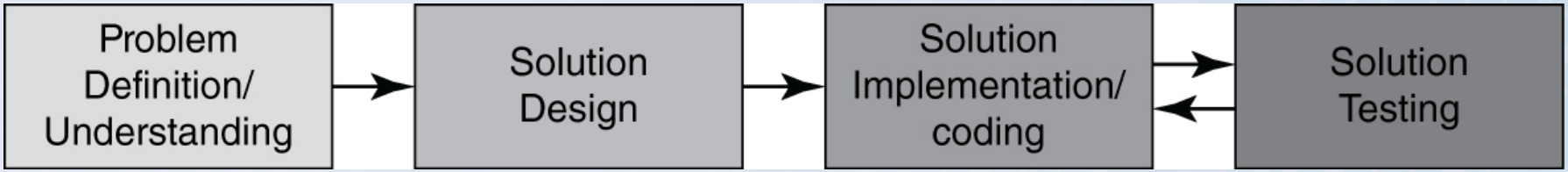
A "Simple" Set of Steps (cont.)

4) Verify/Test the program – check the program results (*via output*) with some predetermined expected set of inputs.
   - The pre-determined inputs are "test cases" and require some thinking.
   - If the results do not match what is expected then:
     - "Debug"
     - Fix
     - Retest — reverify
   - Stop when all test cases produce the expected results.

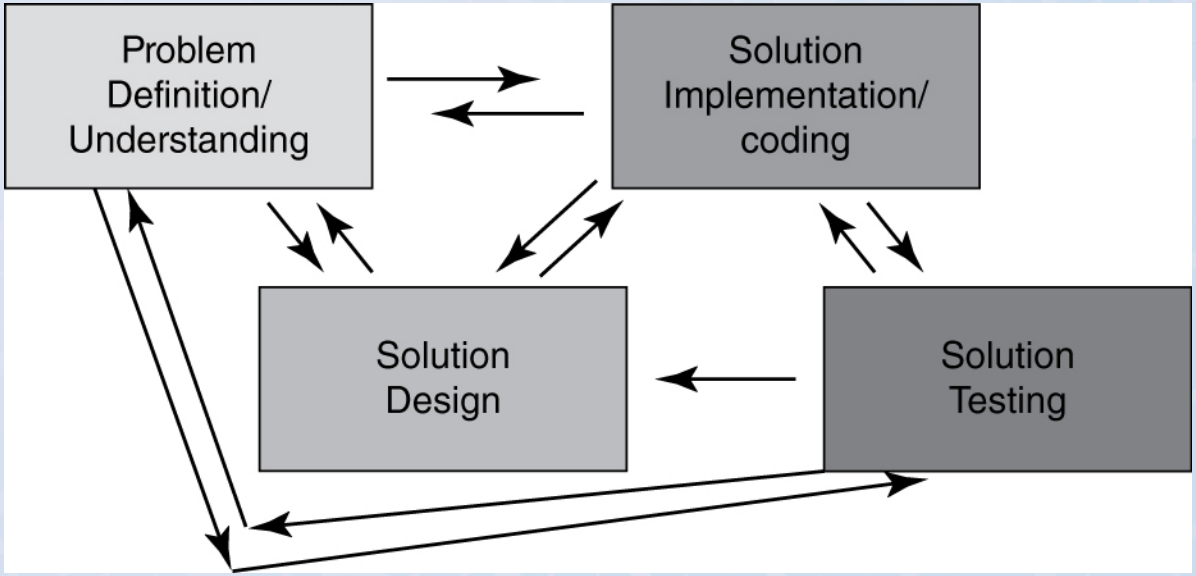
Question: How many test cases should we develop and run?

What Really Happens?

"Imagined" – Ideal Situation



"Actual" – Happening



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.)?***

based on?