ICT 2402 Software Engineering

Software Implementation

Topics covered

- Selection of programming language
- Coding style and documentation guidelines
- Fault tolerance and exception handling
- Software libraries and the use of APIs
- Writing secure software
- Code reviews

Software implementation

- Starts after finalizing, reviewing and approving the software design
- Ideally, a straightforward process involving mapping the design into code if the software design is clear and unambiguous
- Many software errors are caused by poor implementation and poor testing that fails to detect implementation errors.

Selection of programming language

- Selection of the PL depends on several factors.
  - Nature of the application
  - Cost
  - Available resources
  - Client expectations
- Selection of PL may result in HR activities like recruitment and training

Coding style and documentation guidelines

- As a part of the software development plan, a coding standard specifying the style for writing code and documenting it should be adopted.
- Should be realistic, easy to follow and practical
- Adherence to standard checked by SQA and during code reviews using inspection checklists
- Short term cost vs. long term saving (understandability -> maintainability)

What can be standardized?

- Naming standards
- Commenting standards
- Formatting standards

Naming guidelines

- Guidelines for naming, Variables, procedures/functions/methods, classes, packages, user-defined types, tables etc.
  - Examples:
    - Integer name: int temp_i;
    - Long variable name: float avgTempDuringDay_f or avg_temp_during_day_f
    - Table name: STUDENT_TABLE or STUDENT_T
    - Integer array name: class_grades_ia or classGrades_ia

Commenting guidelines

- Comments enhance readability and future understandability and maintainability
  - Example:
    - Each defined variable name should be commented on same line
      - float class_average_f; // variable holds class average
- Comments must be updated when code is updated during maintenance

Formatting guidelines

- Good formatting increases readability
- Makes code structures more visible and identifiable
- Formatting guidelines may include,
  - Indentation
  - Spacing and use of new lines
  - Using new page for new module (function, method, ..)

Software Reuse

- Reusing trusted (well-tested) components increases software reliability
- Writing code with future reuse in mind
  - Write code with clear API
- Writing code from reusable code
  - Software libraries, OTS components, outsourced components

Application Programming Interface (API)

- API is key to proper re-usability;
  - Should be easy to learn and use, secure and hard to misuse, and easy to extend and adapt
  - Should be stable over long time
- Corporate asset: reusable component with clear API
- API and its implementation must reflect the requirements
- Functional and properly protected
  - Does one function only and hides private details

Template for documenting an API

API name	
Brief description	
Syntax or prototype (name and header)	
Typical use scenario	
Usage example	
Detailed description	
Arguments details	
Superclass(es)	
Error messages produced	
Exceptions thrown out	
Returned value	
Prerequisite or preconditions	
Special notes or warning on API use	

Software fault tolerance

- Software ability to deal with some errors and abnormal conditions
  - Detect, recover then continue operation
  - Fault tolerance software are also called: error-recoverable, self-stabilizing or resilient software
- Several approaches to build fault tolerance software,
  - Checkpointing
  - N-version programming
  - Recovery block approach

Checkpointing

- Running a checkpointing procedure to periodically save the 'state' of the software
- In case of a failure, two approaches may be followed to deal with the error and continue operation
  - Backward recovery-In case of failure, continue operating from last saved checkpoint
  - Forward recovery-Steps forward to future desirable state

N-version programming

- Provide N implementations of a design by N independent teams
- Each implementation provides its result (vote) to an arbiter – take a majority vote
- Fails if design is incorrect or the majority of implementations are faulty
- Alternative: M versions of the design, and M.N versions of the implementation

Recovery block

- Save the current valid state before entering the recovery block
- Inside block, N versions are listed in decreasing order of confidence in correctness
- If judge procedure accepts the result exit the block otherwise reset to saved state and proceed to next version
- If none is judged as correct, invoke an error handling procedure

Exception handling

- Dealing with simple run-time errors
  - Divide by 0, invalid array index
- Several types of exceptions,
  - Arithmetic exceptions – happens when there is an error in arithmetic operations specified in the code
  - ArrayIndexOutOfBound exceptions-when pointed to an unavailable index of an array
  - NullPointer exception – when specified a wrong or unavailable destination
  - NumberFormat exception-happens in the case of a wrong number format

Dealing with an exception

- Exception handler code (try-catch)
- 'try' code to include the code that may cause an exception; and a 'catch' block to handle that exception if it occurs at run time
- 'finally' code is executed in all cases – cleanup code
- A module that may throw an exception in one or more of its try or catch blocks should indicate this fact as part part of its signature

Cont...

```
try
{ // code that may cause or generate an exception
}
catch (class X) // exception of class X
{ // code for handling an exception of class X
}
catch (class Y) // exception of class Y
{ // code for handling an exception object of class Y
}
finally
{ // code to execute after the exception handling code is executed
// and before returning to the caller if no exception has occurred
}
```

Cont...

```
public void readUserProfile (String fileName) throws IOException
{
        if (fileName == null)
        {
                throw IOException();
        }
        ....
        InputStream input = new FileInputStream(fineName);
        ....
}
```

Writing Secure Software

- Software security vulnerabilities introduced during the implementation
- Can be avoided if secure coding practices are followed
- Otherwise should be removed after a code review
- Many classifications of software security errors exist

Software security vulnerabilities

- I/O validation and representation errors
- Security violations
- API abuse
- Coding errors and code quality

I/O validation and Representation errors

- Lack of input validation may lead to serious errors
- Buffer overflow – while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory
- Command injection attacks– inject malicious code into the system through input data
- Denial of service attacks– attempt to make a computer resource unavailable to its intended users.

Security violations

- Failure in using proper security functions to avoid attacks on confidentiality, integrity and availability
  - Improper or lack of access control
  - Use of weak random number generator
  - Violation of the least privilege principle
  - Misuse of private information
  - Improper management of user passwords
  - Improper use and creation of temporary files

API abuse

- API is a contract between caller and callee
- Caller deviates from the callee's API specs
- Bad implementation of the caller's module
- Should be detected during code review
- Example:
  - Caller ignoring a returned value by the callee
  - If a bad value returned and no action is taken by caller, errors could happen

Coding errors and code quality

- Errors that can be detected by static checking or code review
  - Invalid test conditions
  - Dead code
  - Unused variables
  - Uninitialized variables
  - Memory leak
  - Type mismatch

Code review

- Non-automatic process performed in a formal meeting
- Normally done prior to dynamic testing and after obtaining an error-free compilation
- Include code walkthrough and code inspection checklists
- SQA sets the rules for conducting the reviews
  - When code should be distributed
  - Who participates
  - Review decision and follow ups

Participants in a code review

- Programmer who wrote the code
- Designer
- Tester
- SQA
- Review leader (could be SQA)
- Review recorder

Questions?