Software Development Life cycle
Maintenance Manual

Author: Tom Reed, Matt Whitmore, Dave Clark, Silhab Csoma, Mike Steel, Chris 'Tux' Lloyd, Aleksandra Badyda, Samuel Jackson, Chris Marriott

Config. Ref.: SE.17.DS.01
Date: 2013-01-30
Version: 1
Status: draft

Department of Computer Science,
Aberystwyth University,
Aberystwyth,
Ceredigion, SY23 3DB,
U.K.

CONTENTS

1 Maintenance Manual

Program maintainers pick up the maintenance documentation because they have a specific question in mind. The goal of your program maintenance documentation should be to answer all of the likely questions, or at least to show the maintainer which part of the program source is likely to provide the answer. Examples of maintainers questions are:

- This program crashes with a particular combination of inputs. Where is the bug likely to be, and how do I rebuild and test the system once I have fixed it?
- How can I extend this sort program to add an option which will delete any repeated lines found while sorting? Indeed, is its structure such as to permit such a modification without altering its design completely?
- Can I speed up this slow program without major change? For example, I have found a new and wonderful sorting procedure. Can I replace the slow sort used in this genealogical data management program?

Most programs do in fact contain bugs. Normal commercial practice, especially with minor bugs, is to document them and correct them at a convenient time (perhaps 3-12 months later), rather than rushing to fix them and releasing a new version immediately. Making immediate fixes is costly in distribution and reinstallation of new versions and above all leads to problems when the fixes have unforeseen side effects. This lead-time on fixing bugs means that documenting any likely changes and how to make them is vital, because the changes may be carried out by someone else. A checklist for the structure of a maintenance manual is:

1.1 Program Discription

This will give a brief description of what the program does and how it does it, e.g. for a sorting program you might say It sorts a list of English words into alphabetical order using the bubble sort method.

1.2 Program Structure

This should describe the design of the program. Design diagrams and pseudo-code are both useful methods of doing this. One of the most useful diagrams for a maintainer is one that shows which routines in the program call which other routines (a flow of control diagram). A list of program modules and their purpose should also be given. There should also be a list of methods. This specifies the name,

parameters and their types, the type returned by a method, and a brief description of what each method does. Often a couple of lines on each module will be enough. Where this information is contained in your design specification, it can be left out of the maintenance document, and a reference made to the appropriate sections of the design specification.

1.3 Algorithms

Here you describe in detail the significant algorithms used in the program or, in the case of well known methods, you may give references. Again, if this information is contained in the design specification, it can just be referenced here.

1.4 The main data areas

This specifies the data structures, including arrays, objects etc. where important information is stored for a substantial part of the main program. For example, in a program that adds a students marks together and calculates a grade, there might be data structures used to store a students project and examination marks for each course. Again, if this information is contained in the design specification, it can just be referenced here.

1.5 Files

It may be that the program accesses certain fixed files or needs files of a certain type to be available. Give such information here. For example, The program creates the file XYZ.test as workspace and later deletes it. If such a file exists already then its contents will be lost. Another example is The program assumes that the current directory contains a file of integers at three per line, separated by spaces.

1.6 Interfaces

Many programs control or read devices such as measuring instruments. Usually there will be certain protocols to be observed, requirements that a terminal is set up in a particular way, etc. For example, The terminal should be set to read and transmit at a baud rate of at most 1200. The possibilities here are endless, but each application is likely to have a few simple rules that must be observed, and such information should be given in this section.

1.7 Suggestions for improvements

Most programs are a compromise between what one would like to do and what one has time to do. Where desirable improvements have had to be omitted because of constraints such as time or the available hardware or software, it is worth suggesting them for the benefit of future programmers tackling the same problem. Where different ways of solving some of the programming problems have been considered, then it can be useful to others who need to work on the same program to have this information. It may be that improvements in hardware or software mean that methods rejected now can be used when the program is revised.

1.8 Things to watch for when making changes

It is desirable to avoid a programming style which means that changes can have knock-on effects which affect other parts of the program. Sometimes this is unavoidable. The programmer should be very careful to list any known effects of this nature.

1.9 Physical limitations of the program

Obviously a computer installation is a finite environment. It can only have so much memory, so much disk space, and only allow each user so much processor time. Some programs will come against these constraints. It is particularly important to list the requirements, where known, because not all environments impose the same constraints - a program might run without restriction in one environment, require special options to be chosen in another, and be completely beyond the capabilities of a third. There is also the question of accuracy when real numbers are used. The documentation should discuss this, if relevant, giving details of the expected accuracy of inputs, that provided by the algorithms, and that of the output.

1.10 Rebuilding and Testing

Maintainers need to know what to do when rebuilding a program. Where are all the files? What should they do to rebuild the system? How do they find out what tests to run? How do they know whether it has passed the tests? How do they add a test when a new problem is discovered? If documents are in a non-standard format (e.g. LaTeX rather than MS Word), then it may also be necessary to describe how to rebuild them.

REFERENCES

[1] *N/A*

DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2013-01-30	Initial creation	dac26