Software Engineering Group Projects – Producing a Final Report

Author:	C. J. Price, N.W. Hardy and B.P.Tiddeman
Config Ref:	SE.QA.11
Date:	1st October 2012
Version:	1.6
Status:	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1 INTRODUCTION

1.1 Purpose of this Document

This document sets out the format and expected contents of the final project delivery from a Software Engineering Group Project in the Department of Computer Science, Aberystwyth University (AU).

1.2 Scope

This document states which documents need to be prepared specifically for the final group project delivery, and gives guidelines for the length of those documents, and what they should contain. It specifies how those documents plus all other project material should be delivered.

It should be read by the Project Leader and QA Manager, and relevant contents should be communicated to other members of the group.

1.3 Objectives

The aim of this document is to make clear to project teams what is expected from the end of project delivery. In particular, it gives guidelines for the production of the following documents:

1. The End-of-Project Report.
2. The Project Test Report.
3. The Project Maintenance Manual.

It also gives a checklist for other documents and code which should be included, and describes how it should all be packaged in order to provide best evidence to the markers that the project was a success.

2 Extra Documents Produced for the Final Delivery

2.1 The End-of-Project Report

The end of project document should state clearly how much has been accomplished on the project. The purpose of the report is to enable the markers to evaluate how well you have done the project.

It should be written as a coherent submission (i.e. as if one person wrote it, not as separate accounts of the project from each team member), and should provide the reader with the following items:

A management summary. This should sum up in one page what the project achieved (what parts of the program work and what parts do not; which documents are in a good state and which are not), what difficulties stood in the way of project completion and how they were overcome, and how well the team performed.

A historical account of the project. This should outline the main events over the lifetime of the project, and how the project team acted to produce a plan and to deliver a product within a constrained lifetime. This should take no more than two pages of A4.

Final state of the project. This should give a summary of which parts of the project are perceived as correct and which are not. It is as well to be as accurate as possible here - more marks will be deducted for problems that are not declared but are detected by the markers than for problems that are declared in the final report. As well as missing or erroneous features in the software, known problems with documents should be included here.

Performance of each team member. The project leader should write a half page description of the duties and performance of each group member, including the group leaders themselves. This should be agreed with the group member if possible, and it should state whether agreement was reached, and if not, should give an explanation why not.
Critical evaluation of the team and the project. This should be no more than a page in length and should address the following subjects:

1. How did the team perform as a whole, and how could that have been improved?
2. How could the project have been improved?

3. What were the most important lessons learned about software projects and about working in teams?

2.2 Appendices

2.2.1 The Project Test Report

This document will be a list of the named/numbered tests planned for the project, plus for failed tests, an explanation of why the test failed if available. In some cases, it may be possible to group failed tests with a single explanation. In such a case, the explanation should be given once, and referenced elsewhere (e.g. *see explanation for test ST22-01-12*).

This report should have a header page, but does not need purpose/scope/objectives - the body of the document can just be a table of tests, the result for each test, and explanations where necessary.

2.2.2 The Project 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:

1. 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?
2. 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?
3. 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:

Program description. 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*.

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.

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.

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 student's marks together and calculates a grade, there might be data structures used to store a student's project and examination marks for each course. Again, if this information is contained in the design specification, it can just be referenced here.

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*.

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.

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.

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.

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.

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.

2.2.3 Personal reflective report

In addition to the deliverables specified above, each person must submit a personal reflective document based on their reflective journal. Individuals may wish to include in this any issues relevant to the assessment of the project which might otherwise go unreported. An example would be problems that arose from personality clashes and led to seemingly unfair allocation of duties. Usually this report would be no more than 1-2 sides A4.

If there are any personal difficulties which the individual wishes to raise but not put them in the personal report they are encouraged to raise them with the group tutor or module coordinator as early as possible.

3 Structure of the Final Delivery

3.1 Final Report

The project final report should be handed in on paper, bound, along with a copy of the project filestore on CD along with a top-level web page indexing the filestore, as described below. Electronic copies of the report and the (zipped) filestore should also be submitted on BlackBoard. A useful checklist for what you should hand-in is provided in the Review Standards document [1].

3.2 Format of CD / blackboard submission

The CD or DVD and zipped BlackBoard hand-in should contain a mirror of the project filestore, in directly accessible form (i.e. not encoded or compacted, and with full length filenames in platform independent format).

3.3 Online access to materials

All documents should be provided in a format which can be examined on screen. This typically means PDF.

Where relevant, all code should be compiled (Java for example) and linked into .jar , .war or other appropriate executable formats (in addition to providing source code).

A top-level Web page should be provided on the CD, with links to all of the documents delivered, so that a chosen document can be selected with a click.

Installation and customisation information should be readily accessible.

REFERENCES

[1] QA Document SE.QA.07 - Review Standards.

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	28/01/02	First version of document	CJP
1.1	N/A	18/07/02	Updated after review with Graham Parker. Reference to review checklist added.	CJP
1.2	N/A	14/09/06	Updated number of document from SE.QA.10	CJP
1.3	N/A	12/09/08	Changed document template to be Aber Uni	CJP
1.4	N/A	04/10/10	Moved to DocBook. Deliverables updated.	NWH
1.5	N/A	01/10/10	Reverted to word, final report now includes all parts (installation manual, test report, personal reports) in a single doc.	BPT
1.6	N/A	11/10/11	Clarified submission should be both physical copy and on BB	BPT