CODING

Below is a summary of sections related to software coding, taken from (Pressman, 1987).

The function of a module (a program section performing a particular function) should be apparent without to a design specification. Source code should be self contained. This can be accomplished by coding style. Coding involves several factors; identifier names, commenting, form of source codes, statements, input and etc.

Identifier Names

Identifier names must be meaningful to improve comprehension. However, long names may be difficult to be written a lot of times.

Commenting

Comments provide the developer with one means of communicating with other readers of source code. Never forget that you comment for others. Commenting can be categorised. There are prologue comments appearing at the beginning of every module. A format such comments includes:

- Purpose statement that indicates module functioning.
- An interface description showing how module can be called, argument description, a list of all subordinate modules.
- A discussion of important variables, restrictions, and limitations.
- A development history providing author, reviewer, modification dates and description.

There are also descriptive comments embedded with the body of source code. They are used to describe processing. Comments should provide something extra, not just paraphrase the code. Descriptive comments should describe blocks of code rather than commenting on every line. Use blank lines or indentation for easy reading, and distinguishing from code.

Form of Source Code and Statement

Form of code as it appears on listing is important for readability. Source code indentation indicates logical constructs and blocks of code by indenting from the left margin so that they are detected easily. Most programming languages have code blocking, block beginning and ending delimeters. Indenting code block between these delimeters to right, puts out the program in an understandable appearance. This also frees the developer from syntax errors.

Many programming languages allow multiple statements per line. However, one statement per line is necessary for readable codes. Packing the source code does not affect on executable code but hardens program maintenance.

A source code is a good one if

- complicated or negated condition statements are not used,
- there are not heavy nesting of loops or conditions,
- parentheses are used to clarify logical and arithmetic expressions,
- spacing is used in a good manner to increase readability.

Input/Output

Input and output style varies with respect to human-computer interaction. There are points indicating good input interaction:

- *A* simple, guided input scheme is useful. Keep the input format simple and uniform.
- Label all interactive input requests. Specify available choices or bounding values.
- Use end of data indicators (eg. User should not be asked "How many data items are to be entered?").
- Meaningful input error checking is essential. Validate all input data. Check the plausibility of input items.
- Good input error recovery leads to useable programs.
- A good output needs labelling all output in a readable *form.*

Reference

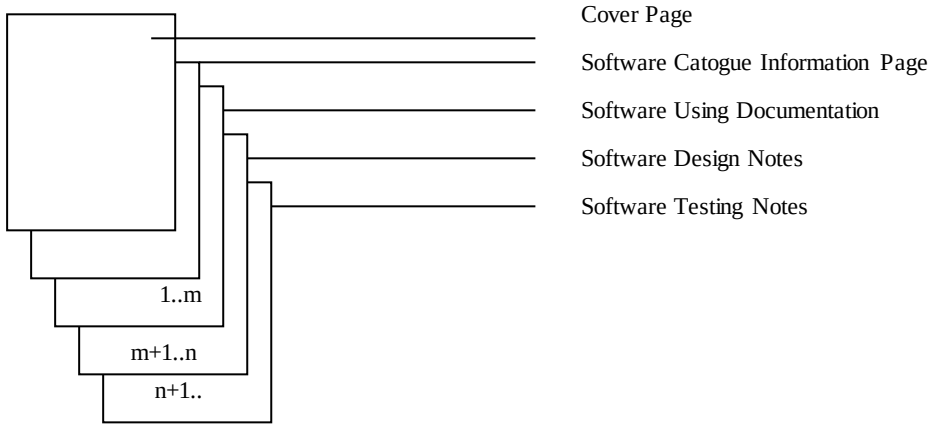
(Pressman, 1987) Pressman**,** "Software Engineering, A Practitioner Approach," Prentice-Hall, 1987

WHAT IS THIS PAPER ABOUT?

This paper provides a document structure for software reporting used in software laboratory courses. Aim is to standardize reports in a clear appearence . This document structure may not be suitable for all kinds of software reports, but it accommodates a common base.

In this paper, each seperate, part is bordered in a page fashion. Throughout these parts, definition sections are printed in italic form, and variable sections to be filled during report writing are printed within angle brackets.

PAGE LAYOUT



SECTION OUTLINES

Cover *page* contains general information for software laboratory advisors.

Hacettepe University
Computer Science and Engineering Department

Name and Surname	: <Your name and surname >
Identity Number	: < Your student number >
Course	: <Code and name of the course>
Experiment	: <Number or/and name>
Subject	: <Subject of the experiment>
Data Due	: <Deadline of the experiment>
Advisors	: <Advisors' names>
e-mail	: <Your e-mail address>
Main Program	: <Main program with name with its full path>

Software Using Documentation *section* refers to the users of the software. Therefore, it should not include anything about software design. It can be separately used as a user's guide.

2. Software Using Documentation

2.1. Software Usage

This section should include information about how the software runs; what the inputs to the program are; screen formats and input and output field explanations; what and how the outputs are; troubleshooting, etc.

(optional)2.2. Provided Possibilities

If there are extra utilities in your software, in this section you should explain these by examples including your reasons why you have included them.

2.3 Error Messages

Error messages are feedback to users. A detailed table of errors is essentiaL A row of this table is as follows.

<error number> :<error in the format user is prompted>
<error conditions>
<solutions to overcome>

Software Design Notes section is specific to software developers. It informs people interested in the software from an enginner point of view.

3.Software Design Notes

3.1. Desctiption of the program

3.1.1. Problem

Under this title, you should state the problem from your point of view.

3.1.2.Solution

Write every solution of the problem, stating its advantages and disadvantages. You should give the details of the solution you have chosen to implement

(not supposed to)3.2. System Chart

A system chart figures programs with their inputs and outputs. Such a chart may be in the form bellow:

INPUT	**PROGRAMS**	**OUTPUT**
<input devices>	<program names>	<output devices>
<Input files>	<with main first>	<output files>

You should give input and output formats in detail

(not supposed to)3.3. Main Data Structures

Important structures should be briefly described by using figures or programming language syntax. Do not explain evey variable in your software.

3.4. Algorithm

An algorithm should bring clarity to its readers. Do not use flow charts for long algorithms. The structure below may be used.

1. Make initialisation.
   1.1. Move zero to Student Count, spaces to StuSuccesfull.
   1.2. Open input: 'STU.DAT', Output: 'STU.REPORT'

2. For every student in 'STU.DAT'
   2.1. Read student data.
   2.2. Calculate mark average of this student
   2.3. Print student with her/his average to 'STU.REPORT'
   2.4. For the processed students
      2.4.1. Find the maximum average among these.
      2.4.2. Store the name of this student to StuSuccesful.

3. Print StuSuccesful to 'STU.REPORT'.
4. Close files

In your algorithm, avoid interpreting your program line by line.

3.5. Special Design Properties
If you have new approaches to the problem. in this section you explain these original approaches in detail.

(not supposed to)3.6. **Execution Flow Between Subprograms**
*Important subprograms should have a brief description here***.**

<-subprogram name with parameters>
<what it does>,
<what it returns>,
<where it is called>.
<what it calls>.

Software Testing Notes *:*

This s*ection* points out the success of your software. It shows the software engineering degree of your software.

4. SOFTWARE TESTING NOTES :

4.1. Bugs and Software Reliability

Give all bugs ,your points will not reduced if you list bugs frankly.

(not supposed to)4.2. Software Extendibility and Upgradability

If additional features may be easily added to your software explain them. Give the approximated program length in lines, and working effort in hours.

(not supposed to)4.3. Performance Considerations

You should include speed of your program, subroutines. For personal computers it is easy to calculate these speeds. However you should use some special routines for obtaining real elapsed time in multiprocessing /multi-user environments.

In mathematical calculation, give the maximum precision that software can handle. There may be other kinds of performance considerations.

(optional)4.4. Comments

You may comment on the problem, problem description, and solution constraints.

REFERENCES

You add the references you used throughout your report at the end of this documentation. A common format for references is:

<reference no> <Author Surname, Name>,
<publication name>,
<publisher>,
<publishing date>.