CUI Abbottabad

Department of Computer Science

SOFTWARE TESTING

Lecture 5 & 6

Code Inspection, Error Checklists, Walkthroughs, and Peer Rating

CONTENT

- Program/Code inspection
- Error Checklists
- Walkthroughs
- Peer Rating

CODE INSPECTION

- It is a step-by-step peer group review of a work product (code reading), with each step checked against predetermined criteria for error detection.
- Error Checklist is used to examine the program for common errors.

INSPECTION SESSION (INGREDIENTS)

- The time and location of the inspection should be planned to avoid all outside interruptions.
- The optimal amount of time for the inspection session appears to be from 90 to 120 minutes
- Most inspections proceed at a rate of approximately 150 program statements per hour.
- Large programs should be examined in multiple inspections, each inspection dealing with one or several modules or subroutines.

CODE INSPECTION TEAM

- An inspection team usually consists of four people.

1. Moderator is expected to be a competent programmer, but he/she is not the author of program and need not be familiar with program details. Moderator is like a quality-control engineer.

- **Duties of moderator:**
  - Distribute materials (program's listing, design specification etc.) in advance for scheduling the inspection session
  - Lead the session
  - Record all errors found
  - Ensure that the errors are subsequently corrected

CODE INSPECTION TEAM

2. Programmer: have written the code.

3. Program's designer: design the program and has understanding of software's architecture, design patterns, and functionality etc.

4. Test specialist: identify error, fault, and other issues that could impact software functionality / performance.

- ALL the participants are expected to familiarize themselves with the material prior to the session.

INSPECTION SESSION (ACTIVITIES)

- The programmer narrates statement by statement the program logic.
- Other participants raise questions, and determine whether errors exist.
- Program is analyzed with respect to a checklist of historically common programming errors.
- After the session, programmer is given a list of the errors found.
- If more than a few errors were found, or if any of errors requires a substantial correction, the moderator might make arrangements to reinspect the program after the errors are corrected. This list of errors is also analyzed, categorized, and used to refine the error checklist to improve the effectiveness of future inspections.

AN ERROR CHECKLIST FOR INSPECTION

- Inspection process make use of a checklist to examine the program for common errors.
- Unfortunately, some checklists concentrate more on issues of style than on errors (for example, “Are comments accurate and meaningful?” and “Are if-else, code blocks, and do-while groups aligned?”), and the error checks are too vague to be useful (such as “Does the code meet the design requirements?”).
- Below are some errors checklist that can occur with any programming language.

ERROR CHECKLIST (DATA REFERENCE ERRORS)

Data reference errors are bugs caused by using a variable, constant, array, string, or record that hasn't been properly declared or initialized for how it's being used and referenced.

- int[] arr;
  int[] arr= new int[10];
  arr[10] = 5;

Try to access 11th element that doesn't exist, so out of bounds exception will occur.

- String str = "123";
  int num = (int) str;

Here, an invalid cast exception will occur.

1. Unset variable used?
2. Subscripts within bounds?
3. Non integer subscripts?
4. Dangling references?
5. Correct attributes when aliasing?
6. Record and structure attributes match?
7. Computing addresses of bit strings? Passing bit-string arguments?
8. Based storage attributes correct?
9. Structure definitions match across procedures?
10. Off-by-one errors in indexing or subscripting operations?
11. Are inheritance requirements met?

ERROR CHECKLIST (COMPUTATION)

Computational or calculation errors are essentially bad math. The calculations don't result in the expected result.

- Adding a floating-point number to an integer.
- Adding a long integer to a short integer
- If a program encounters a divide-by-zero error, an unhandled exception may occur.

1. Computations on nonarithmetic variables?
2. Mixed-mode computations?
3. Computations on variables of different lengths?
4. Target size less than size of assigned value?
5. Intermediate result overflow or underflow?
6. Division by zero?
7. Base-2 inaccuracies?
8. Variable's value outside of meaningful range?
9. Operator precedence understood?
10. Integer divisions correct?

ERROR CHECKLIST (DATA DECLARATION)

Data declaration bugs are caused by improperly declaring or using variables or constants.

- Should a variable be declared a string instead of array of characters?
- Are all variables assigned the correct length, type, and storage class?
- If a variable is initialized at the same time as it's declared, is it properly initialized and consistent with its type?

1. All variables declared?
2. Default attributes understood?
3. Arrays and strings initialized properly?
4. Correct lengths, types, and storage classes assigned?
5. Initialization consistent with storage class?
6. Any variables with similar names?

ERROR CHECKLIST (COMPARISON)

Comparison Errors are caused by incorrect use of Less than, greater than, equal, not equal, true, false operator. Comparison and decision errors are very prone to boundary condition problems.

- < instead of <=
- 1.0000001 close enough to 1.0000002 to be equal?
- in C 0 is false and non-0 is true

1. Comparisons between inconsistent variables?
2. Mixed-mode comparisons?
3. Comparison relationships correct?
4. Boolean expressions correct?
5. Comparison and Boolean expressions mixed?
6. Comparisons of base-2 fractional values?
7. Operator precedence understood?
8. Compiler evaluation of Boolean expressions understood?

ERROR CHECKLIST (CONTROL FLOW)

Control flow errors are the result of loops and other control constructs in the language not behaving as expected. They are usually caused, directly or indirectly, by computational or comparison errors.

- while (true)
  { // do something repeatedly }

Here code creates infinite loop that will never terminate, causing program to hang or crash.

- for (int i = 0; i <= 10; i++) { }
  Here Off-by-one error will occur.

- if (false) {
  // do something
  } else {
  // do something else
  }
  Above code is dead code and first block will never execute as condition is always false.

1. Multiway branches exceeded?
2. Will each loop terminate?
3. Will program terminate?
4. Any loop bypasses because of entry conditions?
5. Are possible loop fall-throughs correct?
6. Off-by-one iteration errors?
7. DO/END statements match?
8. Any nonexhaustive decisions?
9. Any textual or grammatical errors in output information?

ERROR CHECKLIST (INPUT OUTPUT)

Input/Output Errors include anything related to reading from a file, accepting input from a keyboard or mouse, and writing to an output device such as a printer or screen.

- Does the software strictly adhere to the specified format of the data being read or written by the external device?
- If the file or peripheral isn't present or ready, is that error condition handled?

1. File attributes correct?
2. OPEN statements correct?
3. Format specification matches I/O statement?
4. Buffer size matches record size?
5. Files opened before use?
6. Files closed after use?
7. End-of-file conditions handled?
8. I/O errors handled?

ERROR CHECKLIST (INTERFACE)

Interface Testing is performed to evaluate whether systems or components pass data and control correctly to one another. It is to verify if all the interactions between these modules are working properly and errors are handled properly.

- interface MyInterface {
  void myMethod(int num);
  }
  class MyClass implements MyInterface {
  void myMethod(String str) {
  // do something with str
  } }

Here ***myMethod*** method has a different signature (i.e. parameter type) than interface method and will result in compilation error.

1. Number of input parameters equal to number of arguments?
2. Parameter and argument attributes match?
3. Parameter and argument units system match?
4. Number of arguments transmitted to called modules equal to number of parameters?
5. Attributes of arguments transmitted to called modules equal to attributes of parameters?
6. Units system of arguments transmitted to called modules equal to units system of parameters?
7. Number, attributes, and order of arguments to built-in functions correct?
8. Any references to parameters not associated with current point of entry?
9. Input-only arguments altered?
10. Global variable definitions consistent across modules?
11. Constants passed as arguments?

WALKTHROUGHS

- It is a review technique where the author leads the team through a manual or simulated execution of the product using predefined scenarios.
- The walkthrough is an uninterrupted meeting of one to two hours in duration.
- The walkthrough should have a follow-up process similar to that described for the inspection process. Also, the side effects observed from inspections (identification of error-prone sections and errors, style, and techniques) also apply to the walkthrough process.

WALKTHROUGHS TEAM

- Walkthrough team consists of 3-5 people. One of people plays a role similar to that of **moderator**, another person plays the role of a **secretary** (a person who records all errors found), and a third person plays the role of a **tester**.
- The **programmer** is one of those people.
- Suggestions for the other participants include:

(1) a highly experienced programmer,

(2) a programming-language expert,

(3) a new programmer (to give a fresh, unbiased outlook),

(4) the person who will eventually maintain the program,

(5) someone from a different project, and

(6) someone from the same programming team as the programmer.

WALKTHROUGHS EXECUTION IN SESSION

- Participants are given the materials several days in advance to allow them to bone up on the program.
- Here participants “play computer.”
- The person designated as the tester comes to the meeting armed with a small set of paper test cases—representative sets of inputs (and expected outputs) for the program or module.
- During the meeting, each test case is mentally executed. The test data are walked through the logic of the program.
- The state of program (i.e., values of variables) is monitored on paper or whiteboard.

PROGRAM INSPECTIONS AND WALKTHROUGHS

- Similarities among two methods are given below:
  - Inspections and walkthroughs involve a team of people reading or visually inspecting a program.
  - The climax is a "meeting of the minds," at a participant conference. Meeting objective is to find errors but not to find solutions to errors. Here test is done, not debug.
  - Both methods are an improvement over older desk-checking process (process of programmer reading his/her own program before testing it).
  - Inspections and walkthroughs are more effective, again because people other than the program's author are involved in the process.

PROGRAM INSPECTIONS AND WALKTHROUGHS

- These methods generally are effective in finding defects from 30 to 70 percent.
- They are not effective, however, in detecting high-level design errors, such as errors made in the requirements-analysis process of the logic-design and coding errors in typical programs.
- The implication is that inspections/walkthroughs and computer-based testing are complementary; error-detection efficiency will suffer if one or the other is not present.

PEER RATING

- Peer rating is used to evaluate anonymous programs in terms of their overall quality, maintainability, extensibility, usability, and clarity. Purpose is to provide programmer self-evaluation.
- A programmer is selected to serve as an administrator of process and administrator selects approximately 6 to 20 participants.
- The participants are expected to have similar backgrounds.
- Each participant is asked to select two of his/her own programs to be reviewed. Finest program and poor quality program.
- Once the programs have been collected, they are randomly distributed to the participants.
- Each participant is given four programs to review i.e. two “finest” programs and two “poorer” programs, but reviewer is not told about their quality.

PEER RATING

- Each participant spends 30 minutes with each program and then completes an evaluation form after reviewing the program.
- After reviewing all four programs, each participant rates the relative quality of the four programs.
- The evaluation form asks the reviewer to answer questions, on a scale from 1 to 7 (1 meaning definitely “yes, ” 7 meaning definitely “no”).
- Question include (Rated from 1-7 on Likert Scale) :
  - Was the program easy to understand?
  - Was the high-level design visible and reasonable?
  - Was the low-level design visible and reasonable?
  - Would it be easy for you to modify this program?
  - Would you be proud to have written this program?

PEER RATING

- The reviewer also is asked for general comments and suggested improvements.
- After the review, the participants are given the anonymous evaluation forms for their two contributed programs.
- The participants also are given a summary that shows the overall and detailed ranking of their original programs across entire set of programs, as well as an analysis of how their ratings of other programs compared with those ratings of other reviewers of the same program.
- Peer rating allow programmers to self-assess their programming skills. And process appears to be useful in both industrial and classroom environments.

REFERENCES

Book:

The Art of Software Testing, Second Edition, Glenford J. Myers

Chapter 3