SOFTWARE QUALITY ASSURANCE

Lecture 9

Instructor: Mr. Natash Ali Mian

Department of CS and IT
The University of Lahore

Switch off mobile phones during lectures, or put them into silent mode

Use Dustbins to throw waste materials

Term Paper

Finalize Group Members	26-Feb-2013
Finalize Topic	12-Mar-2013
Search Papers and Sort Selected (TODAY)	20-Mar-2013
Go Through the Abstract and Introduction of Selected Papers	27-Mar-2013
Submit a Summary and Comments on related papers	**09-Apr-2013**
Present Your Work till Today	**09-Apr-2013**
Submit Initial Draft	30-Apr-2013
Final Paper Submission	21-May-2013
Feedback on Final Submission + Plagiarism Report	28-May-2013
Final Presentation	4-June-2013

Please note that Every Phase has Marks

Contents

- Programming and Software Quality

Programming and Software Quality

PROGRAMMING

- The act of programming, also known as coding, produces the primary products – executables – of a software development effort
- All prior activities culminate in their development
- Programming is done in a programming language

Coding Defects - 1

- All four categories of defects are found in source code
  - Errors of commission
  - Errors of omission
  - Errors of ambiguity and clarity
  - Errors of speed and capacity
- Errors of commission are the most common when the code is underdevelopment

Coding Defects - 2

- The most surprising aspect of coding defects is that more than fifty (50) percent of the serious bugs or errors found in the source code did not truly originate in the source code
- A majority of the so-called programming errors are really due to the programmer not understanding the design or a design not correctly interpreting a requirement

Coding Defects - 3

- Software is one of the most difficult products in human history to visualize prior to having to build it, although complex electronic circuits have the same characteristic
- Built-in syntax checkers and editors with modern programming languages have the capacity to find many “true” programming errors such as missed parentheses or looping problems
- They also have the capacity to measure and correct poor structure and excessive branching

Coding Defects - 4

- The kinds of errors that are not easily found are deeper problems in algorithms or those associated with misinterpretation of design
- At least five hundred (500) programming languages are in use, and the characteristics of the languages themselves interact with factors such as human attention spans and capacities of temporary memory
- This means that each language, or family of languages, tends to have common patterns of defects but the patterns are not the same from language-to-language

Coding Defects - 6

- There is no solid empirical data that strongly-typed languages have lower defect rates than weakly-typed languages, although there is no counter evidence either
- Of course for all programming languages, branching errors are endemic. That is, branching to the wrong location for execution of the next code segment

Defects in High-Level Languages - 1

- Many high-level languages, such as Ada and Modula, were designed to minimize certain common kinds of errors, such as mixing data types or looping incorrect number of times
- Of course, typographical errors and syntactical errors can still occur, but the more troublesome errors have to do with logic problems or incorrect algorithms

Defects in High-Level Languages - 2

- A common form of error with both non-procedural and procedural languages has to do with retrieving, storing, and validating data
- It may sometimes happen that the wrong data is requested
- Programming in any language is a complex intellectual challenge with a high probability of making mistakes from time to time
- Analogy with typos in a newspaper

Defects in Low-Level Languages

- Since low-level languages often manipulate registers and require that programmers setup their own loop controls, common errors involve failure to initialize registers or going through loops the wrong number of times, not allocating space for data and subroutines
- For weakly-typed languages, mismatched data types are common errors

Quality Practices for General-Purpose Programming - 1

- Use the highest-level programming language possible
- Use integrated development environments
- Adopt a coding standard that prevents common types of defects

Quality Practices for General-Purpose Programming - 2

- Prototype user interfaces and high-risk components
- Define critical regions

Use the Highest-Level Programming Language - 1

- Code written in higher-level programming languages is easier to read and maintain
- Any fool can write code that a computer can understand. Good programmers write code that humans can understand

Use the Highest-Level Programming Language - 2

- Several practical factors influence the selection of a programming language
  - Technology trends
  - Organizational informational technology strategies
  - Customer restrictions on programming language selection
  - Experience of the development team
  - Features of the programming language (e.g., to interoperate with external systems)

Use the Highest-Level Programming Language - 3

- The complexity of software systems is growing quicker than our ability to develop software solutions
- For example, productivity of computer personnel increased about 6% per year during the 1990s, whereas the growth in NASA mission software is about 25% per year

Use the Highest-Level Programming Language - 4

- Productivity is constant in terms of program statement size. That is writing ten lines of code in assembly language requires as much work as writing ten lines of code in C++, but the functionality developed in ten lines of C++ is much more than the ten lines of assembly language code
- We are shrinking the size of the programs by using higher-level languages

Use the Highest-Level Programming Language - 5

- Fred Brooks has said that the advent of high-level programming languages had the greatest impact on software productivity because there was at least a factor of five improvement in productivity
- The use of high-level programming languages results in more reliable software

Use Integrated Development Environments

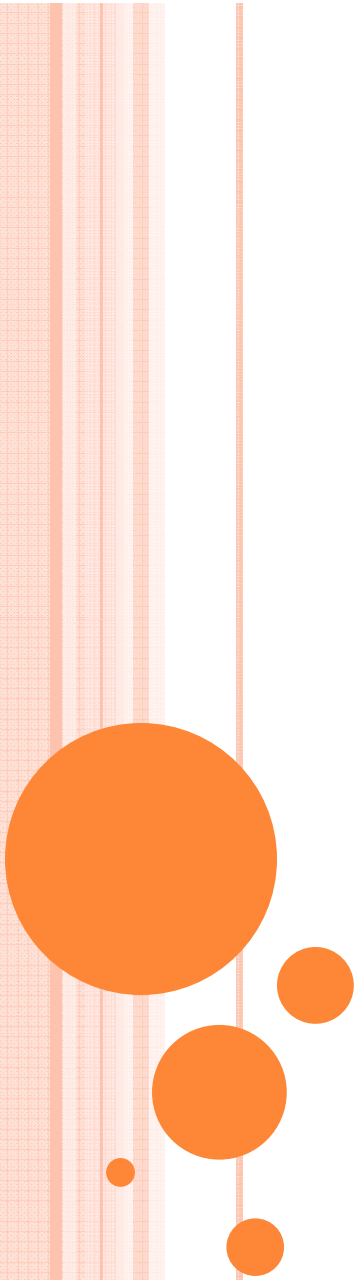
- Also known as IDEs, these suites include an editor, a compiler, a make utility, a profiler, and a debugger. Other tools may also be included
- Recent IDEs include tools to model software designs and implement graphical user interfaces
- These tools, if used properly, can improve the productivity 100%
- They also help identify many coding defects, as they are being introduced in the software

Adopt a Coding Standard to Prevent Common Types of Defects - 1

- Coding standards are controversial because the choice among many candidate standards is subjective and somewhat arbitrary
- Standards are most useful when they support fundamental programming principles

Adopt a Coding Standard to Prevent Common Types of Defects - 2

- So, it is easier to adopt a standard for handling exceptions, than for identifying the amount of white-space to use for indentation
- An organization should always ask itself whether a coding standard improves program comprehension characteristics



Quality Practices Related to Programming

Practices for Internal Documentation - 1

- Specify the amount of white-space that should be used and where it should appear
  - Before and after loop statements and function definitions
  - At each indentation level (two or four spaces have been reported as improving comprehensibility of programs)
- Physically offset code comments from code when contained on the same line

Practices for Internal Documentation - 2

- Use comments to explain each class, function, and variable contained in source code. (Comments can be from 10% and up)
  - Key interactions that a function has with other functions and global variables
  - Complex algorithms used by every function
  - Exception handling
  - Behavior and effect of iterative control flow statements and interior block statements

Practices for Internal Documentation - 3

- Provide working examples in the user documentation or tutorial materials

Practices for Variable Definition - 1

- Declare variables as specifically as possible and initialize them, preferably one declaration per line
- Do not use similarly named variables within the same lexical scope
- Consistently, use clear and easily remembered names for variables, classes, and functions

Practices for Variable Definition - 2

- Follow a uniform scheme when abbreviating name
- Do not use local declarations to hide declarations at greater scope
- Never use a variable for more than one purpose

Practices for Control Flow

- Do not assume a default behavior for multi-way branches
- Do not alter the value of an iteration variable within a loop
- Use recursion, when applicable

Practices for Functions

- Explicitly define input and output formal parameters
- Use assertions (e.g., pre- and post-conditions) to verify the accuracy and correctness of input and output formal parameters. The use of pre- and post-conditions helps programmers detect defects closer to their origin

Practices for Operations

- Make all conversion of data explicit, especially numeric data
- Do not use exact floating-point comparison operations
- Avoid using operators in potentially ambiguous situations

Practices for Exception Handling

- Process all exceptions so that personnel can more easily detect their cause
- Log important system events, including exceptions

Practices for Maintenance

- Isolate the use of nonstandard language functions
- Isolate complex operations to individual functions

Practices for Operational

- Do not permit any compilation to produce warnings
- Optimize software only after it works is complete, and only if required to achieve performance goals

Prototype User Interfaces and High-Risk Components

- User interface prototyping helps identify necessary features that software engineers might otherwise overlook
- Prototyping can reduce the development effort significantly
- Prototyping reduces development risk because is allows programmers to explore methods for achieving performance and other high-risk requirements

Define Critical Regions

- A task that interrupts an interdependent operational sequence before it is completed can leave a program in a vulnerable state, resulting in inconsistent and inaccurate results. We need a critical regions to run such transactions
- Critical regions help prevent deadlocks

References

- Software Engineering Quality Practices by Ronald K. Kandt (Ch. 8)
- Software Quality: Analysis and Guidelines for Success by Capers Jones
- Software Quality: Analysis and Guidelines for Success by Capers Jones
- Software Engineering: A Practitioner's Approach by Roger Pressman (Chapter 13)
- Software Engineering Quality Practices by Ronald K. Kandt


Thanks!