Large-Scale Programming and Testing

Fall 2017 – CSCI 4963/6963 – Week 02
David Goldschmidt – goldschmidt@gmail.com
Office: Amos Eaton 115
Office hours: Mon/Thu 1:00-1:50PM; Wed 1:00-2:50PM

Code reviews

- A code review is an organized analysis of a specific portion of code with the overarching goal of improving the quality of the code/design
  - Involves multiple developers (therefore also called a peer review)
  - Encourages egoless programming
  - Provides a non-threatening collaborative environment
  - Should focus on a reasonable amount of code (e.g., <500 LOC)
  - Encourages the use of best practices
  - Identifies bugs and defects
  - Oftentimes, defects exist at the “edges” of functions, methods, interfaces, APIs, modules, etc.
Conducting good code reviews

- Ensure a productive code review by first focusing on logistics:
  - Schedule code reviews ahead of time
  - Beyond the developer(s) of the given code, invite other developers at all levels; having a mix of new and veteran coders is a good recipe
  - Limit the number of invites to ensure efficiency
  - In the invitations, briefly describe the code to be reviewed
  - Structure the meeting by first describing a few ground rules
  - Identify how defects will be documented and fixed
  - Regardless of your role in the code review, don’t be shy!

Conducting good code reviews

- Ensure positive results from a code review by next focusing on code consistency and the “easy” catches:
  - Start with a broad first pass through the code
  - Focus on comments, looking for typos, incoherence, ambiguities, comments that do not match the code, and other nonsense!
  - Look to omit commented-out sections of code
  - Review unclear variable, function, method, or class names
  - Identify functions (or other logical units) that are doing too much (and suggest ways to subdivide into smaller logical units)
  - Catch any style guide violations (some of this may be automated)
  - Play nice and be positive
Conducting good code reviews

- Dig deeper into the code to identify ways to improve it:
  - Look for opportunities for code reuse
  - Identify edge cases and how the code will react
  - If code is difficult to understand or requires a lot of explanation, it’s likely an implementation problem (so consider a revision)
  - Offer alternative approaches for refactoring that will improve readability, extensibility, and therefore maintainability
  - Ask whether the given code (or function or module or etc.) can be further generalized to handle future larger-scale requirements

Conducting good code reviews

- Overall, encourage egoless programming:
  - Question everything, but be nice about it (e.g., try asking “have you thought about ___”)
  - Do your best to not be defensive when your code is reviewed (oftentimes this requires you to be quiet during the review!)
  - Know that there are bugs and defects lurking that, when caught at this early stage, will have tremendous savings later
  - Trust that spending time doing code reviews increases productivity
  - At the end of the code review, thank all of the participants and summarize both the findings and next steps
Misquoting Drucker: “You can’t manage what you don’t measure.”

For any business process (e.g., code reviews), identifying and recording metrics can help to improve the given process

- Are there aspects of our code reviews that need improvement?
- How do we know we are improving? And what are we improving?

Example code review metrics:
- Defect rate: the number of defects identified per hour
- Defect density: the number of defects identified per LOC
- Code review efficacy: the number of defects identified per participant

Write perfect C code to solve the three problems below.

- You are not allowed to use any library functions from string.h
- Bring a printed copy of your solutions to our next class

Write a function to determine whether a given string called needle is a substring of another string called haystack

Write a function to count how many times a given string called needle is a substring of another string called haystack

Write a function to determine the longest palindrome in a given string called wow
From last time....

- Write perfect C code to solve the three problems below.
  - You are not allowed to use any library functions from \texttt{string.h}
  - Bring a printed copy of your solutions to our next class

- Write a function to determine whether a given string called \texttt{needle} is a substring of another string called \texttt{haystack}
- Write a function to count how many times a given string called \texttt{needle} is a substring of another string called \texttt{haystack}
- Write a function to determine the longest palindrome in a given string called \texttt{wow}

**Code Reviews:**

1. Form groups of 3-8 students each
2. Select (randomly?) someone's code to review
3. Select (also randomly?) someone to document each identified defect (and possible solution)
4. The developer who wrote the code has \texttt{~2} minutes to introduce and describe his or her code
5. Participants take turns critiquing the code, asking questions, making suggestions, etc.
6. The note-taker summarizes the findings and calculates defect density and code review efficacy
7. Go back to step (2)

**Side effects of code reviews**

- Beyond just improving the quality of the reviewed code, other side effects include:
  - Incorporating new developers (or old developers new to your team) into the team, into the code base, into the developer culture, etc.
  - Sharing changes and general knowledge of the system, thereby increasing the knowledge of all developers and eliminating single points of failure
  - Encouraging collaboration among team members
  - Leveling the playing field among new and veteran developers
  - Helping to achieve an underlying consistency and uniformity in the (ever-expanding) code base