Implementation overview

In this chapter we will:

- Become aware of some of the potential dangers of C as an implementation language
- Discover good programming practice in C
- Understand how to organize programming teams
- Appreciate that there are three fundamental ways of testing code, including code inspections
- Learn about CASE tools for the implementation phase

Drawbacks of C

- C is not a strongly typed language
- C compilers do not necessarily check whether the right number of parameters is passed

The way out:

- Manual checking of interfaces (both developers and SQA)
- The use of function declarations
- The use of *lint*

More drawbacks

- Lack of check of array boundaries while referencing array elements

- Memory access via pointers may cause problems:
  - It is possible to reference an arbitrary computer memory location. An attempt to access (read or write) arbitrary locations may lead to problems. **Question:** What kind of problems?
  - Memory blocks with no references to them cause *memory leaks*

On C programming Style

Avoid unreadable code.

Question: What is the meaning of

$$*value++$$

$(*value)++$ or $*(value++)$?

- Every programmer must write code in such a way that future maintenance programmers will have no difficulty in understanding the code
- Every construct that would not be immediately understood by even a below-average programmer must be rewritten
- If SQA cannot understand the function easily by simply reading through it, then the original designers have not done their job properly

Good programming practice

Variable names must be *meaningful* and *consistent*.

- *Meaningful* from the viewpoint of the maintenance programmer

- *Inconsistent* names are
  - $\underline{\text{reg}}$_address
  - $\underline{\text{region}}$_record
  - record_ptr_$\underline{\text{rgn}}$
  - name_of_$\underline{\text{regn}}$

  Problem 1 Do the underlined parts stand for 'region'?
  Problem 2 'region' should be placed either at the beginning or at the end of the variable name.

Variable names should be based on *one language*.

Some simple variable naming ideas

- Variable name should contain type information
- Function names should reflect their behavior w.r.t. their arguments: for example

  $$conc\_d(l1, l2)$$

  consumes its arguments (frees the memory of $l1$ and $l2$), whereas

  $$conc(l1, l2)$$

  does not do so.

On the use of comments

Self-documenting code may exist but it does not guarantee to be better than well-documented code. Well-documented code distinguishes between following kinds of comments

- Prologue comments
- Function comments
- Inline comments

Prologue comments

Prologue comments

- Brief description of what the compilation unit does
- Name of programmer(s)
- Date compilation unit was coded
- Date compilation unit was approved
- Where to find test data
- List of modifications
- Known faults, if any.

Function comments

- Function name, brief description what the function does
- Name of programmer(s), coding date
- List of parameters and their uses
- Files accessed by this function, if any
- Files changed by this function, if any
- Function input-output, if any
- Error-handling capabilities.

When to use inline comments:

- To comment non-obvious code
- To comment subtle aspect of the language.

Final remarks of comments

How to link detailed design to implementation?

Each pseudo-code statement is placed between comments and followed by C-code that implements it.

The purpose of SQA.

SQA must check the consistency of comments and the actual code.

Comments and maintenance phase

Comments must be maintained as well.

Code layout

- One statement per line
- Proper indentation
- Use blank lines between functions, and in large functions between big blocks
- If-statements should not be nested to a level deeper than 3.
- The right use of GOTOs:
  - WRONG: GOTO implements a loop
  - RIGHT: Forward-GOTO implements error handling.

Some companies use these or similar guidelines as *coding standards.*

Team organization

How to divide work between teams and within a team between developers such that the product is delivered on time and at given cost?

If there is one person-year coding involved, but the deadline is in 3 months, then why don't assign 4 programmers to accomplish the task?

Why this does not always work?

> If one champion crosses the English channel in 8h, how long will it take for 8 champions?

So the question is how to divide a big chunk of work between teams or between programmers?

Question: What does sharing cause?

Brooks law:

> Adding additional personell to a late project makes it even later.

Flat team structure means that every team member has to communicate with each other team member.

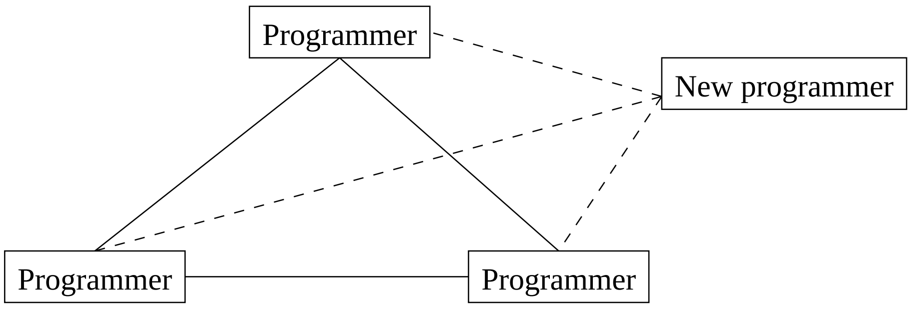

Figure 9.5 Communication paths.

Chief programmer team is hierarchical team organization with less communication between programmers.

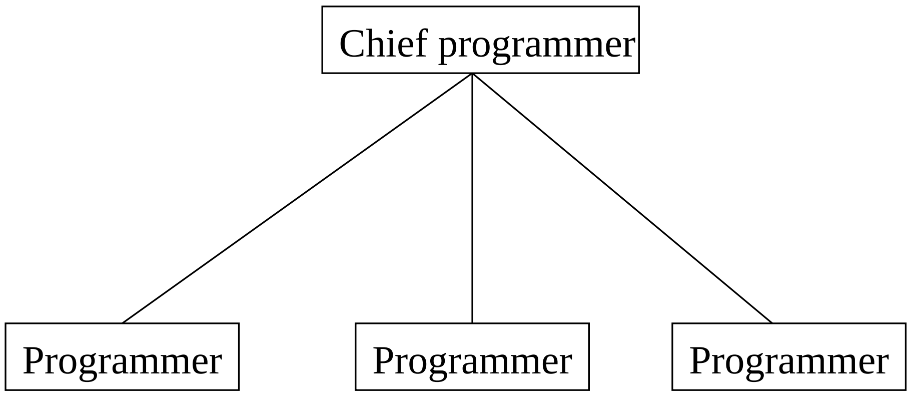

Figure 9.6 Programming in team with fewer communication paths.

Testing during the implementation phase

There are three ways of testing during the implementation phase:

- Black-box
- Glass-box
- Code inspection

Testing during the implementation phase

One popular black-box testing technique is *equivalence-class* testing.

Example.

> A shop is deemed to have achieved its objective for the current month if the shortfall is less than or equal to 5%

First, test cases are derived using partitioning:

- shortfall $< 5\%$,
- shortfall $= 5\%$,
- shortfall $> 5\%$.

Second, these test cases are complemented with boundary cases:

- shortfall $= 1$ $(< 5\%)$
- shortfall $= 4$ $(< 5\%)$
- shortfall $= 5$ $(= 5\%)$
- shortfall $= 6$ $(> 5\%)$
- shortfall $= 39$ $(> 5\%)$

One popular glass-box testing technique is *code coverage*.

Which technique is good where?

- Code inspection finds more interface faults.
- Black-box testing finds more flow-of-control faults.

Recommended application order:

1. Code inspection
2. Black-box
3. Glass-box

When to redesign and recode a function from scratch?

> The more faults are found the greater is the probability that there are still more faults.

Hence after the nth fault a function should be redesigned!!!

CASE-tools in the implementation phase

- *upperCASE* can translate pseudo-code into C
- *indent*
- syntax-directed editors.