QA 01 – Quality assurance plan

Reference of QA plan:

QA 01 – Quality assurance plan which covers overall QA activities

QA 02 –General Documentation Standards

QA 03 –Project management Standards

QA 04 –User Interface Specification Standards

QA 05 –Design Specification Standards for Software Engineering projects

QA 06 –Test Procedure Standards

QA 07 –Review Standards

QA 08 –Operating Procedures and Configuration Management Standards

QA 09 –Coding Standards

QA 10 –Producing Final Report

AIM – To give us guidelines of developing a professional software.

Quality Assurance Plan covers:

- Project management and quality control activities
- The documentation
- Meetings, reviews and walkthroughs
- Software configuration management
- Problem Reporting and corrective activities
- Tools, techniques and methodologies for building a software
- Quality Assurance Manager's responsibilities

Documentation it covers:

- Final Report
- Test Report
- Design specification
- Test specification
- User Interface Specification

QA 02 –General Documentation Standards

Minutes of meeting should include pain text, should be emailed within 24 hours after the meeting. They should include:

- Project title, with the name of the project
- The meeting purpose
- List of people present and any apologies
- The place, the date of the meeting
- Circulation list including all present plus other persons as appropriate (everybody should receive the minutes)
- The author of the minutes
- The Date the minutes were produced
- The version control number of the minutes

Following everything above:

- Matters arising – progress of actions from previous meetings
- New Business –describes each significant new peace of business *numbering each one of them*.
- AOB –Items discussed as *Any Other Business* numbering each one of them

For all sections of the minutes on new line should be" ACTIONS:" followed by the email of the person responsible

Following the end of AOB section must be the minutes' writer email address and the date minutes were produced.

Copies of all minutes must be stored in the project's configuration directory as described in QA 08

DOCUMENTS

All formal documents except User Interface Presentation must contain the following in the *front cover:*

- A title which indicates the project and nature of the document
- The author(s) of the document
- The configuration reference for the document – see QA 08
- The date the latest version of the document was produced
- The version number of the document. Should increment if a new release of the same.
- The document status –see QA 08
- The name and address of Department, together with a copyright notice

Each page of the document must contain

a) A page header containing:

        i) The title of the document
        ii) The version number of the document
        iii) The status of the document in brackets
    b) A page footer containing:
        i) The phrase *Aberystwyth University / Computer Science;*
        ii) The page number in the form: page x of y.

Sections which must be included in document

1. Contents
2. Introduction
3. Specific Sections Are HERE
4. References
5. Document Change History

Documents should be produced according to QA 08. All content should be checked for spelling and other typographical errors using the automated spell checking program before released.

All diagrams should be according to QA 08.

QA 03 –Project Management Standards

Specifies the tasks to be carried out by the Project leader with assistance and advice of the Project Manager

- Identifying tasks to be executed during the project
- Planning those tasks
- Ensuring the tasks are monitored and assessed

  So that the project is completed successfully and on time.

Organisation:

The project will have a member of staff as:

- Project Manager

Student from the group as:

- Project leader –see QA 01
- Quality Assurance Manager

Major Activities

- Project management –monitors progress and directs it

- Quality Assurance
- Spike Work
- Designing the system
- Writing the code
- Testing the system
- Producing maintenance information
- Producing end of project report

Project planning has two main sections – Project plan and Deliverables (tasks).

Project monitoring is executed in weekly tutorial meetings.

QA 04 – User Interface Specification Standards

It has 2 parts

User interface specification has two parts

1. Document describing the main use cases of the system
2. Online presentation showing the user what happens in each of the use cases.

The document (**1**) has to cover the following sections:

- Typical Users
- Use cases
- Error conditions

Typical Users.

The document needs to address each set of users differently. In some software there are different types of users. E.g. moderator, subscriber, etc. (This section will be shorter in our case)

Use Cases

Here, each of the user type's tasks need to be identified. Each of the cases should be given a reference number. Each of these should give a clear description of the whole process of what happens in this case. The descriptions should be brief - from how to initiate it to how to complete the process.

Error conditions

The online presentation (**2**) should give a visual representation of how the final product will look without the expense of having everything done. Power point should be used for this.

QA 05 –Data Specification Standards

A. **Outline Structure**

The outline structure can be anything as long as it is according to QA 02.A table is given to us as example.

B. **Decomposition Description**

It records the division of software and modules that structures to a whole system.

These should be referenced and they might be:

- Programs in system
- Significant classes
- Table mapping requirements onto cases

C. **Dependency description**

It describes the dependencies and dependency relationships between modules. Appropriate way is using UML Component Diagrams. Each section should be referenced as well.

D. **Interface description**

It should give all information needed to programmers that maintain the software about the facilities in each module. It should have an outline specification of the system. It should include the name and type of the class and the classes which it extends and description why. The simplest way of doing so is to follow the guideline of QA 09 –Java coding standards.

E. **Detailed design**

UML Sequence diagrams are good way of showing how the classes work together for the major operations of the program.

Textual representation of what the algorithms in the program do is also needed.

Significant data structures occur when many objects are linked together in a more complex structure. Class diagrams showing the entity relationships between classes should be made. Also object diagrams that show static relationships in class diagrams in real time (for the object diagrams' purpose). If there is a data transfer between programs it needs to be described.

QA 06 –Test procedure standards

General approach to testing

Testing cannot prove the absence of errors but it can demonstrate their presence. Correction of errors should be left until all of the tests specified have been conducted. If errors are found then we need to follow the *specified problem reporting* and *change control procedures* (QA 08). This is known as *regression testing*.

It is important the emphasis on testing boundary situations. Error conditions should also be tested.

The software we are building will be subjected to 3 levels of testing – module, system and acceptance. The programmer is responsible for *module testing*. Ideally it needs to be done before designing and writing the code.

System testing involves integrating all the modules together to form a complete system and to be able to execute it. It is better practice to carry these tests out using a person not linked with the developing of this or any software.

Acceptance testing involves tests done by the Client exercising the entire system before agreeing to accept the product.

Test Plan

Normally a plan is done outlining what testing needs to be done and when. Module testing should be left to the coder and system testing should be done by writing a system testing specification during the design phase. When it is finished, a Test Report will documents any features that work incorrectly.

Test specifications

Its purpose is to specify in detail each of system tests to be executed. The test spec must be according the Requirements specification for each feature being tested. It should provide a set of reproducible actions to test all the main functionality of the system.

Each test spec has to have introductory section and a collection of test procedures. The introductory section has to match QA 02. An example table is given to us in the QA document.

Test result reporting

Details of the test results must be maintained in a test rep folder in the project repository. It must have two sections –*Module tests* and *System tests*.

A date report must be added whenever a new version of the software is built saying which tests fail.

QA 07 –Review Standards

The scope is the detection of problems but not correction of them. The problems have to be formally recorded.

Selecting a review team and arranging a meeting

The team should always include the project leader, the QA manager and the person(s) who wrote the item to be reviewed. The review meeting that is arranged should be up to 2 hours. The QA manager is responsible for finding a location for the meeting and to give notice for the meeting at least 1 week before.

Distribution of relevant documents

All documents must be given to the review team by the QA manager in advance of the review meeting. The following documents should be brought along with other appropriate documents:

- for a user interface spec review, requirements spec and the user interface specification must be available
- for a design spec review, the requirements specification and design spec
- for test spec review, all these 2 + test spec
- for software verification review the design spec and the code must be available

Distribution of these docs can be done electronically since we have to keep a folder with all documents in our git hub repositories. The QA manager's responsibility is to send to all.

Conduct of the review

It will be chaired by the Project Leader.

QA manager is responsible for recording problems and actions. All other review team members should also contribute to the meeting.

Appendix A –Checklist for all documents

- all formal project documents have to have some necessarily information on the front cover
- Header of the document must contain title, version and status on each page.
- Footer must contain *Aberystwyth University / Computer Science* phrase and has to be paged correctly
- The document must contain some necessarily section specified in the QA document
- These sections must be numbered correctly from 1
- Are the fonts appropriate in headers and in texts?

Appendix B –Design specification review

The goal here is to determine whether the design covers all the client's requirements.

Appendix C –Test specification review

The goal is to decide if the testing covers the system at that level so to be said that it meets the client's requirements

Appendix D –Software verification review

The goal is to check whether the code implements all aspects of the design as described in the design specification.

For bot appendix B and C there are questions to ask specified in the QA document.

QA 08 –Operating procedures and Configuration Management Standards

Software Configuration management

Configuration management is the management and control of all the changes in the system so that the state of each component is always known. Test specification and the User Interface document must be kept under control of configuration management system.

It will allow user to

1. Retrieve copies of any version of a *file,* enabling recovery of previous or old versions.
2. Retrieve copies of any version of directory structure.
3. Check in changes to the file.
4. Inquire about differences between versions, obtain a log summarising the changes checked in for a particular version and produce a history of file showing all changes and the user responsible.

If the following items are produced they must be kept under control of the configuration management system:

1. List of project deliverables
2. Requirements Specification
3. Design Spec
4. Test Spec
5. User Interface document
6. Maintenance Manual
7. End-of-Project Reports
8. Source code and tools (e.g. build files)

The QA manager will be responsible for adherence to the configuration management procedures.

Configuration items are project items which are controlled by the configuration management system.

QA manager is responsible of the allocation of configuration references and for the maintenance of a file *config_refs* in the configuration directory.

Any document must have some of these statuses: *Draft, for review, Release*

The Structure of directories.

One aspect of config management is the *standardisation* of the directory structure. This structure is managed by the version control system.

QA manager ensures that an initial structure is created. The following directories should be present + additional if required:

Docs –contains submitted documents produced by the group

Man –management documents are stored in this directory

Src –includes the source code for the project

Dev –This directory should contain a folder of the date of each tutorial. If student creates a draft document or some code as part of their duties for that week then it should be submitted to this folder.

*IDEs will typically manage file directories at an appropriate level for including your emerging product in **src.** For example, NetBeans will use a directory called **nbproject.** Eclipse will have a number of "dot" files. These must be under version control. Unit test code will be in the **src** directory. The IDE will normally manage this. Directories generated by the IDE from source must not be under version control.*

Managing documents

Keep a proper status level for each document. When updating a document the responsible must ensure that they have the correct version. Major changes must take place through version control branch.

Managing minutes of meeting

They will be stored in the **man** minutes directory. There must be only 1 file per meeting and it has to have the date text plus any file extension where needed

Problem reporting and corrective action

Problems relating to items in **docs** directory must go through a formal problem reporting and action process.

A form called Change Control Form is necessary to record the progress of changing released documents. There is an example of this CCF document in GPDocs. Completed forms should be kept in a ring binder know as Change File. This binder must contain dividers with these headings: ***Outstanding CCFs, CCFs Dealt with****.* The *change file* should be maintained by the Project leader. The Quality assurance manager must make sure these files are completed and used correctly. The following is a problem reporting and corrective action procedure:

- When we discover a problem in a configuration item that has been allocated a vs number and has a status of *draft, or release,* the details of the problem must be noted on a Change control Form
- The CCF must be given to the QA Manager, who will assign a unique CCF number to the form and who will then investigate the problem with the aid of other team members as appropriate.
- When a change has been completed whoever made the change must report to the QA manager. Then QA manager may initiate formal testing to confirm the change fixed the problem.
- If the change is accepted by the QA manager, the configuration item must have its version number incremented. The CCF corresponding to the reported problem will be then moved from the ***Outstanding CCFs*** to ***CCFs Dealt With****.*
- If the change is not accepted, then the QA manager will advise on further action if necessary to complete a new CCF.

Project teams can choose to run all reporting and corrective action online, In order to do so they have to:

- Create a directory *CCFs* at the top level. A copy of each raised CCF should be kept in the CCFs directory. The first should be named CCF001.doc.
- Two files must be maintained by the Project Leader in the CONFIG directory. These files are called *Outstanding_CCFs* and *CCFs_Dealt_With.*

QA 09 –Java coding standards

Introduction

Specifies standards for writing Java software on departmental group projects.

Any files with Java-associated text which is not the language e.g. configuration files in XML are not included in this document.

The document has a set of rules and guidelines which should be followed when writing Java programs. A template for classes is provided in appendix A.

The document covers the following aspects of writing Java:

- Code organisation: strategies for organising and naming packages
- Identifier Naming Conventions: rules for naming identifiers to make clear type and purpose
- Class organisation : rules for how to arrange the parts of a class to make them easier to read
- Comments: rules for content of mandatory comments in programs
- Identation: how to set out Java code for the group project
- Language features: rules and guidelines for use or avoidance of some of the features of Java.

Code organisation

Java allows of the organisation of related classes into packages. By convention a company's package names should start with the reversed domain-name of the company. In our case all packages should start with *uk.ac.aber.cs221.<groupname>. The package identifiers should be in lower-case without underscores.* For general purpose classes we follow the naming convention used in the implementation of Java itself. Also packages in Java are hierarchical. If there is a subset of related classes within a package, it often makes sense to create a new sub-package from them.

Each new application should be given a new package. This package should contain the top-level application class, and any other classes that may be of use in other code.

Objects, not specific to a particular application should be in a separate package. For example, a Diary class might be associated with the heating Control application.

It is important to maintain a clear package structure. It is easier to find classes and the risk of duplication is reduced.

Identifier Naming Conventions

All identifiers should use the U.S spelling. The reason is for consistency with external libraries.

For names, try to apply following guidelines.

- Names that are as much self- documenting as possible.
- Try to make them real worlds object names.
- Use predicate clauses or adjectives for Boolean objects.
- Use action verbs for procedures and entries
- Use constants rather than variables for constant values.

Class and interface names start with capital letter. (Camel case)

When the word in the class would be upper-case only the first letter should be a capital letter.

Methods and *variable* names start with lower-case letter and use capitals to separate words. The naming of methods should follow the JavaBeans convention. Indexed properties should normally have get and set methods that allow you to access individual values.

Constants (static final variables) follow the same conventions that are in the normal variables.

Class organisation

Every class should have its methods and variables arranged into groups and preceded by a comment. This is so we can group the related methods together. Structure of a class should be following that shown on Appendix A.

Inner classes may be used to break up the complexity of large classes. They are also useful when creating GUIs with nested panels.

Anonymous classes should only be used to pass simple implementations of an interface as parameters to a method.

Comments

The commenting style is driven by the requirements of Javadoc.

Each file should have a simple header giving the filename , a copyright message and the version and date e.g. :

/*

*@(#)SomeClass.java 1.1 2016/10/18

*

*Copyright(c) 2016 Abertystwyth University

*All rights reserved

*

*/

Each class or interface should have a standard Javadoc class header. Note that:

- Description should provide an overview of the class, but doesn't go to a great detail.
- Description should be separated by the tags with an empty line
- An @author tag must be included for each author.
- A @version tag must be included for each version of the file.(except inner-classes)
- Anonymous classes do not need headers.

Each Method should contain a Javadoc header. Note that:

- Description should never cover the purpose of the method, and any side-effects.
- All parameters and return values should have @param or @return tags, even if they seem obvious. This helps give the resulting documentation a more complete feel.
- Tags of the same type should be lined up (all with @param tags)
- Every type of exception thrown by the method should have an @exception tag
- @see tags should be used to cross-reference related methods or classes
- Methods in anonymous classes do not always need headers
- Methods in skeletal test classes do not always need headers

Block comments are used to describe a group of related code. Most block comments should be one line, but if more than one line comment is needed, the extra lines should each begin with the double slash. Block comments should also be indented to match the indentation of the line of code following it. A single blank line should precede the comment and the block of code should follow immediately after.

It is often useful to put comments before control structures (loops, ifs, whiles, etc.) to explain the purpose of the code in the blocks that follow.

Indentation

The standard unit if indentation is three spaces. Watch out because if we use tabs, the tabs can be mapped with more than three spaces.

The first line of a class or interface should declare the name of the class and its parent. If a class implements an interface, this should be declared on the following line.

The first line of a method should declare the return type, name and parameters of the method. If the method throws any checked exceptions, these should be declared on the following line.

Language Features

Nested assignments

No nested assignment. It is possible to write expression like a = b+ (c=d*e) in java, where both a, and c are given a value. This saves very little, and makes the code less clear. We want to avoid it.

Exceptions

Exceptions should only ever be used for exceptional circumstances –never as a means of communicating the result of a method. Exceptions used in this way can confuse the flow of control in the code. Where exceptions are needed, methods should always throw exception of an appropriate class. If such a class does not exist, a new one should be defined.

Method Overloading

A class will often provide a number of overloaded methods. The only restriction on this, is that the overloaded methods must all perform the same task.

QA 10 –Producing a Final Report

Introduction

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

- The end-of-project report
- The project test report
- The project maintenance manual report
- The personal reflective report
- Also gives a checklist for other documents and code which should be included.

Extra Documents Produced for the Final Delivery

The End-Of-Project Report

This document should state clearly how much has been accomplished on the project. Typically the report will be written by the Project Leader with contributions from the rest of the team. It should be written as a coherent submission, and should provide the reader with the following items:

A management summary: This should sum up in one page what the project achieved. It should not take more than one page of A4.

A historical account of the project: This should outline the main events over the lifetime of the project. It should take no more than 2 a4 pages.

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 good to be as accurate as possible. Also it needs to address any missing features.

Performance of each team member. The project leader should write a description of up to half a page of the duties and performance of each group member, including himself.

Critical evaluation of the team and the project: No more than a page and should address the following:

- How did the team perform as a whole and how could that have been improved?
- How could the project hat was set for you have been improved.
- What were the most important lessons learned about software projects and about working in teams?

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.

The document should report on both the results for the tests in your test plan, and the acceptance tests given by the client. This report should have a standard document header page, but does not need *purpose/scope/objectives.* The body of the document can just be two tables of tests.

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 installer and maintainers of your software which part of the program source is likely to provide the answers.

Most programs contain bugs. It is almost impossible to develop a software without any bugs. Normal commercial practice especially with minor bugs is to document them and correct them at a convenient time, rather than rushing to fix them and releasing a new version immediately. Making immediate fixes is costly in distribution and reinstallation of a new versions.

A checklist for the structure of a maintenance manual is:

Program description – gives a brief description of what the program does.

Program structure – Describes the design of the program

Algorithms –Description in detail of the significant algorithms used in the program.

The main data areas – It specifies the data structures.

Files - It may be that the program accesses certain fixed files or needs files of a certain type to be available.

Interfaces –Many programs control or read devices such as measuring instruments.

Suggestions for improvements –Most programs are a compromise between what one would like to do and what one has time to do. It is worth giving suggestion for future improvement of the software if it wasn't possible to do it at the developing time because of various problems.

Things to watch for when making changes – It is desirable to make only changes or to be careful when making changes to the software. The developer must be aware that when they make changes these changes shouldn't change any other working part of the program.

Physical limitations of the program – A computer installation is a finite environment. It can only have so much memory. Some programs will come against these constraints. It is important to list the requirements, where known, because all environments impose the same constraints.

Rebuilding and testing - Maintainers need to know what to do when rebuilding the program. What is the location of all the files? What should be done in order to rebuild the software. How do they add a test when a new problem is discovered. If documents are not in standard format, then it is worth to describe how to build them.

Feedback will have been given on the documents delivered before Easter (for user interface plan, the test plan and the design specification.) Groups are expected to use the feedback along with experience of trying to implement their design to produced improved version of the three documents.

Structure of the Final Delivery

Reports: All of the new reports plus the revised reports should be submitted for the group on Blackboard:

- The End-of-Project Report
- The Project Test Report
- The Project Maintenance Manual
- The User Interface Hand-In
- The Test Plan
- The Design Specification

All documents should be provided in a format which can be examined on screen. MS Word is acceptable.

The group project team should make sure that the state of their project repository at the end of Coding Week is such that it contains a working version of the project, and that it is accessible and easy to run. The client will meet with the team on the following week for demonstration of the software.

Personal Reflective report

In addition to previous documentations, each person must submit a personal reflective document. It should review the roles and tasks undertaken by the student during the project. Here individuals may include 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. This report should not be more than 2 pages A4. This report should be submitted in blackboard. A link to the student’s blog should also be included, and the blog should be kept available until the end of May.