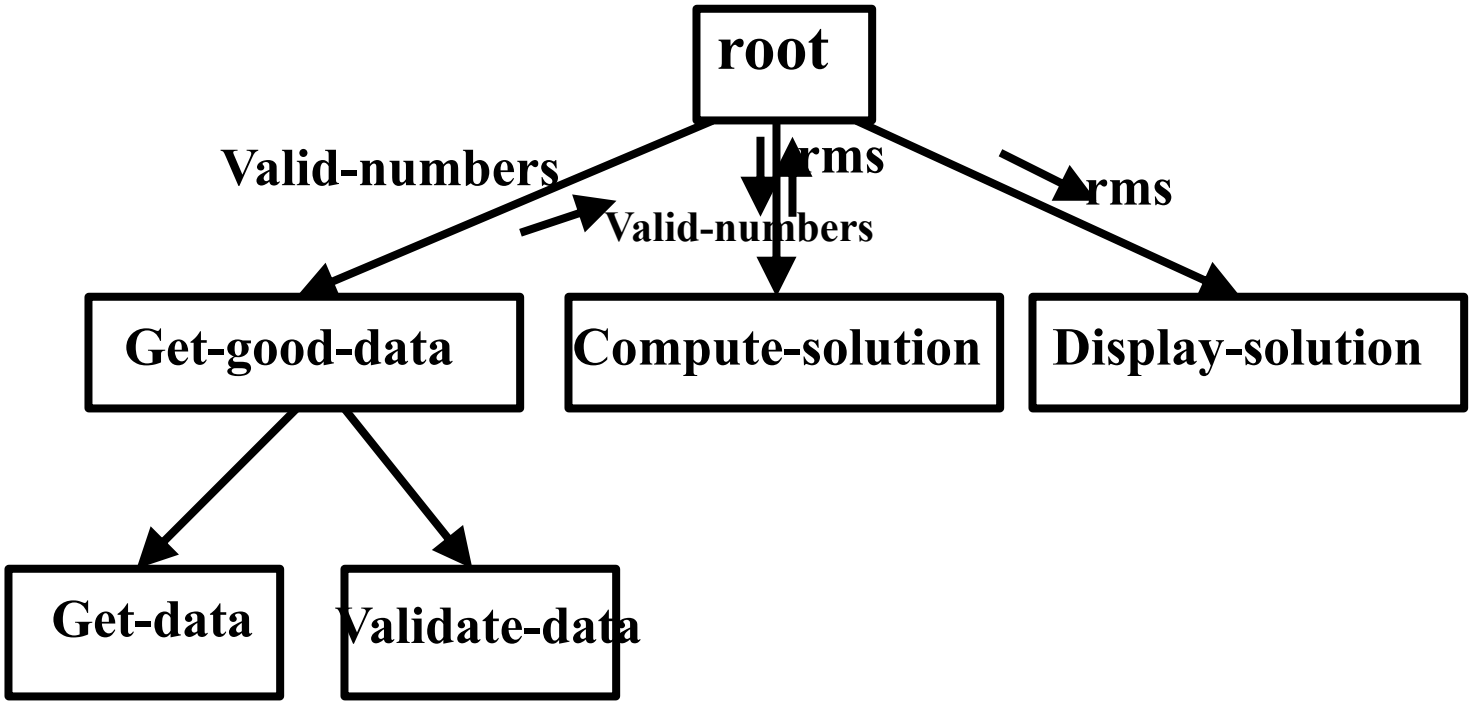
Coding

Coding Phase

- Coding is undertaken once design phase is complete.
- During coding phase:
  - every module identified in the design document is coded and unit tested.
- Unit testing (aka module testing):
  - testing of different modules (aka units) of a system in isolation.

Example Structured Design



Unit Testing

- Many beginners ask:
  - Why test each module in isolation first?
  - then integrate the modules and again test the set of modules?
  - why not just test the integrated set of modules once thoroughly?

Unit Testing

- It is a good idea to test modules in isolation before they are integrated:
  - it makes debugging easier.

Unit Testing

- If an error is detected when several modules are being tested together,
  - it would be difficult to determine which module has the error.
- Another reason:
  - the modules with which this module needs to interface may not be ready.

Integration Testing

- After all modules of a system have been coded and unit tested:
  - integration of modules is done
    - according to an integration plan.

Integration Testing

- The full product takes shape:
  - only after all the modules have been integrated.
- Modules are integrated together according to an integration plan:
  - involves integration of the modules through a number of steps.

Integration Testing

- During each integration step,
  - a number of modules are added to the partially integrated system
    - and the system is tested.
- Once all modules have been integrated and tested,
  - system testing can start.

System Testing

- During system testing:
  - the fully integrated system is tested against the requirements recorded in the SRS document.

Coding

- The input to the coding phase is the design document.
- During coding phase:
  - modules identified in the design document are coded according to the module specifications.

Coding

- At the end of the design phase we have:
  - module structure (e.g. structure chart) of the system
  - module specifications:
    - data structures and algorithms for each module.
- Objective of coding phase:
  - transform design into code
  - unit test the code.

Coding Standards

- Good software development organizations require their programmers to:
  - adhere to some standard style of coding
  - called coding standards.

Coding Standards

- Many software development organizations:
  - formulate their own coding standards that suits them most,
  - require their engineers to follow these standards rigorously.

Coding Standards

- Advantage of adhering to a standard style of coding:
  - it gives a uniform appearance to the codes written by different engineers,
  - it enhances code understanding,
  - encourages good programming practices.

Coding Standards

- A coding standard
  - sets out standard ways of doing several things:
    - the way variables are named,
    - code is laid out,
    - maximum number of source lines allowed per function, etc.

Coding guidelines

- Provide general suggestions regarding coding style to be followed:
  - leave actual implementation of the guidelines:
    - to the discretion of the individual engineers.

Code inspection and code walk throughs

- After a module has been coded,
  - code inspection and code walk through are carried out
  - ensures that coding standards are followed
  - helps detect as many errors as possible before testing.

Code inspection and code walk throughs

- Detect as many errors as possible during inspection and walkthrough:
  - detected errors require less effort for correction
    - much higher effort needed if errors were to be detected during integration or system testing.

Coding Standards and Guidelines

- Good organizations usually develop their own coding standards and guidelines:
  - depending on what best suits their organization.
- We will discuss some representative coding standards and guidelines.

Representative Coding Standards

- Rules for limiting the use of globals:
  - what types of data can be declared global and what can not.
- Naming conventions for
  - global variables,
  - local variables, and
  - constant identifiers.

Representative Coding Standards

- Contents of headers for different modules:
  - The headers of different modules should be standard for an organization.
  - The exact format for header information is usually specified.

Representative Coding Standards

- Header data:
  - Name of the module,
  - date on which the module was created,
  - author's name,
  - modification history,
  - synopsis of the module,
  - different functions supported, along with their input/output parameters,
  - global variables accessed/modified by the module.

Representative Coding Standards

- Error return conventions and exception handling mechanisms.
  - the way error and exception conditions are handled should be standard within an organization.
  - For example, when different functions encounter error conditions
    - should either return a 0 or 1 consistently.

Representative Coding Guidelines

- Do not use too clever and difficult to understand coding style.
  - Code should be easy to understand.
- Many inexperienced engineers actually take pride:
  - in writing cryptic and incomprehensible code.

Representative Coding Guidelines

- Clever coding can obscure meaning of the code:
  - hampers understanding.
  - makes later maintenance difficult.
- Avoid obscure side effects.

Representative Coding Guidelines

- The side effects of a function call include:
  - modification of parameters passed by reference,
  - modification of global variables,
  - I/O operations.
- An obscure side effect:
  - one that is not obvious from a casual examination of the code.

Representative Coding Guidelines

- Obscure side effects make it difficult to understand a piece of code.
- For example,
  - if a global variable is changed obscurely in a called module,
  - it becomes difficult for anybody trying to understand the code.

Representative Coding Guidelines

- Do not use an identifier (variable name) for multiple purposes.
  - Programmers often use the same identifier for multiple purposes.
  - For example, some programmers use a temporary loop variable
    - also for storing the final result.

Example use of a variable for multiple purposes

```
for(i=1;i<100;i++)
        {.....}
     i=2*p*q;
 return(i);
```

Use of a variable for multiple purposes

- There are several things wrong with this approach:
  - hence should be avoided.
- Each variable should be given a name indicating its purpose:
  - This is not possible if an identifier is used for multiple purposes.

Use of a variable for multiple purposes

- Leads to confusion and annoyance
  - for anybody trying to understand the code.
  - Also makes future maintenance difficult.

Representative Coding Guidelines

- Code should be well-documented.
- Rules of thumb:
  - on the average there must be at least one comment line
    - for every three source lines.
  - The length of any function should not exceed 10 source lines.

Representative Coding Guidelines

- Lengthy functions:
  - usually very difficult to understand
  - probably do too many different things.

Representative Coding Guidelines

- Do not use goto statements.
- Use of goto statements:
  - make a program unstructured
  - make it very difficult to understand.

Code Walk Through

- An informal code analysis technique.
  - undertaken after the coding of a module is complete.
- A few members of the development team select some test cases:
  - simulate execution of the code by hand using these test cases.

Code Walk Through

- Even though an informal technique:
  - several guidelines have evolved over the years
  - making this naive but useful analysis technique more effective.
  - These guidelines are based on
    - personal experience, common sense, and several subjective factors.

Code Walk Through

- The guidelines should be considered as examples:
  - rather than accepted as rules to be applied dogmatically.
- The team performing code walk through should not be either too big or too small.
  - Ideally, it should consist of between three to seven members.

Code Walk Through

- Discussion should focus on discovery of errors:
  - and not on how to fix the discovered errors.
- To foster cooperation:
  - avoid the feeling among engineers that they are being evaluated in the code walk through meeting,
  - managers should not attend the walk through meetings.

Code Inspection

- In contrast to code walk throughs,
  - code inspection aims mainly at discovery of commonly made errors.
- During code inspection:
  - the code is examined for the presence of certain kinds of errors,
  - in contrast to the hand simulation of code execution done in code walk throughs.

Code Inspection

- For instance, consider:
  - classical error of writing a procedure that modifies a formal parameter
  - while the calling routine calls the procedure with a constant actual parameter.
- It is more likely that such an error will be discovered:
  - by looking for this kind of mistakes in the code,
  - rather than by simply hand simulating execution of the procedure.

Code Inspection

- Good software development companies:
  - collect statistics of errors committed by their engineers
  - identify the types of errors most frequently committed.
- A list of common errors:
  - can be used during code inspection to look out for possible errors.

Commonly made errors

- Use of uninitialized variables.
- Nonterminating loops.
- Array indices out of bounds.
- Incompatible assignments.
- Improper storage allocation and deallocation.
- Actual and formal parameter mismatch in procedure calls.
- Jumps into loops.

Code Inspection

- Use of incorrect logical operators
  - or incorrect precedence among operators.
- Improper modification of loop variables.
- Comparison of equality of floating point values, etc.
- Also during code inspection,
  - adherence to coding standards is checked.

Software Documentation

- When developing a software product we develop various kinds of documents :
  - In addition to executable files and the source code:
  - users' manual,
  - software requirements specification (SRS) document,
  - design document, test document,
  - installation manual, etc.
- All these documents are a vital part of good software development practice.

Software Documentation

- Good documents enhance understandability and maintainability of a software product.
- Different types of software documents can be classified into:
  - internal documentation,
  - external documentation (supporting documents).

Internal Documentation

- Internal documentation:
  - documentation provided in the source code itself.
- External documentation:
  - documentation other than those present in the source code.

Internal Documentation

- Internal documentation provided through:
  - use of meaningful variable names,
  - code indentation,
  - code structuring,
  - use of enumerated types and constant identifiers,
  - use of user-defined data types, etc.
  - module headers and comments

Internal Documentation

- Good software development organizations:
  - ensure good internal documentation through coding standards and coding guidelines.
- Example of unhelpful documentation:
  - a = 10; /* a made 10 */

Internal Documentation

- Careful experimentation suggests:
  - meaningful variable names is the most useful internal documentation.

External Documentation

- Users' manual,
- Software requirements specification document,
- Design document,
- Test documents,
- Installation instructions, etc.

External Documentation

- A systematic software development style ensures:
  - all external documents are produced in an orderly fashion.
- An important feature of good documentation is consistency.

External Documentation

- Unless all documents are consistent with each other,
  - a lot of confusion is created for somebody trying to understand the product.
- All the documents for a product should be up-to-date:
  - Even a few out-of-date documents can create severe confusion.

Comprehensibility of Documents

- Readability is an important attribute of textual documents.
- Readability determines understandability
  - hence determines maintainability.
- A well-known readability measure of text documents:
  - Gunning's Fog Index.

Gunning's Fog Index

$$F = 0.4 \times \frac{\text{Number of Words}}{\text{Number of Sentences}} + \text{Percentage of words of 3 or more syllables}$$

- F corresponds to the number of years of schooling to easily understand the document.

Gunning's Fog Index

- A document is easy to understand if:
  - all sentences are small
    - use only 4 to 5 words each
  - small number of characters used per word:
    - normally not exceeding five or six characters.

Summary

- Coding standards:
  - enforce good coding practice
- Coding guidelines:
  - suggestions to programmers
  - exact implementation depends on discretion of the programmers.

Summary

- It is necessary to adequately document a software product:
  - Helps in understanding the product
  - Helps in maintenance

Summary

- Documentation
  - Internal
  - External
- Internal documentation
  - provided in the source code itself.
- Comprehensibility of text documents:
  - mesured using Gunning's Fog index.