

Lecture #7

To Ponder

- Write down either α or β on a slip of paper and turn it in
- Final grades are determined by random pairing of students (you won't know your partner)
- Your grade will be:
 - A, if you chose α , your partner chose β
 - B-, if you chose α , your partner chose α
 - B+, if you chose β , your partner chose β
 - C, if you chose β , your partner chose α

Game Matrix: Grades

	Alpha	Beta
Alpha	B- / B-	A / C
Beta	C / A	B+ / B+

Payoff to: Row / Col

Game Matrix: Taxes

	Evade	Pay
Evade	-1000 / -1000	1000 / 990
Pay	990 / 1000	990 / 990

Payoff to: Row / Col

Game Matrix: Group Work

	Shirk	Work
Shirk	/	/
Work	/	/

Payoff to: Row / Col

A Little About Documentation

- See “common errors” link on web page
- Resource: OSU Technical Writing Center
 - see link on web page
- Clarification of some elements of the Programmer's Guide:
 - Data Structures / Types
 - Specifications
- Especially important: *shared elements*

Testing

1. Philosophy
2. Example
3. "How to"s (including code) and caveats
4. Levels of Testing

Testing: Philosophy

Definition of Testing

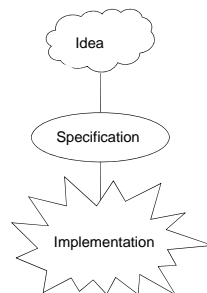
- What is "testing"?
 - A process whereby we *increase our confidence* in an implementation by *observing its behavior*
- Fundamental point:
 - testing can detect the *presence* of mistakes, never their *absence!*
- Fail a test case ==>
- Pass *all* test cases ==>

Importance of Testing

- Despite limitations, testing is the most practical approach for large systems
- Knuth quotation:
"Warning: I've only *proven* this algorithm is correct... I haven't *tested* it!"

Theory

- 3 levels of abstraction in functionality
- Want: the idea
- Have: implementation
- "Testing" requires comparing it against something, but what?



Theory (II)

- Ideal: test against our "idea"
 - but the idea is usually too fuzzy
- So make it concrete by writing specification
 - defines desired mapping from input to output



Testing: An Example

Example: Sorting a List

- Idea: function sorts a list in _____ order
- Spec: `void sort (List x)`
requires: $|x| \leq 100$
modifies: `x`
ensures:
- Q: do we really need the “expected output”?
 - i.e. why not just look at actual output and see if it is sorted?

Expected Output

- A:
- Specifications often relate *final* states to *initial* ones
 - but not necessarily true
 - e.g. `void f(int x)`

Testing: “How To”s and Caveats

The Right Frame of Mind

- Tests should be written to *break* a program
 - not to show it works!
- When a test reveals an error, that's success!
- Good approach: have someone else test your code

Importance of Independent Testing

- See IEEE Computer, Oct '99
 - study at NASA Langley
 - had two groups working in parallel
- The group with independent testers found:
 - *more* faults overall (critical and non-critical)
 - found these faults *earlier* in the process
 - fixed these faults with *less effort*

Figure 1 from Arthur article

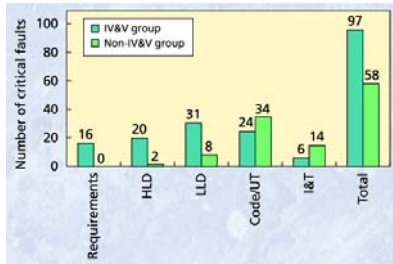
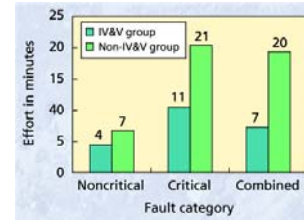


Figure 2 from Arthur article



How To Choose Test Input

- Too many possible inputs to test them all
 - space of possible inputs defined by *requires*
- Important kinds of test input:
 - extremes (e.g. empty list, $|x| = 100$)
 - trivial / degenerate ($|x| = 1$, x is already sorted)
 - error-generating
 - different categories (+ve number, -ve numbers)
 - typical input (random list)

Lecture #8

How To Generate Expected Output

1. By hand
 - error-prone and tedious
2. With another program
 1. also error-prone
 2. often just redoing the implementation, and making the same mistakes!
3. Work backwards
 1. an inverse may be easier to calculate
 2. e.g. start with a sorted list, and permute it

Alternate: Validating Output

- Steps:
 1. Keep a copy of the input
 2. Run the program
 3. Validate the actual output against input
- Example: sorting
 - write two functions:
 - copy the input
 - run program and check:
- Checking functions may be simpler than the full implementation

Dangers with Testing

- “Expected output” is wrong
- Testing program is wrong
 - extra code means more chances to mess up
 - e.g. `permute(A,B)` always returns true
- With these errors, there are 2 dangers:
 1. spurious test failures
 2. false positives
- Which is worse?

Dangers with Testing (II)

- A third, more subtle, potential error: The specification is “wrong”
 - how can this be?
 - may not be exposed during testing
 - to increase the chances of finding these problems, have *someone else* test your code!

Testing: Levels

Levels of Testing

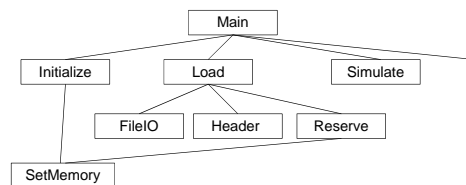
- Typical testing path:
 1. Unit tests
 2. Integration tests
 3. System tests

Unit Tests

- Individual modules tested in isolation
- Two flavors:
 1. Black box: testing based *only* on specification (tester doesn't even look at code)
 2. White box: testing based on code structure (e.g. tester makes sure every branch of a switch statement is followed)

Integration Tests

- Modules tested in combination in order to check the *interfaces*
- Best done incrementally



Bottom-up vs Top-down Testing

Bottom-up

- start with most basic modules
- easy to exercise all the features
- write a “driver” for higher-level modules

Top-down

- start at top (main)
- test interfaces early
- write “stubs” for lower level modules

Often these two occur simultaneously, in tandem

System Tests

- Verify that system as a whole meets the requirements and specifications
- Three flavors:
 1. alpha: by developers, before release
 2. beta: by “friendly customers”, before general release
 3. acceptance: by end customer, to decide whether or not to hire you next time!

Technical Writing

What Is “Technical Writing”?

- Writing we do as part of our jobs
- Possible purposes:
 - Inform
 - Instruct
 - Persuade
 - Call to action
- Missing from this list:
 - Entertain

Four Characteristics of Effective Technical Writing

1. Engages a specific audience
2. Uses plain and objective language
3. Stresses presentation (obvious structure, understandable at a glance)
4. Employs visual aids

Why Bother?

- Communication is fundamental in society
 - Politics, law, science, personal lives, health,...
- Fundamental in personal success:
 - Good idea
 - **Ability to communicate that idea**
- Highly valued by employers

Writing in Computer Science

- Taking exams
- Documentation (for users and developers)
- Reports and memos
- Papers (journals, conferences, magazines...)
- Proposals
- Reviews of others' work
- Books

Good and Bad News

- The bad news
 - Most of us are not very good at it
 - We enjoy “technical” challenges much more
- The good news
 - Writing is actually not too different from computer science!

Lecture #9

Parallels Between Writing and Computer Science

Programming	Writing
Must identify/determine:	Must identify/determine:
<ul style="list-style-type: none">• User• Purpose of program• Program features• User interface	<ul style="list-style-type: none">• Audience• Purpose of document• Depth of document• Style and tone
<i>“Preprogramming”</i>	<i>“Prewriting”</i>

Parallels Between Writing and Computer Science II

Software Development	Technical Writing
<ul style="list-style-type: none">• Requirements and design• Implementation• Testing• Debugging	<ul style="list-style-type: none">• Prewriting• Composition• Reviewing• Revising

Analyzing the Audience

- Few people read this stuff for fun
- Must correctly identify a “customer” and what that customer’s needs are
- Consider writing an audience profile
 - Novice, technician, expert, manager, VC, ...
 - Reading level
 - Motivations, biases, expectations, ...

Analyzing the Audience II

- Make this analysis concrete by stating assumptions made on background, motivation, needs, etc.
 - “The reader is expected to be familiar with the predicate calculus.”
 - “This manual is designed for application programmers who write S47G applications for the insurance industry.”

Technical Audience

- Function-oriented organization
 - E.g., alphabetical listing of all functions
- Want a complete (exhaustive) resource
 - *All* information they might want is there somewhere
- Willing to spend a great deal of time
 - Will read the document *carefully*

Customer Audience

- Prefer task-oriented organization
 - E.g., enumerated steps for each possible task
- Want just the necessary information
 - Only the information critical to their jobs is there
- Will spend as little time as possible
 - Must be concise and easy to read

Identify the Purpose

- Rule of business letters and memos:
 - Begin with clear statement of what you want!
- Larger documents are no different
 - What information are you trying to impart?
 - What are you trying to teach?
 - What view do you want the reader to adopt?
 - What action do you want done?

The Depth of Writing

- Bloom's taxonomy of cognition:
 1. Knowledge
 2. Comprehension
 3. Application
 4. Analysis
 5. Synthesis
 6. Evaluation

Verbs Used in Statement of Purpose

- Knowledge
 - count, define, draw, identify, indicate, list, name, quote, recall, recite, recognize, record, state, tabulate, trace, write.
- Comprehension
 - associate, compare, compute, contrast, describe, differentiate, discuss, distinguish, estimate, extrapolate, interpolate, predict, translate.

Verbs Used in Statement of Purpose II

- Application
 - apply, calculate, classify, complete, construct, demonstrate, employ, examine, illustrate, practice, relate, solve, use.
- Analysis
 - analyze, detect, explain, group, infer, order, relate, separate, summarize, transform.

Verbs Used in Statement of Purpose III

- Synthesis
 - arrange, combine, construct, create, design, develop, formulate, generalize, integrate, organize, plan, prepare, prescribe, produce, specify, research.
- Evaluation
 - appraise, assess, critique, determine, evaluate, grade, judge, measure, rank, select, test.

Prewriting: Getting Started

- Read the question / problem statement *carefully*.
- Make a list of the required cognitive tasks.
- Assess what you know.
- Compare this knowledge with the level of the required cognitive task

Example

“Compare the performance of two cache replacement algorithms”

- Cognitive tasks
 - Compare
 - Contrast
 - Maybe analyze and recommend?
- Recall various issues in cache algorithms

Lecture #10

Prewriting Tasks

- *Quick* list
 - Specific points for each cognitive task
- Brainstorm
 - List everything you know about the topic
 - Do not judge or weed anything out
 - Objective: quantity
- Review list
 - Assess where research is needed

Prewriting Tasks II

- Choose a single point that will be in the final product and outline a section to develop that point.
- Involve others as appropriate.
- Do the research.
- Plan the format.

This is not a linear process!

Prewriting Tasks III

- Outlining
 - Get the planned structure down
 - Avoid forgetting a key point
 - Check for the logical flow of arguments and information
- This is an easy step to skip, but good work here will pay dividends in the future!*

Writing the Document

- The better the preparation in the prewriting phase, the smoother this goes.
- Regardless, it's still work!
- Requires tools, skills, practice, experience, and motivation.

Technical Writing: "Document Component Engineering"

- Component = section of the document.
- Often has its own heading.
- Large components consist of smaller ones.
- Each (large enough) element from the outline becomes a component

Advantages of Components

- The whole document is too intimidating
- Obvious milestones
 - Reduces panic (you know where you stand)
 - Permits time budgeting
- Reduces writing to a step-by-step process
- Instant gratification
- Easy cure for writer's block: work on a different section

Writing a Component

- Know the purpose
- Have all the information
- Different strategies:
 - Write a draft using sentences
 - Jot down points in any form, then flesh out into sentences
 - Combination (sentences, phrases, points)

Overcoming Writer's Block

- Start *anywhere*
- If the problem is lack of information, go back and do more research
- Explain it to someone else (verbally)
- Work on a different section
- Take a walk (in a snow storm?)
- Imagine life when you are done

Overcoming Writer's Block II

- Force yourself to sit at your desk until done
- Revise your outline or organization
- Revise some section you've already written
- Change your environment
- Diagram the structure of the component
- Set an impossible schedule... and then panic!
- Take a break

Rhetorical Patterns

- Every culture has well-established patterns of exposition
 - Ready-made structures into which specific information may be dropped
- The reader is already familiar with these patterns
- The technical writer does not have the time (or skill) to invent new ones

General-to-specific Pattern

- Often used for introductory section
- Start with the most general statement
 - “More computing resources are devoted to the management of data than to any other task”
- Gradually get more specific
 - “This program simplifies the manipulation of numeric data on a personal computer”
- Finally specific statements

Classification Pattern

- Organize information by dividing it into categories
 - E.g., a section on each of initialization, data entry, selection, access control, etc...
- Within each category, present parallel information
 - E.g., purpose, prerequisites, results, error messages, alternatives, references, etc...

Comparison-contrast Pattern (Point-by-point)

- Consider one aspect at a time
 - In football, the ball may be thrown forward from behind the line of scrimmage
 - In rugby, only lateral passes are allowed
 - In football, play ends when the ball-carrier is tackled
 - In rugby, play continues after a tackle, but the tackled player must release the ball

Comparison-contrast Pattern (Whole-to-whole)

- Two ways to create a new object (zack)
 1. From the command line
 - On the command line, type “edit zack”
 - Fill in attributes of the presented template
 - Select Save from the File menu
 2. From within the application
 - Select New from the File menu
 - Fill in attributes of the presented template
 - Select Save As from File menu, and type “zack”

Definition Pattern

- Typically short and simple
- Example
 - “*Undo* is a function that restores an object to its state immediately prior to that last operation”
 - Places “Undo” in the class of functions, then distinguishes it from other functions

Chronological Pattern

- Typical for task-oriented instructions
- Given in the order in which they must be performed

Cause-and-effect Pattern

- Often used for error messages
- Give a list of error messages
 - ordered alphabetically, by error number, ...
- For each message, list the possible causes
 - ordered most to least likely
- After each cause, give the action(s) the user should take to recover

Putting Components Together

- Add headings (part, chapter, section, ...)
- Add transitions where needed
 - Important to prompt reader for what to expect, or to reinforce that some change is coming

Possible Transitions

- Moving to the next point in a sequence
 - “Firstly, secondly, ...”
- Contrasting item or viewpoint
 - “However, on the other hand, otherwise, ...”
- A result or conclusion
 - “Therefore, in consequence, ...”
- Relating things in time
 - “Now, then, soon, immediately, ...”

Possible Transitions

- Introducing an example
 - “For example, that is, ...”
- Further strengthening a point
 - “Moreover, similarly, further, ...”
- Concluding
 - “In conclusion, in summary, finally, ...”

Preliminary Draft

- Starting is always difficult
 - Helps to remember it's just a draft!
- Don't worry about spelling, grammar, form
- Spend effort on sound communication of major points
- Fill in your outline

Middle Draft

- Build on the base of the preliminary draft
- Refine the organization and fill in points
- Ensure each point belongs in that paragraph
- Cut and paste
- Play with the text
 - Font, layout, spacing, page count

Final Draft

- Spelling and grammar
 - Run the spell checker, but that's not enough!
- A “which” hunt
- Word choice
- Transitions
- Typos
- Pagination

Revising

- Where bad writing becomes good writing
- First draft is always bad
 - Tempting to become attached to text we've written
 - Write the first draft anticipating that it will change in the future

Revising Tasks

- Add flow and smooth transitions
- Careful, accurate movement from one point to the next, one section to the next
- Make decisions that have been put off
- Reduce wordiness
- Clarify subordination relationships between points