CSE306 Software Quality in Practice

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
May 8 class

I have arranged for several folks from local industry to come talk to us on Wednesday May 8.

They will describe their work environment, tools, processes, workflow, etc, and we'll have time for Q&A.

Keep in mind that these people are taking unpaid time off from work to share their experiences with you.

Attendance on this day is expected, and attendance will be taken.

As an added incentive, everyone present will earn an extra 3 points on their course grade (out of 100).
LPR ground rules

You may

- view tool documentation
- view your prior code in BitBucket or on timberlake
- consult your own hand-written notes

You may NOT

- communicate verbally, on-line, or otherwise, with others (sole exception: CSE306 UTAs)
- share code with others
- receive code from others
Exercise:
What is your process?
PROCESS

1.) Set up version control
   (e.g. git on BitBucket)
2.) Set up trello & share
3.) Simple makefile
4.) Understand requirements
5.) Blackbox testing
6.) Implement functionality
7.) Whitebox testing
8.) Utilize debug tools (-gcov, val)

9.) Reflect

branch

Commit freq.
Process:
1) Understand the requirements
2) Repository -> Bitbucket
3) Scrum Board -> Trello
4) Makefile -> Compilers
5) Stub-out code (C / gd6)
6) Black Box Tests (Unit)
7) Implementation -> C / gd6
8) White Box Tests -> gcov
9) Profiling -> gprof / callgrind
10) Memory checking -> Valgrind

Dev Process
1. Understand the requirements
2. Planning / Trello board / create repo
3. Create Source and header files ( stubs )
4. Create Makefile
5. Create blackbox tests
6. Implement Code
   6a. If bugs create bug fix branch
7. Whitebox testing
8. Run gprof, gcov, gdb, valgrind
9. If not complete do steps 7 and 8 one again
1. Understand the requirement

2. Bitbucket

3. Trello

4. Stubs (*.h and *.c)

5. Create make file

6. TDD (blackbox testing)

7. Implementation

8. Whitebox testing

9. gcov, gprof

10. Callgrind, Valgrind (memory check)

11. Bngfix via gdb (if exists)

12. Merge branches (if exist) & Commit

- Understand the requirements
- Trello Board - update frequently
- BitBucket - share w/ TAs
- Makefile
- BlackBox Testing
- WhiteBox Testing
- GDB
- Memcheck/Valgrind
- Gcov/GProf - output file
- Commit regularly
- Compile & run
Stubbing out functions

double foo(char * s, int x) {
    return 0.0;
}

char * bar(...) {
    return NULL;
}

Commit comments
1) What tool did you use & why
2) Results of tool captured in file & committed to repo
3) What will you do in response to tool output.

What to document in commit comments
Essential tools

- compiler (e.g. gcc)
- debugger (e.g. gbd)
- memory checker (e.g. memcheck (valgrind))
- runtime profiler (e.g. gprof, callgrind (valgrind))
- automated testing framework (e.g. cunit)
- build tool (e.g. make)
- code repository (e.g. git)
- collaboration (e.g. git, Trello)
The 13 Golden Rules of Debugging

1. Understand the requirements
2. Make it fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate facts from interpretation
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didn’t fix it, it ain’t fixed
13. Cover your bugfix with a regression test

Blackbox tests
Add failing test to reveal bug
Remove irrelevant details
All but first are questionable
Check the obvious things first
Make decisions based on facts
Be methodical, use process of elimination
Use all available tools/techniques
Verify behavior after *each* change
Document all efforts
Seek advice when you've done your homework
Intermittent bugs will recur
BB & WB tests
Process Review
(a.k.a. LPR prep)

- Create repo (git)
- Set up collaboration/organizer board (trello)
- Add/refine task(s) for board
- make makefile
- stub out functions
- write blackbox test (cunit)
- implement code (editor, compiler, cunit) / debug
code (gdb, memcheck) / use branches appropriately
- write whitebox test (gcov)
- performance test (gprof, callgrind)