GUIDELINES FOR DOCUMENTS PRODUCED BY STUDENT PROJECTS IN SOFTWARE ENGINEERING

Declan Delaney and Stephen Brown
Department of Computer Science, NUI Maynooth
September 2002
Technical Report: NUIM-CS-TR2002-06

Abstract

This technical report provides detailed guidelines for the contents of a minimal set of software development documents, tailored for use by students in software engineering projects, and based on IEEE standards, as described in technical report NUIM-CS-TR2002-05 [3]. It is intended to be read along with that report, and used to assist in determining the exact contents of each section.

1 Background

Many engineering university courses fail to address the issue of documentation, and students struggle when they are expected to produce professional documentation in the workplace [2]. If documentation best-practice is not covered as part of the curriculum, then it is difficult to subsequently train students in industry to expected standards [12]. This is the motivation for producing these guidelines, which provide the essential basis for an understanding of documentation best practice (this underlies many of the important skills of a software engineer, as described, for example, in [15]).

These guidelines do not advocate the use of any particular development process, and do not address the issue of assessment. They emphasise particular issues that are most pertinent to best practice in industry. Software projects are a critical element of a software engineer's education [14] – these guidelines also encourage the students to reflect on their project work [16].

These guidelines are based on the IEEE standards for software engineering documentation [4][5][6][7][8][9][10][11]. These have been significantly refined based on the authors' experience (both in industry and in teaching software engineering), and other educational documentation standards – such as the University of South-western Louisiana [13], and the University of Texas at Austin which has particularly well developed standards [1]. The refinement has three purposes:

1. to simplify the IEEE wording to make it accessible to undergraduate students;

2. to make students aware of the key components of software development projects through use of documentation;

3. to provide a roadmap to encourage the students to explore connections between the various software development activities.

Once students have gained experience, then they will be in a position to make good use of the full IEEE standards.

In this technical report, guidelines are provided for the following:

Section	Description
Section 2	Sections Common to all documents
Section 3	Software Project Management Plan (SPMS)
Section 4	Software Requirements Specifications (SRS)
Section 5	Software Design Description (SDD)
Section 6	System Test Document (STD)

2 Sections Common to all Documents

The following paragraphs describe sections that are common to all documents. The Cover Page and Revisions Page come at the beginning of the document, whereas the Additional Material is included at the end.

2.1 Cover page

The cover page should provide the following information:

1. Name of the document - *not the filename*;
2. Project title
3. Document version number – *never 'release' two different copies with the same version number – it is good practice to increment the version number every time you finish making a change (or set of changes) to the document*
4. Printing date - *this helps to remove confusion when you don't use version numbers properly!!*
5. Location of the electronic version of the file - *for example: server, folder/directory, and filename;*
6. Department and University name.

2.2 Revisions page

The revisions page should provide the following information:

1. Overview: *a very brief description of the document*
2. Target Audience: *specify who will read, approve, and use the document*
3. Project Team Members
4. Version Control History: *use a table showing (for each 'released' revision) the* Version, Primary Author(s), Description of Version, Date Completed
5. Signatures of Approval: *provide space for reviewers to sign the final version of each document to indicate their approval of its contents*

2.3 Additional Material

Each document may contain additional material. Some of this is specified for each document, in the form of appendices, but the following apply to each of the 4 documents:

2.3.1. ADDITIONAL ISSUES: *space for any additional issues you feel are necessary for the document*

2.3.2. DFINITIONS, ACRONYMS, AND ABBREVIATIONS: *make sure to define any special terms you use in the document*

2.3.3. RFERENCES: *the exact format is not critical, but it must be possible to easily find each reference you cite in your document in this section, and to then find the actual document. Refer to a paper in a well-known journal to get examples – and stick to a consistent style. You must specify:*

- ✍✍*the names of the authors*
- ✍✍*for a journal paper: the name of the paper, the name of the journal, the month or volume or issues of the journal, the publisher and year of the journal*
- ✍✍*for a conference paper: the name of the paper, the name of the conference, the date of the conference, the location of the conference, the publisher and year and publication of the conference proceedings*
- ✍✍*for a book: the name of the book, the edition of the book, the publisher and year of publication.*

It is good practice to include the ISBN or ISSN (if known). Web-references should be avoided if possible – refer to the document contained in the web page if you can. If you must use web references, try to use a reference from an institutional web site (such as: a library, ACM, IEEE, W3C, IETF), rather than a company-specific web site (that may well change or disappear).

For example:

> C. Brammer and N. Ervin, "Bridging the Gap: A Case Study of Engineering Students, Teachers, and Practitioners", in *Proceedings of the 1999 Professional Communication Conference*, Pages: 251-255, IEEE, 1999

2.3.4. APPENDICES: *additional appendices are where you can place any other additional material.*

2.4 TERMINOLOGY

Components *– in most methodologies, a software design is composed of a number of sub-systems, referred to here as components. In turn these may be composed of further entities in a hierarchical manner. Exactly what you mean by a component should be defined within your documents and consistency should be maintained within and across project documents. For example, using OO terminology, a system might consist of multiple subsystems, each of which might in turn consist of multiple objects. Or alternatively, objects might be grouped into subsystems, which in turn might be grouped to form a system. The subsystems and objects are all components.*

3 Software Project Management Plan (SPMP)

Relevant IEEE standards: IEEE-1058[9], IEEE-1540[10]

The software project management plan specifies:

?? *the objectives of the project,*

?? *the project dependencies and constraints,*

?? *the project deliverables, and*

?? *the project timetable.*

This document describes how you intend to develop your software; how you expect to approach the development process; the times and deliverables involved; problems foreseen and constraints imposed. The plan accompanies the project life span and is usually subject to a number of revisions.

1 INTRODUCTION

1.1 Project Overview

Provide a description of the purpose of the software, who the project is being executed for and the expected delivery date.

1.2 Project Deliverables

List the project deliverables/" work products" (documents, source code, library files, executable code, databases), and the delivery date for each (the date that each is to be complete).

2 PROJECT ORGANIZATION

2.1 Software Process Model

Identify and describe the software process model you will use in terms of the following:

?? *the flow of information and work products,*

?? *reviews to be conducted,*

?? *major milestones to be achieved,*

?? *versions to be established,*

?? *project deliverables to be completed,*

To describe the process model, a combination of graphical and textual notations may be used.

2.2 Roles and Responsibilities

In the case of a group project, identify the various roles, the project team members, and their assignments. Take care to identify how information will be communicated between the different roles – for example: how will the design be passed to the coder.

Use diagrams to show the project structure, and the lines of communication within the project.

2.3 Tools and Techniques

Specify the development methodologies, notations, programming languages, techniques, and tools you plan to use at each stage of the project.

For example: you might use OO for the analysis phase, using the UML notation, and the Rational Rose Tool. In order to track changes to the analysis, you might use version control, and the CS-RCS tool.

This section should include details of the coding standards to be used.

3 PROJECT MANAGEMENT PLAN

3.1 Tasks

Identify the tasks involved in executing the project. Some sample tasks are:

- ?? *requirements analysis/clarify the requirements*
- ?? *requirements analysis/develop an exploratory prototype and get feedback from the user*
- ?? *requirements analysis/write the SRS*
- ?? *system design/develop the design using Rational Rose*
- ?? *system design/write the SDD*
- ?? *development/develop version 1*
- ?? *development/develop version 2*
- ?? *system test/test version 1*
- ?? *system test/test version 2*

Tasks should be specified in enough detail to allow estimation of the time required, and to allow tracking of project status. As a very rough guide, tasks should take about one week.

3.1.n Task-*n*

Provide a name for the task, and a unique identifier of the form SPMP-Tnnnn.

3.1.n.1 Description

A brief description of the task.

3.1.n.2 Deliverables and Milestones

A list of the deliverables of the task (documentation, code, other), and the milestones associated with these. For example, milestone 1 of the project might be the delivery of the final revisions of the SRS, STD, and SPMP; milestone 2 might be delivery of the STD, milestone 3 might be completion of version 1 (development & test), etc.

3.1.n.3 Resources Needed

Identify the resources needed to execute the task (equipment, access to papers/books, etc.)

3.1.n.4 Dependencies and Constraints

Identify the dependencies that must be met before the task can be started or completed, and any constraints placed on the task.

For example: detailed design can't start until the requirements are complete, or there is a constraint that Java be used as the programming language.

3.1.n.5 Risks and Contingencies

Identify the risks associated with getting this task completed on time, and the contingency plans to cope with this risk. You can never identify all the risks; and some events would just stop the project completely. Limit this section to reasonable risks.

For example: what is the probability of you being unable to get the programming language you plan on using running on your computer? What would you do to recover?

3.2 Assignments

For group projects, identify the assignment of team members to tasks.

3.3 Timetable

Provide a timetable showing the estimated start and completion dates for each task. This is best illustrated using a Gantt diagram.

This timetable should be updated throughout the project to reflect changes in: the tasks, the execution time for tasks, the completion dates, etc.

4 Software Requirements Specifications (SRS)

Relevant IEEE standards: IEEE-830[5]

The SRS is a specification of the requirements for the software "product" you will produce in your project. The basic issues to be addressed are:

a) Functionality. What is the software supposed to do?
b) External interfaces. How should the software interact with people, the operating system, hardware, networks, and other software?
c) Performance. What are the requirements for speed, availability, response time, recovery time of various software functions, etc.?
d) Quality Attributes. What are the requirements for portability, correctness, maintainability, security, etc?
e) Design constraints imposed on an implementation. Is there a requirement for a particular programming language? Are there resource limits (such as disk or memory size)? Must it run on a particular operating system? Must it inter-operate with particular web browsers? etc.

*The SRS contains **requirements**, and not your **design solutions**; it is the "what" and not the "how" of your project. The information is collected from the project client.*

In a good SRS, the requirements should be: Correct, Unambiguous, Complete, Consistent, Ranked for importance, Verifiable, and Traceable.

*A requirement is **verifiable** if there exists a way to check that the software meets the requirement. Non-verifiable requirements include statements such as "works well", "good human interface". If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised.*

Requirements must be traceable through design, implementation, and system test – that is, it must be possible to trace each requirement to the parts of the design that support it, to the code that supports it, and to make sure it has been tested in the final software.

1 INTRODUCTION

1.1 Product Overview

Clearly define the purpose of the software, the environment it will run in, and who will be using it (in terms of their educational level, experience, and technical expertise). Do not go into detail, but outline the general requirements in a way that provides the reasons why specific requirements are later specified in the SRS.

2 SPECIFIC REQUIREMENTS

This section of the SRS should contain all of the software requirements to a level of detail sufficient to:

1. *enable the SRS to be checked by the originator of the original system requirements (usually the project supervisor),*
2. *enable designers to design a system to satisfy those requirements, and*
3. *testers to test that the system satisfies those requirements.*

Throughout this section, every stated requirement should be externally perceivable by users, or other external systems. This ensures that all features are testable.

2.1 External Interface Requirements

2.1.1 User Interfaces

This should specify the following:

a) The characteristics of the user interface. Include the characteristics necessary to accomplish the software requirements (for example: required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys).

b) All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do's and don'ts on how the system will appear to the user. Like all others, these requirements should be verifiable.

2.1.2 Hardware Interfaces

This should specify the characteristics of each interface between the software and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. Only use this section if your project requires specific hardware to be used – do not, for example, specify the hardware if there is a requirement to establish a network connection.

2.1.3 Software Interfaces

This should specify the use of other required software which your software must interface with (for example: a database system, an operating system, or a mathematical package).

For each required software product, the following should be provided:

- Name and Version number.

For each interface, the following should be provided:

- Discussion of the purpose of the interfacing software as related to this software product.

- Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

2.1.4 Communications Protocols

This should specify the various interfaces to communications such as local network protocols, etc. Make reference to any well defined protocols, specifying exactly which parts or options of that protocol the software needs to support.

2.2 Software Product Features

This section should consist of a numbered list of required features.

*Typically, all of the requirements that relate to a software product are not equally important. The importance of each feature should be indicated using one of the following terms: **essential**, important, or **desirable**.*

Each feature requirement should include:

- ?? *a description of every input (stimulus) into the system,*
- ?? *a description of every every output (response) from the system,*
- ?? *a description of every state change within the system,*
- ?? *a description of all the functions performed by the system in response to an input or in support of an output.*

Functional requirements should define the actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as "shall" statements starting with "The system shall". These include

a) Validity checks on the inputs

b) Exact sequence of operations

c) Responses to abnormal situations, including

- *1) Overflow*
- *2) Communication facilities*
- *3) Error handling and recovery*

d) Effect of parameters

e) Relationship of outputs to inputs, including

- *1) Input/output sequences*
- *2) Formulas for input to output conversion*

It may be appropriate to partition the functional requirements into subfunctions or subprocesses. This does not imply that the software design will also be partitioned that way.

2.3 Software System Attributes

2.3.1 Reliability

Specify the required reliability of the final software system. This is particular important for applications such as embedded software. It can be quantified using MTTF (Mean Time To Failure) measurements.

2.3.2 Availability

Specify the required availability of the final software system: define requirements such as checkpointing, recovery, and restart.

2.3.3 Security

Specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure.

Specific requirements in this area could include the need to
a) Utilize certain cryptographical techniques;
b) Keep specific log or history data sets;
c) Assign certain functions to different modules;
d) Restrict communications between some areas of the program;
e) Check data integrity for critical variables.

2.3.4 Maintainability

This should specify attributes of the software that relate to the ease of maintenance of the software itself. Specify any requirements for certain modularity, interfaces, complexity, etc that make the software easier to maintain.

Requirements should not be placed here just because they are thought to be good design practices.

2.3.5 Portability

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

a) Percentage of code that is host dependent;

b) Use of a proven portable language;

c) Use of a particular compiler or language subset;

d) Use of a particular operating system.

2.3.6 Performance

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole.

Static numerical requirements may include, for example, the following: minimum number of simultaneous users, minimum data storage, etc.

Dynamic numerical requirements may include, for example, the required number of transactions per second for both normal and peak workload conditions, etc.

All of these requirements should be stated in measurable terms.

2.4 Database Requirements

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

a) Types of information used by various functions;

b) Accessing capabilities;

c) Data entities and their relationships;

d) Integrity constraints.

Notes:
The appendices should include verification of requirements consistency. This may be in the form of formal proofs, appropriate diagrams, or other notation.

5 Software Design Description (SDD)

Relevant IEEE standards: IEEE-1016[8]

The SDD shows how your software will be structured to satisfy the requirements. It describes the software structure, software components, interfaces, and data necessary for the implementation phase. In a complete SDD, each requirement must be traceable to one or more components.

An SDD is a representation or model of the software system to be created. The model should provide the precise design information needed for planning, analysis, and implementation of the software system. It should represent a partitioning of the system into components and describe the important properties and relationships between them.

1 INTRODUCTION

1.1 Design Overview

Give a description of the design approach, highlighting essential features that allow the design to meet the stated requirements.

1.2 Requirements Traceability Matrix

Provide a matrix showing where each feature identified in the SRS is supported by the design components.

	Component 1	***Component 2***	***Component 3***	***Component 4***
Requirement 1		*X*		*X*
Requirement 2	*X*	*X*	*X*	*X*
Requirement 3			*X*	
Requirement 4	*X*			*X*

If you are developing the software in multiple increments, then a traceability matrix should be produced for each version.

2 SYSTEM ARCHITECTURAL DESIGN

2.1 Chosen System Architecture

Describe the system architectural design, identifying the major component groupings and the interfaces (both internal and external). Make sure to identify any significant technical risks, and identify contingency plans for each.

2.2 Discussion of Alternative Designs

Discuss in a reasonable level of detail other design options explored, and the reasons for not choosing them.

2.3 System Interface Description

Describe the system interfaces in detail: O/S interface, files, networking, libraries, graphics libraries etc. (Describe the user interface in section 4.)

3 DETAILED DESCRIPTION OF COMPONENTS

3.*n* Component-*n*

For each component, the following items should be described here as appropriate: responsibilities, constraints, composition, interactions, and resources. Use appropriate diagrams or other notation to describe your design.

4 USER INTERFACE DESIGN

In this section describe the design of the user interface in detail.

4.1 Description of the User Interface

4.1.1 Screen Images

Show the design of layout and menus for each screen.

4.1.2 Objects and Actions

Identify all the objects on each screen, and define the actions to be taken by each object for each event.

Notes:
The appendices should include the design verification. This should consist of formal proofs, sequence diagrams, or other notations as appropriate. The level of detail should be adequate to demonstrate that the design meets the requirements.

6 System Test Document (STD)

Relevant IEEE standards: IEEE-829[4], IEEE-1008[6], IEEE-1012[7]

This system test document specifies the tests for the entire software system, defines the test schedule, and records the test results.

This document does not cover unit testing (the testing of individual sub-systems or components of the system).

The system may consist of multiple items that are to be tested separately.

The system may be tested in one or more increments of functionality; the system test document should cover each version of the system separately. When different versions of the system are tested, make sure to clearly identify the version of the software and the relevant test and test result. Also, extend the unique identifier scheme to include the version of the software system under test (SUT) – for example, use TC-vvvv-nnnn to identify test case nnnn for software system version vvvv.

1 INTRODUCTION

1.1 System Overview

Briefly detail the software system and items to be tested. Identify the version(s) of the software to be tested.

1.2 Test Approach

Describe the overall approach to testing. For each major group of features or feature combinations, specify the approach that will ensure that these feature groups are adequately tested. Specify the major activities, techniques, and tools that are used to test the designated groups of features. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each one. Identify significant constraints on testing, such as deadlines.

2 TEST PLAN

Describe the scope, approach, resources, and schedule of the testing activities. Identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task in the case of a group project.

2.1 Features to be Tested

Identify all software features and combinations of software features to be tested. Identify the test case(s) associated with each feature and

each combination of features. Identify the version of the software to be tested.

When multiple versions of the software are tested in a planned, incremental manner, then use section numbers 2.1.n to identify the features to be tested for each version.

2.2 Features not to be Tested

Identify all features and significant combinations of features that will not be tested and the reasons for not doing so.

2.3 Testing Tools and Environment

Specify test staffing needs. For an individual project, specify the time to be spent on testing. For a group project, specify the number of testers and the time needed.

Specify the requirements of the test environment: space, equipment, hardware, software, special test tools. Identify the source for all needs that are not currently available.

3 TEST CASES

A test case specification refines the test approach and identifies the features to be covered by the case. It also identifies the procedures required to accomplish the testing and specifies the feature pass/fail criteria. It documents the actual values used for input along with the anticipated outputs.

If an automated test tool is to be used:
1. *document each test case here as a specification for the test tool;*
2. *document the procedure that must be followed to use the test tool.*

3.*n* Case-*n* (*use a unique ID of the form TC-nnnn for this heading)*

3.*n*.1 Purpose

Identify the version of the software and the test items, and describe the features and combinations of features that are the object of this test case. For each feature, or feature combination, a reference to its associated requirements in the software requirement specification (SRS) should be included.

3.*n*.2 Inputs

Specify each input required to execute the test case. Some of the inputs will be specified by value (with tolerances where appropriate), while others, such as files or URLs, will be specified by name. Specify all required relationships between inputs (e.g., ordering of the inputs).

3.*n*.3 Expected Outputs & Pass/Fail criteria

Specify all of the expected outputs and features (e.g., response time) required of the test items. Provide the exact value (with tolerances where appropriate) for each required output or feature. Specify the criteria to be used to determine whether each test item has passed or failed testing. If an automated test tool is used, identify how the results of that tool are to be analysed.

3.*n*.4 Test Procedure

Detail the test procedure(s) needed to execute this test case. Describe any special constraints, such as: special set up, operator intervention, output determination procedures, and special wrap up.

Notes:

APPENDIX A. TEST LOGS

A test log is used by the test team to record what occurred during test execution.

A.n Log for test-*n* (*use a unique ID of the form TL-nnnn for this heading)*

A.n.1 Test Results

For each execution, record the date/time and observed results (e.g., error messages generated). Also record the location of any output (e.g., window on the screen). Record the successful or unsuccessful execution of the test.

A.n.2 Incident Report (*add a unique ID of the form TIR-nnnn to this heading)*

If the test failed, or passed with some unusual event, fill in this incident report with the details. Summarize the incident, identifying the test items involved, and the anomaly in the results. Indicate what impact this incident will have on the project.

7 Conclusions and Further Work

In this report we have presented explanations for the student software engineering documentation templates introduced in TR05. These explanations are based on the IEEE standards. Their purpose is to guide the students through the process of documenting their software development, and raise their awareness of certain key issues in software engineering.

We recognise that each software project is unique and so we do not advocate using a specific software engineering process based on these templates. In the future we hope to find time to document approaches that students take in producing software and identify the most common processes and pitfalls.

References

[1] V. Almstrum, *CS373: S2S Project Documentation Standards*, Department of Computer Sciences, University of Texas at Austin, January 15, 2002

[2] C. Brammer and N. Ervin, "Bridging the Gap: A Case Study of Engineering Students, Teachers, and Practitioners", in *Proceedings of the 1999 Professional Communication Conference*, Pages: 251-255, IEEE, 1999

[3] D. Delaney and S. Brown, *Document Templates for Student Projects in Software Engineering*, Technical Report NUIM-CS-TR2002-05, Department of Computer Science, National University of Ireland, Maynooth, 2002

[4] IEEE Std. 829-1998 *IEEE Standard for Software Test Documentation*

[5] IEEE Std. 830-1998 *IEEE Recommended Practice for Software Requirements Specifications*

[6] IEEE Std. 1008-1997 *IEEE Standard for Software Unit Testing*

[7] IEEE Std. 1012-1998 *IEEE Standard for Software Verification and Validation*

[8] IEEE Std. 1016-1998 *IEEE Recommended Practice for Software Design Descriptions*

[9] IEEE Std 1058-1998 *IEEE Standard for Software Project Management Plans*

[10] IEEE Std 1540-2001 *IEEE Standard for Software Life Cycle Processes – Risk Management*

[11] IEEE 12207.2-1997 *Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology - Software Life Cycle Processes - Implementation Considerations*

[12] P.S. Katz and T.E. Warner, "Writing as a Tool for Learning", *IEEE Transactions on Education*, Vol. 31, No. 3, August 1988

[13] R. A. McCauley, U. Jackson, B. Manaris "Documentation Standards in the Undergraduate Computer Science Curriculum", in *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, Pennsylvania, February 1996, Pages: 242-246, ACM 1996

[14] R. Thomas, G. Semeczko, H. Morarji, G. Mohay, "Core Software Engineering Subjects: A Case Study ('86 - '94)", in *Proceedings of the Software Education Conference 1994*, Pages: 24-31, IEEE, 1995

[15] S. Tockey, "Recommended Skills and Knowledge for Software Engineers", in *Proceedings of the Twelfth Conference on Software Engineering Education and Training*, Pages: 168-176, IEEE, 1999

[16] R.L. Upchurch and J.E. Sims-Knight, "Reflective Essays in Software Engineering", in *Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference*, Vol. 3, Pages: 13A6/20, IEEE Computer Society, 1999