MENOUFIA UNIVERSITY
جامعة المنوفية
منارة المعرفة فى قلب الدلتا

Software Engineering

Dr. Mohammed Badawy

mbmbadawy@yahoo.com
mohamed.badawi@el-eng.menofia.edu.eg
Room: 417,   Ext. 7332

13 May 2016

Chapter 6

Software Coding

Contents

Introduction

- The goal of the **coding** activity is to implement the design in the best possible manner.
- The coding activity affects both testing and maintenance.
- The goal during coding should be to reduce the cost of later phases, even if it means that the cost of this phase has to increase.

Introduction

- There are many different criteria for judging a program:
  - readability,
  - size of the program,
  - execution time, and
  - required memory.
- Having readability and understandability as a clear objective can itself help in producing software that is more maintainable.
- So, <u>ease of understanding and modification are the basic goals of the programming activity</u>.

1. Programming Style

- The aim of a programming style is to optimize the code with desired results.
- Some general rules with respect to programming style.

1. Naming: you are required to name the module, processes, and variables and so on (avoid cryptic names, unknown acronyms, or names totally unrelated to the entity.)
2. Control Constructs: It is desirable that as many single-entry and single-exist constructs as possible be used.
3. Information Hiding: Information hiding should be supported where possible.

Programming Style

4. Gotos: Gotos should be used sparingly and in a disciplined manner. Only when the alternative using gotos is more complex then gotos should be used.
5. User-defined Type: Modern languages allow users to define types like the enumerated type. When such facilities are available, they should be exploited where applicable.
6. Nesting: If the condition-based nesting is too deep, the code becomes very complex.
7. Module Size: The module size should be uniform. Its size should not be too small or too big.
8. Program Layout: A good layout is one that helps to read the program faster and to understand it better.

Programming Style

9. Module Interface: A module with a complex interface (has multiple functions) should be carefully examined and avoided.
10. Robustness: A program is robust if it does something planned even for exceptional conditions(incorrect input, the incorrect value of some variable, and overflow.)
11. Side Effects: a module sometimes has side effects of modifying the program state beyond the modification of parameters listed in the module interface definition.
12. Switch case with default: If there is no default case in a "switch" statement, the behaviour can be unpredictable.

Programming Style

13. Empty if, while Statement: A condition is checked but nothing is done based on the check [if (x == 0) { }].
14. Read Return to be Checked: Often the return value from reads is not checked, assuming that the read returns the desired values.
15. Give Importance to Exceptions: Most programmers tend to give less attention to the possible exceptional cases and tend to work with the main flow of events, control, and data.

2. Internal Documentation

- Program documentation is one of two types:
  - external, addressing information about the program
  - internal, which has to be close to the program, program statement, and program block to explain it.
- Internal documentation of the program is done through the use of comments.
- All programming languages provide a means of writing comments in the program.
- The comment is a text written for the user, reader, or programmer to easily understand

Internal Documentation

- Comments should provide information on the following:
  - Functionality
  - Parameters and their role and usage
  - Attributers of inputs, i.e., assumptions values, range, max and min, etc.
  - Mention of global variables
- The main objective of internal documentation is to <u>provide on-line help to the user and programmer</u> to get a quick understanding of the program and the problem and to enable them to modify the program as fast as possible.

3. Monitoring and Control for Coding

- Code reviews are mainly designed to detect defects that are originated during the coding phase;
- After the successful completion of code, code inspection and reviews are held.
- Any defects found should be corrected
- Code defects can be divided into two groups: <u>logic and control defects</u> and <u>data operations and computations defects</u>.

Monitoring and Control for Coding

- Examples of logic and control defects are unreachable code, incorrect predicate, infinite loops, improper nesting of loops, unreferenced labels, etc.
- Examples of data operations and computations defects are incorrect access of array components, improper initialization, misuse of variables, etc.
- The following are some of the items that can be included in a checklist for a code review:

Monitoring and Control for Coding

- When are the divisors tested for zero applicable?
- Is important data tested for being validated?
- Is the loop termination condition right?
- Are indexes initialized?
- Are all variables used?
- Are the arrays indexed within bounds?
- Are all branch conditions right?
- Are the labels unreferenced?

4. Structured Programming

- Structured programming refers to a general methodology of writing good programs.
- A good program is one that has the following properties:

1. It should perform all the desired actions.
2. It should be reliable, i.e., perform the required actions within acceptable margins of error.
3. It should be clear, i.e., easy to read and understand.
4. It should be easy to modify.
5. It should be implemented within the specified schedule and budget.

Structured Programming

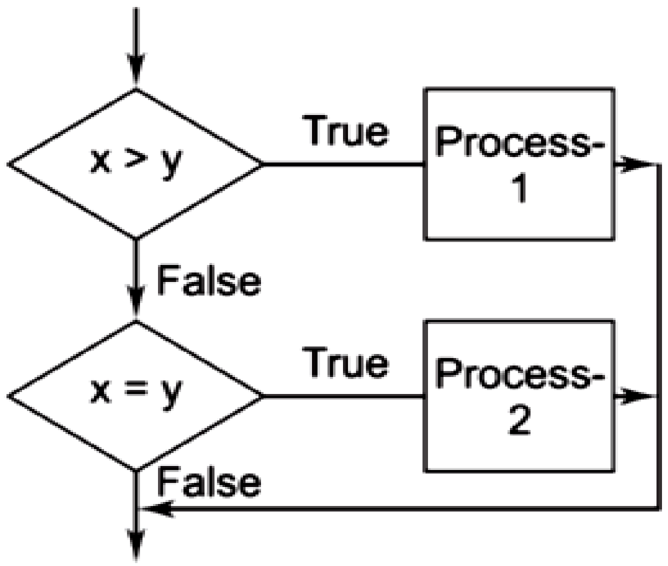
- A program is one of two types: Static structure or Dynamic structure.
- The static structure is a linear organization of statements of the program.
- The dynamic structure of the program is the sequence of statements executed during the execution of the program.
- Both static and dynamic structures are the sequence of statements. The only difference is that <u>the sequence of statements in a static structure is fixed</u>, whereas <u>in a dynamic structure it is not fixed.</u>
- That means the dynamic sequence of statements can change from execution to execution.

Objectives of Structured Programming

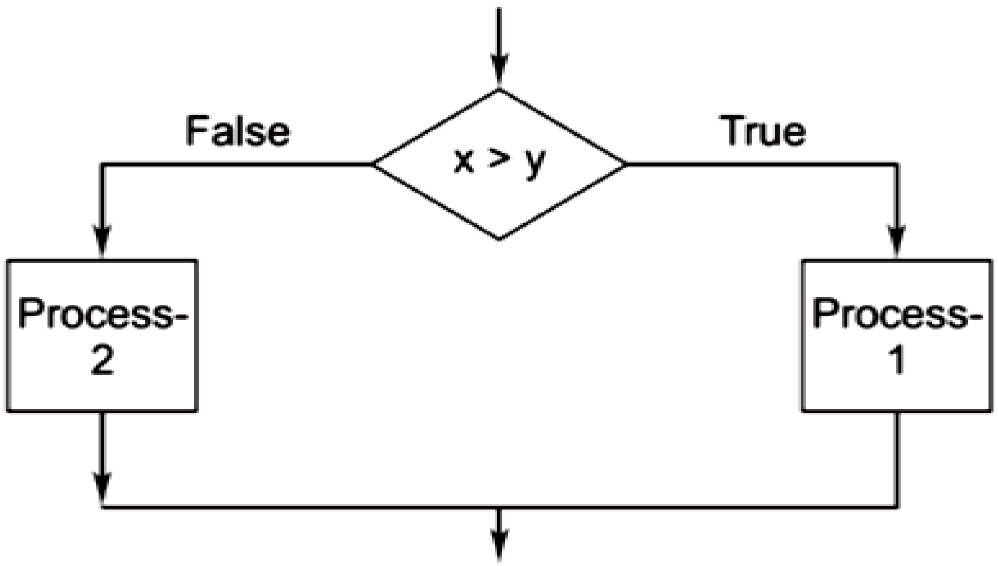
- The goal of structured programming is to ensure that the static structures and the dynamic structures are the same.
- All structured program design methods <u>are based upon the two fundamental principles</u> stepwise refinement and three structured control constructs.
- <u>The process of stepwise refinement</u> is such an approach that the stated program function is broken down into subsidiary functions in progressively increasing levels of detail until the lowest level functions are achievable in the programming language.

Principles of Structured Programming

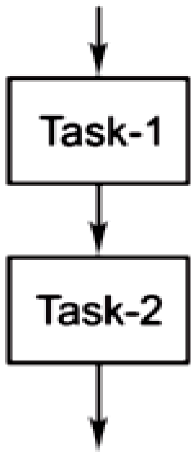
- The second principle of structured program design is that any program can be constructed using only three structured control constructs.
- The constructs selection, iterations, and sequence are shown in the next Figure (a, b, c, d).
- Any program independent of the technology platform can be written using these constructs, i.e., selection, repetition, sequence.
- These structures are the basis of structured programming.



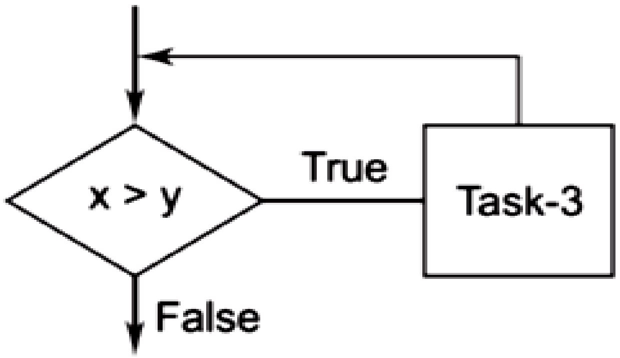
Selection Contract
(a)



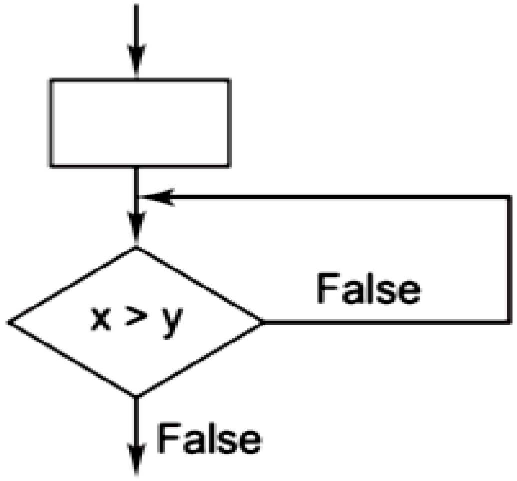
Selection Contract
(a)



Sequence
(b)



Do While Repetition
(c)



Repeat Until Repetition
(d)

Key Features of Structured Programming

- The key feature of a structured program is that it has a single-entry and a single-exit.
- The most commonly used single entry and single exit statements are:

```
Selection:      if customer type is X
                then price is $100
                else price is $90

Iteration:      while customer type is X do use
                     price formula P1.
                repeat P1 until type is X.

Sequencing:     Task 1, Task 2, Task 3.
```

Advantages of Structured Programming

- It is very convenient to put logic systematically into the program.
- Due to the ease of handling complex logic, the user, reader, and programmer understand the program easily.
- It is easy to verify, conduct reviews, and test the structured programs in an orderly manner. If errors are found, they are easy to locate and correct.

5. Verification

- Once a programmer has written the code for a module, it has to be verified before it is used by others.
- Though testing is the most common method of verification, there are other effective techniques also.
- Verification is not mean proving correctness of programs, which for our purposes is only one method for program verification.
- The techniques that are now widely used are
  - inspections (including code reading),
  - unit testing, and
  - program checking.

Code Inspections

- **Inspection**, which can be applied to any document, has been widely used for detecting defects.
- Code inspections are usually held after code has been successfully compiled and other forms of static tools have been applied.
- The documentation to be distributed to the inspection team members includes the code to be reviewed and the design document.
- The team for code inspection should include the programmer, the designer, and the tester.

Code Inspections

- **The aim** of code inspections is to detect defects in code in addition to quality issues which code inspections usually look for, like efficiency, compliance to coding standards, etc.
- Often the type of defects the code inspection should focus on is contained in a checklist that is provided to the inspectors.
- Some of the items that can be included in a checklist for code reviews are:

- Do data definitions exploit the typing capabilities of the language?
- Do all the pointers point to some object? (Are there any "dangling pointers"?)
- Are the pointers set to NULL where needed?
- Are pointers being checked for NULL when being used?
- Are all the array indexes within bound?
- Are indexes properly initialized?
- Are the entire branch conditions correct (not too weak, not too strong)?
- Will a loop always terminate (no infinite loops)?
- Is the loop termination condition correct?
- Is the number of loop executions "off by one"?
- Where applicable, are the divisors tested for zero?
- Are imported data tested for validity?
- Do actual and formal interface parameters match?
- Are all variables used? Are all output variables assigned?
- Can statements placed in the loop be placed outside the loop?
- Are the labels unreferenced?
- Will the requirements of execution time be met?
- Are the local coding standards met?

Checking Forms

- There are many techniques for verification that are not testing-based, but directly check the programs through the aid of analysis tools.
- Three forms of checking are becoming popular—**model checking, dynamic analysis, and static analysis**.
- In **model checking**, an abstract model is constructed for the program being verified.
- The model captures those aspects that affect the properties that are to be checked.
- The desired properties are also specified and a model checker checks whether the model satisfies the stated properties.

Checking Forms

- In **dynamic analysis**, the program is instrumented and then executed with some data.
- The value of variables, branches taken, etc. are recorded during the execution.
- Using the data recorded, it is evaluated if the program behaviour is consistent with some of the dynamic properties.

Static Analysis

- The most widely used program checking technique is **static analysis**.
- Static analysis is analyzing of programs by methodically analyzing the program text.
- Static analysis is usually performed mechanically by the aid of software tools.
- The aim of static analysis is to detect errors or potential errors in the code and to generate information that can be useful in debugging.
- Many compilers perform some limited static analysis.
- Static analysis tools focus on detecting errors.

Static Analysis

- **Two approaches** are possible.
- **The first** is to detect patterns in code that are "unusual" or "undesirable" and which are likely to represent defects.
- The other is to directly look for defects in the code.
- In either case, a static analyzer, as it is trying to identify defects without running the code but only by analyzing the code, sometimes identifies situations as errors which are not actually errors, and sometimes fails to identify some errors.

Unit Testing

- Unit testing is another approach for verifying the code that a programmer is written.
- Unit testing is like regular testing where programs are executed with some test cases except that the focus is on testing smaller programs or modules called units.
- A unit may be a function, a small collection or functions, a class, or a small collection of classes.
- Most often, it is the unit a programmer is writing code for, and hence <u>unit testing is most often done by a programmer</u> to test the code that he or she has written.

Unit Testing

- Selection of test cases is a key issue in any form of testing.
- For now, it suffices that during unit testing the tester will execute the unit for a variety of test cases and study the actual behaviour of the units being tested for these test cases.
- Based on the behaviour, the tester decides whether the unit is working correctly or not.
- If the behaviour is not as expected for some test case, then the programmer finds the defect in the program (debugging)

Unit Testing

- After removing the defect, the programmer will execute the test case that caused the unit to fail again to ensure that the fixing has indeed made the unit behave correctly.
- For a functional unit, unit testing will involve testing the function with different test data as input.
- Typically, the test data will include some data representing the <u>normal case</u>, that is, the one that is most likely to occur.
- In addition, test data will be selected for <u>special cases</u> which must be dealt with by the program and which might result in special or exceptional result.

Unit Testing

- As the unit being tested is not a complete system but just a part, it is not executable by itself.
- Due to this, unit testing often requires <u>drivers or stubs</u> to be written.
- Drivers play the role of the "calling" module and are often responsible for getting the test data, executing the unit with the test data, and then reporting the result.
- Stubs are essentially "dummy" modules that are used in place of the actual module to facilitate unit testing.

Unit Testing

- So, if a module M uses services from another module M' that has not yet been developed, then for unit testing M, some stub for M' will have to be written so M can invoke the services in some manner on M' so that unit testing can proceed.
- If incremental coding is practiced, then unit testing needs to be performed every time the programmer adds some code.
- For incremental testing it is desirable that the programmer develops this unit testing script and keeps enhancing it with additional test cases as the code evolves.
- In object-oriented programs, the unit to be tested is usually an object of a class.

Fourth-Generation Techniques

- Fourth-Generation Technique (4GT) includes the application of tools and techniques for expeditious development of software solutions.
- Besides expeditious development, fourth-generation techniques help to control effort, resources, and cost of development.
- Commonly used fourth-generation techniques in development models are:

Fourth-Generation Techniques

- Report Generation
- Database Query Languages
- Data Manipulation
- Screen Definition and Interaction
- Code Generation
- Connectivity, such as ODBC, JDBC, and Interfacing
- Web-engineering Tools
- Graphics, Spreadsheet Generation Capability
- CASE Tools

Fourth-Generation Techniques

- The 4GT paradigm for software engineering focuses on the ability to specify software using specialized language forms or a generic notation that describes the problem to be solved in terms that the customer can understand.
- 4GT has a number of query languages, report writers, generators, program and application generators, and high-end object-oriented languages.
- Extensive use of 4GTs saves a lot of software-specific coding and allows rapid prototype development.