Slide 16 and 17: Coding

Made by Md. Mehedi Hasan Rafy

Coding

- ***Objective:*** turn design of a system into code in a HLL and then to unit test this code.
- Must follow a coding standard for consistency.
- ***Benefits:*** uniform look of the code by different engineer, easier understanding, encourages good practices.

Characteristics of a Good Programming Language

1. ***Readability*** – code close to English, self-documenting.
2. ***Portability*** – machine independent.
3. ***Generality*** – supports wide variety of programs.
4. ***Brevity*** – fewer lines for same logic.
5. ***Error checking*** – compile-time + run-time errors.
6. ***Cost*** – overall efficiency.
7. ***Familiar notation –*** helps programmers to understood the language easily.
8. ***Efficiency*** – Efficient object code.
9. **Modularity** – Programs can be written in separate modules.
10. ***Quick Translation*** – must admit quick translation.

Coding Standards

- Limiting the use of global variables.
- Module headers should include: name, date, author, history, synopsis, functions, I/O params, globals used.
- Naming conventions: globals (Capital), locals (small), constants (ALL CAPS).
- Error handling conventions: consistent return/error mechanisms (e.g. functions while encountering error should return a 0 or 1).

Coding Guidelines

- Avoid cryptic code (too difficult to understand code).
- Avoid obscure side effects (e.g., hidden global changes).

Modifying a non-local variable:

```
1  x=3
2  def square(x):
3      x=x*x
4      return x
5  def square_side_effect():
6      global x
7      x=x*x
8      return x
9  print(x)                    # 3
10 print(square(x))            # 9
11 print(x)                    # 3
12 print(square_side_effect())     # 9
13 print(x)                    # 9
```

- Don’t reuse identifiers for multiple purposes because it leads to confusion and makes future enhancement difficult.
- Use descriptive names.
- Well-documented: ~1 comment per 3 lines.
- Length of functions ≤ 10 lines ideally. Else it could have more bugs.
- Avoid goto statements as it makes programs unstructured and difficult to understand.

Code Review

- Happens after compilation of the module and syntax errors are eliminated.
- Goal: reduce coding errors, improve quality.
- Two types: Code walkthrough and Code inspection.

Code Walkthrough

- ***Type****:* Informal code analysis technique.
- ***Goal****:* Find algorithmic & logical errors (not syntax).

- ***Process****:*
    1. Code is compiled (no syntax errors).
    2. Team members read code beforehand the walkthrough meeting, designs test cases.
    3. Hand-simulate execution (trace statements/functions).
    4. Discuss findings in a meeting with the coder.
- ***Guidelines****:*
    1. Team size: ***3–7 members***.
    2. Focus only on ***error discovery***, not fixing.
    3. Managers don’t attend, to keep it cooperative, not evaluative.

Code Inspection*:*

- ***Type****:* Formal code review.
- ***Goal****:* Detect common errors from oversight or bad practices.
- ***Method****:* Examine code directly not hand-simulate execution like in walkthrough.
- ***Checks****:*
    - Adherence to coding standards.
    - Common errors (e.g., uninitialized variables, jumps into loops, array out of bounds, non-terminating loops, mismatched parameters, operator precedence issues, comparison of equal floating point variables etc.).
- ***Practice****:* Good companies keep statistics of frequent errors → used as a checklist in inspections.

Clean Room Testing (IBM approach)

- Relies heavily on walkthroughs, inspection, and formal verification.
- Programmers ***cannot execute*** their own code (except syntax check).
- Focus on preventing defects via inspections, not finding them later.
- Inspired by semiconductor “clean rooms.”
- Characteristics:

- ***Formal specification*** – The software to be developed is formally specified. A state transition model which shows system responses to stimuli is used to express the specification.

  Incremental development – The software is partitioned into increments which are specified with customer input at an early stage.

- ***Structured programming*** – A limited number of control and data abstraction are used and the the program development is a stepwise refinement of the specification.
- ***Static verification*** – Static verification of the software is done rigorously. There is no unit or module testing for code components.
- ***Statistical system testing*** – The integrated software increment is tested statistically to determine its reliability.

Software Documentation

- Includes source code, manuals, SRS, design docs, test docs, installation guides.
- ***Benefits:***
  - Improves understandability, maintainability, and reduce the time and effort needed for maintenance.
  - Helps the users to effectively use the system.
  - Effectively helps in manpower turnover handling.
  - Progress of the development process can be easily tracked.
- ***Types:***
  - ***Internal documentation*** – Code comprehension features provided as part of the source code itself. It is provided through comments, headers, indentation, naming, code structure etc.
  - ***External documentation*** – Provided through user manuals, SRS, design docs, test docs, etc.