

## CS 350, slide set 9

M. Overstreet  
Old Dominion University  
Spring 2006

---

---

---

---

---

---

---

---

## Term Project Requirements - 1

- Overview:
  - Loop M times
    - Generate test data: N numeric values
    - Store N values in a file
    - Run P different programs
      - Each read test data
      - Executes
      - Store output in a file
    - Read output files to compare output from M programs
      - Report differences
      - Count differences
  - Compute simple estimate off reliability of each program

---

---

---

---

---

---

---

---

## Requirements – 2 simplifying assumptions

- All programs to be tested read from standard input
- Test programs have a fixed set of names: pgm01, pgm02, pgm03, etc.
  - Not desirable, but simplifies coding
- All input is numeric.
- All programs read their input until EOF.
- All output is numeric.
  - But you don't know how much!
- These programs may be buggy, but are all supposed to do the same thing.

---

---

---

---

---

---

---

---

### Requirements - 3

- Your program reads 3 values from the command line:
  - M, the outer loop parameter: the number of test cases
  - N, the number data values per test case
  - P, the number of programs. Assume P is at least 2.

---

---

---

---

---

---

---

---

### Requirements – 4 Program Outline

- Main
  - Loop M times:
    - Generate 1 test case with N numbers
    - Run each of P programs
      - Each program reads test case, produces output
    - All P output files are compared & differences reported
  - Generate reliability estimates for each of the P programs

---

---

---

---

---

---

---

---

### Requirements – 5 Random Values

- Use an existing random number generator
  - do "man random" on a UNIX box
- You should generate values between 0 and 10.
- For debugging, you want to be able to generate exactly the same set of random numbers on different runs.
  - Set the seed explicitly to a constant.

---

---

---

---

---

---

---

---

## Requirements – 6

### Computing Reliability - 1

- We will compute only an **estimate** of reliability.
- If one program produces an output that disagrees with all other programs and they agree, count this as an error for that single program.
- If more than one program produces a different value, assume all programs are in error. See example next slide
- Two programs is a special case. You can count either way, but be sure you can handle this.

---

---

---

---

---

---

---

---

## Requirements – 7

### Reliability - 2

If we're testing five programs:

Prog	p1	p2	p3	p4	p5	Comment
val 1	2	2	2	2	2	no error detected
val 2	-1	-1	-1	586	-1	p4 error
val 3	-1	-2	-1	586	-1	all programs in error
val 4	5	5		5	5	p3 error

---

---

---

---

---

---

---

---

## Requirements – 8

### Comparing Floats

- For coding simplicity, assume all data to be compared are floats
  - Would be better to treat integer values differently from float (or double) values
- Assume two floats are equal if they differ by less than 0.01%.
  - Use pgm01 output as the basis for the percentage comparison.

---

---

---

---

---

---

---

---

## Hints: use **system** OS call

- Do "man system" on a UNIX box for documentation.
- Example: to run program **abc** from a C++ program and have it read from file **Input** (as standard input) and write to file **Output** as standard output, use:  

```
system( "./abc < Input > Output" );
```
- System is an OS call to the operating fork a new shell and run the parameter in that shell as if you had typed that command.

---

---

---

---

---

---

---

---

## Use of files

- Data generator module generates one file
- That file is read repeatedly, once by each program being tested.
- Testing generates several output files, one per test program
- The comparison module:
  - Reads one value from each output file
    - Loop using array of files.
  - Compare values, reports and counts
  - Loops until end-of-file

---

---

---

---

---

---

---

---

## High-level Design: Modules

- Main
  - Outlined above
- Test data generator
- Loop module
  - Runs each of the programs
- Comparison module
- Report module
- Each of these are small, easily designed, easily reviewed and inspected
  - For some you have a base or a reuse source

---

---

---

---

---

---

---

---

## Intent

- Design, coding, testing should be easy
  - Including reviews and group inspections
- Unit testing straightforward
- Integration testing straightforward
- So focus on:
  - Group cooperation, group mutual aid
    - Inspect together before running
  - Estimates, scheduling, data collection, forms

---

---

---

---

---

---

---

---

## What's hard to code? - 1

- What happens if one version fails and produces no output?
- What happens if one version loops?
- What happens if one version incorrectly produces nonnumeric output?
- Answer: ignore problems 2 and 3; handle 1.

---

---

---

---

---

---

---

---

## What's hard to code? - 2

- Use of UNIX /tmp directory for random data files
  - Advantage: cleaned out by OS
  - You're not required to do this, but it's the right way
- If you do this, must generate a unique file name.
  - Normally, the process id of the generating program is part of the file name
- Good news: you don't need to do this.

---

---

---

---

---

---

---

---

### What's hard to code? - 3

- Unexpected end-of-file
- What if the different output files have different numbers of values?
  - Then hit end-of-file on some but not all
- Suggestion: get a version running that handles the case where all files have the same number of values and make sure that works.
  - Then add functionality. But don't break anything.

---

---

---

---

---

---

---

---

### Immediate steps! - 1

- Team and Individual Goals
  - Due by next Wed.
- High level design
  - What are the modules?
- Who codes what?
- Size estimates for each module
  - Sum for project size estimate
- Time estimates
  - Group discussed, approved

---

---

---

---

---

---

---

---

### Immediate steps! 2

- Schedule; group discussed & approved
  - What's to be completed when? Main mileposts:
    - Date each module designed & approved
    - Date each module will be coded
    - Date each module will be tested
    - Date all code integrated
    - Date test plans approved by group
    - Date testing complete
    - Date reporting complete
- What's omitted (& not required)
  - Code review dates & forms
  - Code inspection dates & forms

---

---

---

---

---

---

---

---

## Testing hints - 1

- Unit testing is required. So you must provide a driver and test data for each unit.
  - Are any stubs required?
- Separate compilation is required
  - Different team members code and test different units
  - Will give examples in class & on web
- To test your program, you will need to have several programs for it to run.
  - Consider using existing programs.
  - If you write them, make them as simple as possible as long as you can use them to test your code thoroughly.

---

---

---

---

---

---

---

---

## Testing hints - 2

- Consider programs, each consisting of one line:
  - `pgm1: cout << "1 2 3" << endl;`
  - `pgm2: cout << "1 2 3" << endl;`
  - `pgm3: cout << "1 2 4" << endl;`
  - `pgm4: cout << "1 2" << endl;`
  - etc. as useful for testing
- How many requirements can you test using programs like these?

---

---

---

---

---

---

---

---

## Simplified Task Planning Template (p. 82)

- Rows: include **only**
  - Plan-tasks and schedule
  - System test plan
  - Only one line** per module (but see next slide)
  - Postmortem
  - Totals
- Columns: include **only**:

Task Name	Planned Value
Plan hours by person	Cumulative PV
Total team hours	Actual hours
Cumulative hours	Cumulative hours
Size	Actual Completion Date
Planned Completion Date	

---

---

---

---

---

---

---

---

## Task Plan/Report per Module

Task	Size	Who	Plan-hrs	PV	Act-hrs	Plan com. date	Act. com. date	EV
Design								
Design rev								
Code								
Code rev.								
Compile								
Code insp.								
Test data								
Test driver								
Test rev.								
Test insp.								
Testing								

---

---

---

---

---

---

---

---

---

---

---

---

## Another omission

- Given the limited time:
  - Log where errors are **found** only
  - Omit noting where errors are inserted
- You must report errors found for each task
  - Some tasks may be unlikely to discover errors, but also include them in your error report.

---

---

---

---

---

---

---

---

---

---

---

---

## Comments

- Make sure you know what data you will need to report so it will take less time to provide it!
  - Look over the forms NOW!
- You set your own schedules
  - But we will be checking on what you submit & prompting for things!
- Project is about TSPI (simplified), not coding
  - Eval. biased on determining if you followed TSPI!
    - Did you make reasonable plans (time, sizes, req. tasks)?
      - Were you able to handle the inevitable surprises?
    - Did you set goals, design, inspect, test, record errors, log time, etc.?
    - Did you measure your performance against your goals?
      - Did you support your team?

---

---

---

---

---

---

---

---

---

---

---

---

## Reading

- TSP text, omit Ch. 6; central part of CS 410
- TSP text: Ch. 7, 8, 9, 10, App. B (config. mgmt) will be discussed in class
- TSP text: Read Ch. 16 (Managing yourself), Ch. 17 (Being on a team), & Ch. 18 (Teamwork)

---

---

---

---

---

---

---

---

## What's due when?

- April 30
- Will provide checklist on web site
- Submit all completed components (code, forms, etc. )to userid cs350
  - The grader will check for them there

---

---

---

---

---

---

---

---

## Topics

- Designing in teams
- Design script
- Implementation
- Implementation script

---

---

---

---

---

---

---

---

### Comments on team design

- Not easily done by large group
  - Some believe all good designs are done either by one individual or at most by a very small team
- Design can start with brain-storming session
  - All team members have input and contribute ideals
  - Then 1 or 2 people create the design

---

---

---

---

---

---

---

---

### Design completion details

- Goal: Inspection team can determine correctness of design
- Programmers can produce the intended implementation directly from the design and their background knowledge
  - Sometimes reference materials may be needed
  - And domain knowledge

---

---

---

---

---

---

---

---

### Design standards

- Naming conventions
- Interface formats
- System and error messages
- Design representation notations

---

---

---

---

---

---

---

---

## Levels of design

- Common levels:
  - System
  - Subsystem
  - Product
  - Component
  - Module
- For term project, you only need
  - System (mostly done)
  - Module (for each selected module)

---

---

---

---

---

---

---

---

## Design goals

- Reuse
  - Depends in part on standard documentation
- Usability
  - Need to review design from perspective of all people who will use the software
- Testability
  - May choose one design over another

---

---

---

---

---

---

---

---

## Design script - 1

Step	Activity	Description
1	Design process review	Review design process <ul style="list-style-type: none"> <li>■ Review sample design</li> <li>■ How design inspection is performed</li> </ul>
2	High-level design	Develop manager leads design team <ul style="list-style-type: none"> <li>■ Define program structure</li> <li>■ Name program components</li> <li>■ Identify design tasks to be completed and documented</li> </ul>
3	Design standards	Quality/process manager leads the effort to produce the name glossary and design standards
4	Design tasks	The development manager leads the team through <ul style="list-style-type: none"> <li>■ Outlining the SDS and the work to produce it.</li> </ul>
5	Task allocation	The team leader help allocate task among team members <ul style="list-style-type: none"> <li>■ Also obtains commitments from members</li> </ul>

---

---

---

---

---

---

---

---

## Design script - 2

Step	Activity	Description
6	Design specification	Each team member <ul style="list-style-type: none"> <li>■ Produces and reviews his/her portions of the SDS document</li> <li>■ Provides these to development manager</li> </ul> Development manager produces composite SDS draft
7	Integration test plan	Development manager leads team in producing and reviewing integration test plan
8	Design & int. plan inspection	Quality/process manager leads team through inspecting the SDS draft and integration test plan <ul style="list-style-type: none"> <li>■ Design is complete and correct</li> <li>■ Integration test plan is adequate</li> <li>■ Each problem is recorded and fix responsibility decided</li> </ul> Inspection is documented in form INS, defects recorded in LOGD

---

---

---

---

---

---

---

---

## Design script - 3

Step	Activity	Description
9	Design update	Development manager SDS obtains sections and <ul style="list-style-type: none"> <li>■ Combines them into final SDS</li> <li>■ Verifies traceability to the SRS</li> </ul>
10	Update baseline	Support manager baselines the SDS

---

---

---

---

---

---

---

---

## Standards

- Design standards
  - Let's assume pseudocode is good enough
    - Use another if you prefer
- Coding standards
  - What we have is good enough
- Size standards
  - Max size of a single component?
- Defect standards
  - What we have is good enough
- Standards need to be reviewed and changed if they don't work
  - But not this semester

---

---

---

---

---

---

---

---

## IMP script - 1

Step	Activity	Description
1	Implementation process overview	Instructor has described <ul style="list-style-type: none"> <li>■ Importance of quality implementation</li> <li>■ Need for coding standards</li> <li>■ Strategy for handling poor-quality components</li> </ul>
2	Implementation planning	Development manager leads work to <ul style="list-style-type: none"> <li>■ Define and plan implementation task (SUMP &amp; SUMQ)</li> </ul>
3	Task allocation	Team leader helps allocate tasks to team members <ul style="list-style-type: none"> <li>■ Obtains commitments for when they will complete tasks</li> </ul>
4	Detailed design	Programmers produce detailed designs <ul style="list-style-type: none"> <li>■ They do individual thorough design reviews</li> <li>■ They complete LOGD and LOGT forms</li> </ul>
5	Unit test plan	Tester produces unit test plans

---

---

---

---

---

---

---

---

---

---

## IMP script - 2

Step	Activity	Description
6	Test development	Testers follow script UT to produce unit test plans, test procedures, and test data for each component
7	Test inspect.	Quality/process manager leads team in inspection of test cases, procedures and data (use INS script and forms INS and LOGD) for each component
8	Detailed design inspect.	Quality/process manager leads team in inspection of DLD of each component (script INS and forms INS and LOGD)
9	Code	Programmers produce component source code <ul style="list-style-type: none"> <li>■ Do a code review using personalized checklist</li> <li>■ Compile and fix the code until it compiles</li> <li>■ Complete LOGD and LOGT forms</li> </ul>
10	Code inspect.	Quality/process manager leads team in code inspection of each component (script INS and INS and LOGD forms)

---

---

---

---

---

---

---

---

---

---

## IMP script - 3

Step	Activity	Description
11	Unit test	Programmers, following script UT <ul style="list-style-type: none"> <li>■ Conduct the unit tests and complete forms LOGD and LOGT</li> </ul>
12	Component quality review	Quality/process manager reviews each component's data to determine if component quality meets establish team criteria <ul style="list-style-type: none"> <li>■ If so, component is accepted for integration testing</li> <li>■ If not, quality/process managers recommends either:               <ul style="list-style-type: none"> <li>the product be reworked and reinspected, or</li> <li>the product be scrapped and redeveloped</li> </ul> </li> </ul>
13	Component release	<ul style="list-style-type: none"> <li>■ When components are satisfactorily implemented and inspected, developers release them to support manager</li> <li>■ The support manager enters the components in the configuration management system</li> </ul>

---

---

---

---

---

---

---

---

---

---

### Some goals/guidelines

- Design time > coding time
- Design review time > 50% of design time
- Code review time > 50% coding time, preferable > 75%
- You should find twice as many defects in code review as compile
- You should find > 3 defects / review hr.
- Review rate < 200 LOC/hr.

---

---

---

---

---

---

---

---