Coding

Coding

- Goal is to implement the design in best possible manner
- Coding affects testing and maintenance
- As testing and maintenance costs are high, aim of coding activity should be to write code that reduces them
- Hence, goal should not be to reduce coding cost, but testing and maint cost, i.e. make the job of tester and maintainer easier

Coding...

- Code is read a lot more
  - Coders themselves read the code many times for debugging, extending etc
  - Maintainers spend a lot of effort reading and understanding code
  - Other developers read code when they add to existing code
- Hence, code should be written so it is easy to understand and read, not easy to write!

Coding...

- Having clear goal for coding will help achieve them
- Weinberg experiment showed that coders achieve the goal they set
  - Diff coders were given the same problem
  - But different objectives were given to diff programmers – minimize effort, min size, min memory, maximize clarity, max output clarity
  - Final programs for diff programmers generally satisfied the criteria given to them

Weinberg experiment..

	Resulting Rank (1=best)				
	o1	o2	o3	o4	o5
Minimize Effort (o1)	1	4	4	5	3
Minimize prog size (o2)	2-3	1	2	3	5
Minimize memory (o3)	5	2	1	4	4
Maximize code clarity (o4)	4	3	3	2	2
Maximize output clarity (o5)	2-3	5	5	1	1

Programming Principles

- The main goal of the programmer is write simple and easy to read programs with few bugs in it
- Of course, the programmer has to develop it quickly to keep productivity high
- There are various programming principles that can help write code that is easier to understand (and test...)

Structured Programming

- Structured programming started in the 70s, primarily against indiscriminate use of control constructs like gotos
- Goal was to simplify program structure so it is easier to argue about programs
- Is now well established and followed

Structured Programming...

- A program has a *static* structure which is the ordering of statements in the code – and this is a linear ordering
- A program also has *dynamic* structure – order in which statements are executed
- Correctness of a program refers to the dynamic structure

Structured Programming...

- To show a program as correct, we must show that its dynamic behavior is as expected
- But we must argue about this from the code of the program, i.e. the static structure
- This will become easier if the dynamic and static structures are similar
- Closer correspondence will make it easier to understand dynamic behavior from static structure
- This is the idea behind structured programming

Structured Programming...

- Meaningful programs cannot be written as sequence of simple statements
- To achieve the objectives, structured constructs are to be used
- These are single-entry-single-exit constructs
- With these, execution of the statements can be in the order they appear in code
- The dynamic and static order becomes same

Structured Programming...

- SP was promulgated to help formal verification of programs
- Without linear flow, composition is hard and verification difficult
- But, SP also helps simplify the control flow of programs, making them easier to understand and argue about
- SP is an accepted and standard practice today – modern languages support it well

Information Hiding

- Software solutions always contain data structures (DS) that hold information
- Programs work on these DS to perform the functions they want
- In general only some operations are performed on the information, i.e. the data is manipulated in a few ways only
- E.g. on a bank's ledger, only debit, credit, check current balance etc are done

Information Hiding...

- Information hiding – the information should be hidden; only operations on it should be exposed
- i.e. data structures are hidden behind the access functions, which can be used by programs
- Info hiding reduces coupling
- This practice is a key foundation of OO and components, and is also widely used today

Some Programming Practices

- Control constructs: Use only a few structured constructs (rather than using a large no of constructs)
- Goto: Use them sparingly, and only when the alternatives are worse
- Info hiding: Use info hiding
- Use-defined types: use these to make the programs easier to read

Some Programming Practices..

- Nesting: Avoid heavy nesting of if-then-else
- Module size: Should not be too large
- Module interface: make it simple
- Robustness: Handle exceptional situations
- Side effects: Avoid them, document carefully

Some Programming Practices..

- Empty catch block (exception handling): always have some default action rather than empty
- Empty if, while: bad practice
- Read return: should be checked for robustness, ensure read action is successful, or exception handling ready

Coding Standards

- Programmers spend more time reading code than writing code
- They read their own code as well as other programmers' code
- Readability is enhanced if some coding conventions are followed by all
- Coding standards provide these guidelines for programmers
- Generally are regarding naming, file organization, statements/declarations, ...
- Some conventions discussed here (for reference only, as there is no universal standard).

Coding Standards...

- Naming conventions
  - Type names should be nouns and start with uppercase (Day, DateOfBirth,...)
  - Var names should be nouns in lowercase; vars with large scope should have long names; loop iterators should be i, j, k...
  - Const names should be all caps
  - Method names should be verbs starting with lower case (eg getValue())
  - Prefix *is* should be used for boolean methods

Coding Standards...

- Statements
  - Vars should be initialized where declared in the smallest possible scope
  - Declare related vars together; unrelated vars should be declared separately
  - Class vars should never be declared public
  - Loop vars should be initialized just before the loop (this is followed in C++)
  - Avoid using break and continue in loops
  - Avoid executable statements in conditionals (in C/C++)
  - Avoid using the do... while construct

Coding Standards...

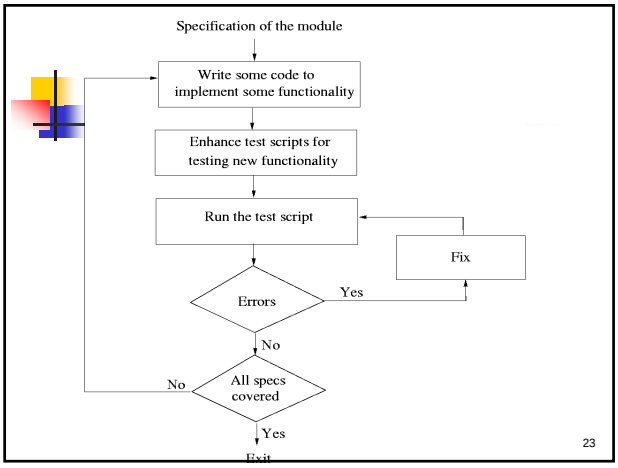
- Commenting and layout
  - Single line comments for a block should be aligned with the code block
  - There should be comments for all major vars explaining what they represent
  - A comment block should start with a line with just /* and end with a line with */
  - Trailing comments after statements should be short and on the same line

Coding Process

- Coding starts when specs for modules from design is available
- Usually modules are assigned to programmers for coding
- In top-down development, top level modules are developed first; while in bottom-up development, lower levels modules first.
- For coding, developers use different processes; we discuss some here

Incremental Coding Process

- Basic process: Write code for the module, unit test it, fix the bugs
- It is better to do this incrementally – write code for part of functionality, then test it and fix it, then proceed

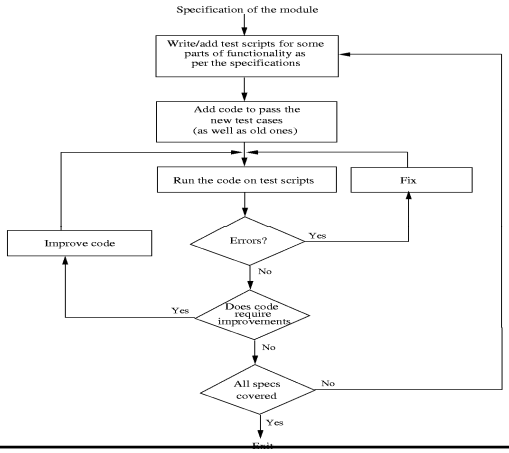


Test Driven Development

- This coding process changes the order of activities in coding
- In TDD, programmer first writes the test scripts and then writes the code to pass the test cases in the script
- This is done incrementally
- Is a relatively new approach, and is a part of the extreme programming (XP)

TDD...

- In TDD, you write just enough code to pass the test
- i.e. code is always in sync with the tests and gets tested by the test cases
  - Not true in code first approach, as test cases may only test part of functionality
- Responsibility to ensure that all functionality is there is on test case design, not coding
- Help ensure that all code is testable



Pair Programming

- Also a coding process that has been proposed as key practice in XP
- Code is written by pair of programmers rather than individuals
  - The pair together design algorithms, data structures, strategies, etc.
  - One person types the code, the other actively reviews what is being typed
  - Errors are pointed out and together solutions are formulated
  - Roles are reversed periodically

Pair Programming...

- PP has continuous code review, and reviews are known to be effective
- Better designs of algorithms/DS/logic/...
- Special conditions are likely to be dealt with better and not forgotten
- It may, however, result in loss of productivity
- Ownership and accountability issues are also there
- Effectiveness is not yet fully known

Source Code Control and Built

- Source code control is an essential step programmers have to do
- Generally tools like CVS are used
- A tool consists of repository, which is a controlled directory structure
- The repository is the official source for all the code files
- System build is done from the files in the repository only

Verification

- Code has to be verified before it can be used by others
- Here we discuss only verification of code written by a programmer (system verification is discussed in testing)
- There are many different techniques; key ones – unit testing, inspection, and program checking
- Program checking can also be used at the system level

Code Inspections

- The inspection process can be applied to code with great effectiveness
- Inspections held when code has compiled and a few tests passed
- Usually static tools are also applied before inspections
- Inspection team focuses on finding defects and bugs in code
- Checklists are generally used to focus the attention on defects

Code Inspections...

- Some items in a checklist
  - Do all pointers point to something
  - Are all vars and pointers initialized
  - Are all array indexes within bounds
  - Will all loops always terminate
  - Any security flaws
  - Is input data being checked
  - Obvious inefficiencies

Code inspections...

- Are very effective and are widely used in industry (many require all critical code segments to be inspected)
- Is also expensive; for non critical code one person inspection may be used

Unit Testing

- Is testing, except the focus is the module a programmer has written
- Most often UT is done by the programmer himself
- UT will require test cases for the module – will discuss in testing
- UT also requires drivers to be written to actually execute the module with test cases
- Besides the driver and test cases, tester needs to know the correct outcome as well

Unit Testing...

- If incremental coding is being done, then complete UT needs to be automated
- There are tools available to help
  - They provide the drivers
  - Test cases are programmed, with outcomes being checked in them
  - I.e. UT is a script that returns pass/fail
  - e.g. JUnit for Java classes

Static Analysis

- These are tools to analyze program sources and check for problems
- Static analyzer cannot find all bugs and often cannot be sure of the bugs it finds as it is not executing the code
- So there is noise in their output
- Many different tools available that use different techniques
- They are effective in finding bugs like memory leak, dead code, dangling pointers,..

Common Coding Errors

- Goal of programmer is to write quality code with few bugs in it
- Much of effort in developing software goes in identifying and removing bugs
- Common bugs which occur during coding directly or indirectly manifest themselves to a larger damage to the running program
- List of common coding errors can help a programmer avoid them

Memory Leaks

- A memory leak is a situation, where the memory is allocated to the program which is not freed subsequently
- Occurs frequently in the languages which do not have automatic garbage collection
- Can cause increasing usage of memory which at some point of time can lead to exceptional halt of the program
- e.g char* foo(int s)

```
{
    Char *output;
    if (s>0)
        Output=(char*) malloc (size)  //request memory
    if (s==1)
        return NULL /* if s==1 then mem leaked */
    return (output);
}
```


Freeing an Already Freed Resource

- Programmer tries to free the already freed resource
- May be serious, if some malloc between the two free statements as the freed location may get allocated to a new variable, and subsequent free will deallocate the new variable.

e.g

```
main()
{
    char *str;
    str=(char *)malloc(10);
    If(global==0)
        free(str);
    free(str); /* str is already freed */
}
```


NULL Dereferencing

- Occurs when we access a location that points to NULL
- Improper initialization and missing the initialization in different paths leads to the NULL reference error
- Can also be caused by aliases- for e.g two variables refer to same object , and one is freed and an attempt is made to dereference the second

Lack of Unique Addresses

- Aliasing creates many problems, among them is violation of unique addresses when we expect different addresses.
- For e.g in the string concatenation function, we expect source and destination addresses to be different.
- strcat (src , destn);

  /* In above function , if src is aliased to destn , then we may get a runtime error */

Synchronization Errors

- Possible when there are multiple threads which are accessing some common resources, in a parallel program
- three categories of synchronization errors: deadlocks, race condition, live lock
- Deadlock example:

```
Thread 1:
synchronized (A){
synchronized (B){ }
}
Thread 2:
synchronized (B){
synchronized (C){ }
}
Thread 3:
synchronized (C){
synchronized (A){ }
}
```

Synchronization Errors...

- Race condition occurs when two threads access same resource and result depends on the order of the execution

E.g
```
class reservation
{
int seats_remaining ;
public int reserve (int x)
{
if (x <= seats_remaining ) {
seats_remaining -=x;
return 1;
}
return 0;
}
}
```

- Inconsistent synchronization represents the situation where there is a mix of locked and unlocked accesses to some shared variables, particularly if the access involves updates

Buffer overflow

- It is a security vulnerability, can be exploited by executing arbitrary code by a malicious user

```
char s1 [1024];
void mygets ( char *str ){
int ch;
while (ch= getchar () != '\n' && ch != '\0 ')
*( str ++)= ch;
*str = '\0 ';
}
main (){
char s2 [4];
mygets (s2 );
}
```

- If we have malicious code in s1 and if return address of mygets() is replaced by this address by overflowing the buffer s2, then we can have the code in s1 executed

Other common type of errors

- Array index out of bounds, care needs to be taken to see that the array index values are not negative and do not exceed their bounds
- Enumerated data types can lead to overflow and underflow errors, care should be taken while assuming the values of such types

```
typedef enum {A, B,C, D} grade ;
void foo( grade x){
int l,m;
l= GLOBAL_ARRAY [x -1];
/* Underflow when A */
m= GLOBAL_ARRAY [x +1];
/* Overflow when D and size of array is 4 */
}
```


Other common type of errors..

- Arithmetic exceptions, include errors like divide by zero and floating point exceptions
- Off by one errors like starting variable at 1 instead of starting at 0 or vice versa, writing <= N instead of < N or vice versa etc.
- String handling errors like failure of string handling functions e.g strcpy, sprintf, gets etc.
- Illegal use of & instead of &&, arises if non short circuit logic (like & or |) is used instead of short circuit logic (&& or ||)

E.g
```
if(object!=null & object.getTitle()!=null)
/* Here second operation can cause a null dereference */
```


Summary

- Goal of coding is to convert a design into easy to read code with few bugs
- Good programming practices like structured programming, information hiding, etc can help
- There are many methods to verify the code of a module – unit testing and inspections are most commonly used