Examining the Code

[Reading assignment: Chapter 6, pp. 91-104]
Static white-box testing

• Static white-box testing is the process of carefully and methodically reviewing the software design, architecture, or code for bugs without executing it.

• Unfortunately, static white-box testing is rarely done in practice (unlike dynamic black-box testing).
Formal code reviews

• A formal code review is the process under which static white-box testing is performed.
  – Can be a simple one-on-one meeting or a detailed rigorous code inspection.
  – May be organized by the programming or the testing team.
Essential elements of a formal code review

• Identify problems:
  – Find problems with the software such as missing items, mistakes, etc.

• Follow rules:
  – Amount of code to be reviewed, how much time will be spent, etc.

• Prepare:
  – Each participant should prepare in order to contribute to the review.

• Write a report:
  – Summarize the results of the review, make report available to the development team.
Informal code inspections

• **Peer reviews:**
  – An informal small group of programmers and/or testers act as reviewers.
  – Participants should follow the 4 essential elements even through the review is informal.

• **Walkthroughs:**
  – A more formal process in which the author of the code formally presents the code to a small group of programmers and/or testers.
  – The author reads the code line by line explaining what it does, reviewers listen and ask questions.
  – Participants should follow the 4 essential elements.
Formal code inspections

- *Code presenter* is not the author of the code.
- The other participants are the *inspectors*.
- There is a *moderator* to assure that the rules are followed and the meeting runs smoothly.
- After the inspection a report is composed. The programmer then makes changes and a re-inspection occurs, if necessary.
- Formal code inspections are effective at finding bugs in code and designs and are gaining in popularity.
Code review checklist: Data reference errors

- Is an un-initialized variable referenced?
- Are array subscripts integer values and are they within the array’s bounds?
- Are there off-by-one errors in indexing operations or references to arrays?
- Is a variable used where a constant would work better?
- Is a variable assigned a value that’s of a different type than the variable?
- Is memory allocated for referenced pointers?
- Are data structures that are referenced in different functions defined identically?
Code review checklist: Data declaration errors

• Are the variables assigned the correct length, type, storage class?
  – E.g. should a variable be declared a string instead of an array of characters?
• If a variable is initialized at its declaration, is it properly initialized and consistent with its type?
• Are there any variable with similar names?
• Are there any variables declared that are never referenced or just referenced once (should be a constant)?
• Are all variables explicitly declared within a specific module?
Discussion

• Why is limiting the scope of a variable a good thing for testing?
• What do you do when you want to share data among many modules of your code?
• What is your opinion of the programming language feature ‘declaration upon use’?
Code review checklist: Computation errors

- Do any calculations that use variables have different data types?
  - E.g., add a floating-point number to an integer
- Do any calculations that use variables have the same data type but are different size?
  - E.g., add a long integer to a short integer
- Are the compiler’s conversion rules for variables of inconsistent type or size understood?
- Is overflow or underflow in the middle of a numeric calculation possible?
- Is it ever possible for a divisor/modulus to be 0?
- Can a variable’s value go outside its meaningful range?
  - E.g., can a probability be less than 0% or greater than 100%?
- Are parentheses needed to clarify operator presence rules?
Code review checklist: Comparison errors

• Are the comparisons correct?
  – E.g., < instead of <=

• Are there comparisons between floating-point values?
  – E.g., is 1.0000001 close enough to 1.0000002 to be equal?

• Are the operands of a Boolean operator Boolean?
  – E.g., in C 0 is false and non-0 is true
Code review checklist: Control flow errors

- Do the loops terminate? If not, is that by design?
- Does every `switch` statement have a `default` clause?
- Are there `switch` statements nested in loops?
  - E.g., careful because `break` statements in `switch` statements will not exit the loop ... but `break` statements not in `switch` statements will exit the loop.
- Is it possible that a loop never executes? If it acceptable if it doesn’t?
- Does the compiler support short-circuiting in expression evaluation?
Code review checklist:
Subroutine parameter errors

• If constants are passed to the subroutine as arguments are they accidentally changed in the subroutine?
• Do the units of each parameter match the units of each corresponding argument?
  – E.g., English versus metric
  – This is especially pertinent for SOA components
• Do the types and sizes of the parameters received by a subroutine match those sent by the calling code?
Code review checklist: Input/Output errors

• If the file or peripheral is not ready, is that error condition handled?
• Does the software handle the situation of the external device being disconnected?
• Have all error messages been checked for correctness, appropriateness, grammar, and spelling?
• Are all exceptions handled by some part of the code?
• Does the software adhere to the specified format of the date being read from or written to the external device?
Code review checklist: Other checks

• Does your code pass the lint test?
  – E.g., How about gcc compiler warnings?
• Is your code portable to other OS platforms?
• Does the code handle ASCII and Unicode?
• How about internationalization issues?
• Does your code rely on deprecated APIs?
• Will your code port to architectures with different byte orderings?
  – E.g., little (increasing numeric significance with increasing memory addresses) versus big (the opposite of little) endian?
Discussion …

• If you had to make a choice whether to use dynamic black-box testing or static white-box testing, which would you choose and why?
• A common security vulnerability is buffer overflow. What testing strategy would best mitigate this category of vulnerability?
• Do programmers have a unique style? What would you look for if you were interested in software forensics analysis for code authorship identification?
You now know …

• … static white-box testing
• … code reviews
• … informal code inspections
• … formal code inspections
• … code review checklists