Department of CSE, BUET

QMS Support for the Developer team

Software Quality Assurance

Md. Tanvir Al Amin

Abstract

QMS (Quality Management System) describes a process which ensures and demonstrates the quality of the products and services provided by an organization. It is a set of procedures forming the basis for executing organization's product/service delivery mechanisms. These procedures are the documents telling staffs how to follow a quality system in their daily work. Thus QMS is the management's means to establish a uniform and consistent approach to product realization or service delivery. In this report summarize the relationship between QMS and the developer team in terms of cooperation and collaboration. We emphasize on what support a developer can expect to get from QMS.

Introduction

A number of users within and outside of the organization uses the quality management system. In a software development organization, there are people like Senior Management, Project Manager, Programmers, Application System Designers, Analysts, and Staffs, Testers and Marketing Department. Each of them has certain expectations from the quality management system deployed in the organization.

After the design phase of software project is complete, programmer or a developer is given the specification (input, output, parameters) for a functional module or a class. The developer's job is then to convert it to program code. And this code is taken through unit testing. Most programmers also carry out modification to the code as requirements keep evolving.

Responsibility of the developer team:

The development team is responsible for:

1. Reviewing and commenting on the SQA Plan for the specific project.
2. Implementing the quality program in accordance with the SQA Plan.
3. Resolving and following-up on any quality issues raised by SQA related to software design and development.
4. Identifying, implementing, and evaluating the quality factors to be implemented in the software.
5. Implementing the software design/development practices, processes, and procedures as defined in references and other program/project planning documents.

Expectation of the developer team

The developer teams can expect from QMS a set of standards or best practices as described in this article.

Coding Standard

Setting a coding standard is very important. Because, use of coding standards makes testing and debugging faster. The developers should expect a consistent and logically intuitive coding standard for the QMS team. Use of coding standards, guidelines makes testing and debugging easier.

Many source code programming style guides, which often stress readability and usually language-specific conventions are aimed at reducing the cost of source code maintenance. Some of the issues that affect code quality include:

- Readability
- Ease of maintenance, testing, debugging, fixing, modification and portability
- Low complexity
- Low resource consumption: memory, CPU

- Number of compilation or lint warnings
- Robust input validation and error handling, established by software fault injection

Some coding standards widely followed are :

- The organization may require that all helper class names start with H.
- All class names should follow ThisIsAClass naming convention and database tables should follow this_is_a_table convention, i.e. first letters of different words of a class name are capitalized and there are no underscore, and different words of a table name are separated by underscore and there are no capital letters.
- Variables should be named in Hungarian notation. An integer variable should have a prefix 'n', a Boolean variable should have prefix 'b' or a prefix 'is'. String variables should start with str. So, nLength is automatically perceived to be an integer, isAvailable is a boolean variable, strText is a string.
- Variables should be named according to their specific function. For example names like x, y, z, i, j, m, n should be limited to temporary variables or iterators and name of the variables representing a major task should have some semantics.
- All constant names should be capitalized. Example : DEFAULT_SOCKET
- Each routine/function name should describe exactly what the routine/function does.
- Explicitly comment variables changed out of the normal control flow or other code likely to break during maintenance. Embedded keywords are used to point out issues and potential problems. Consider a robot will parse your comments looking for keywords, stripping them out, and making a report so people can make a special effort where needed.

Gotcha Keywords

- **:TODO: topic**
  Means there's more to do here, don't forget.

- **:BUG: [bugid] topic**
  means there's a Known bug here, explain it and optionally give a bug ID.

- **:KLUDGE:**
  When you've done something ugly say so and explain how you would do it differently next time if you had more time.

- **:TRICKY:**
  Tells somebody that the following code is very tricky so don't go changing it without thinking.

- **:WARNING:**
  Beware of something.

- **:COMPILER:**
  Sometimes you need to work around a compiler problem. Document it. The problem may go away eventually.

- **:ATTRIBUTE: value**
  The general form of an attribute embedded in a comment. You can make up your own attributes and they'll be extracted.

A sample document of coding standard is included in Appendix A.

Test Unit

The QMS should provide directions to the developer to store away test data and test outcomes in files. This way, re-programming becomes easier. The developer can retrieve test data, modify it in response to the requirements change and then apply it again. A sample document of coding standard is included in Appendix C.

Complexity Management

Layering is the primary technique for reducing complexity in a system. A system should be divided into layers. Layers should communicate between adjacent layers using well defined interfaces. When a layer uses a non-adjacent layer then a layering violation has occurred.

> A layering violation simply means that there is dependency between layers that is not controlled by a well defined interface. When one of the layers change code could break.

Repository

The QMS should provide a standard that insists that the names of the software chunks, which have been produced by a developer, have a correspondence to the files used to store the source code, object code, test data and the test outcomes for the software. In other words, the QMS should guide the developers and the configuration manager towards organizing the entire software project repository. Typically the items in a project repository are the following:

- Software Requirements Specification Document (SRS document)
- Design/Application Architecture Document.
- Change Requests
- Coding Standards
- Quality Documents
- Test procedures
- Test cases and Test Scripts
- Test logs
- Source Files
- Binary Files
- Model files, Property Files

Conclusion

SQA guidelines help to organize the entire project repository. Use of tools and update tools are highly recommended by SQA. The QMS should provide a consistent standard and guideline for versioning management also. A sample development folder is attached in Appendix B

BIBLIOGRAPHY

[MAGU93] Steve Maguire, Writing Solid Code, Microsoft Press, Redmond, WA, 1993.

[MCCO93] Steve McConnell, Code Complete, Microsoft Press, Redmond, WA, 1993.

[NINA04] Nina S. Godbole, Software Quality Assurance, Narosha Publishing House, Kolkata, India

[C++CodStan] C++ Coding Standard, URL : http://www.possibility.com/Cpp/CppCodingStandard.html

[JavaCodStan] Java Coding Standard, http://www.geosoft.no/development/javastyle.html
[C++BP] Herb Sutter, Andrei Alexandrescu C++ Coding Standards : Rules, Guidelines, and Best Practices

[Goddard] Information and tools for Software Assurance practitioners in the NASA community URL: http://sw assurance.gsfc.nasa.gov/disciplines/quality/index.php Goddard Space Flight Center

Appendix A: Sample QMS Coding Standard

CODING STANDARDS

This document is the Coding Guideline document for the generic maintenance process architecture for the XYZ Corporation project Alpha. This document describes the basic conventions and coding guidelines to be used for the project. The scope is coding style and not functional organization. Coders should follow these conventions to the maximum extent possible in order to ensure a uniform appearance and improve maintainability.

1.0 COMMENTING TECHNIQUE CHECKLIST

1.1 GENERAL

- Does the source listing contain most of the information about the module?
- Can someone pick up the code and immediately start to understand it?
- Do comments explain the code's intent or summarize what the code does, rather than just repeating the code?
- Has tricky code been rewritten rather than commented?
- Are comments up to date?
- Are comments clear and correct?
- Does the commenting style allow comments to be easily modified?

1.2 STATEMENTS AND PARAGRAPHS

- Does the code avoid endline comments?
- Do comments focus on why rather than how?
- Do comments prepare the reader for the code to follow?
- Are surprises documented?
- Have abbreviations been avoided?
- Is the distinction between major and minor comments clear?
- Is the code that works around an error or undocumented feature commented?

1.3 DATA DECLARATIONS

- Are units on data declarations commented?
- Are the ranges of values on numeric data commented?
- Are coded meanings commented?
- Are limitations on input data commented?

- Are flags documented to the bit level?
- Has each global variable been commented where it is declared?
- Are magic numbers documented or, preferably, replaced with named constants or variables?

1.4 CONTROL STRUCTURES

- Is each control statement commented?
- Are the ends of long or complex control structures commented?

1.5 ROUTINES

- Is the purpose of each routine commented?
- Are other facts about each routine given in comments, when relevant, including input and output data, interface assumptions, limitations, error corrections, global effects, and sources of algorithms?

2.0 NAMING CONVENTION CHECKLIST

2.1 GENERAL NAMING CONSIDERATIONS

- Does the name fully and accurately describe what the variable represents?
- Does the name refer to the real-world problem rather than to the programming language solution?
- Is the name long enough that you don't have to puzzle it out?

2.2 NAMING SPECIFIC KINDS OF DATA

- Are loop index names meaningful (something other than i, j, or k if the loop is more than one or two lines long or is nested)?
- Have all "temporary" variables been renamed to something more meaningful?
- Are boolean variables named so that their meanings when they are True are clear?
- Do enumerated-type names include a prefix or suffix that indicates the category - for example, Color for ColorRed, ColorGreen, ColorBlue, and so on?
- Are named constants named for their abstract entities they represent rather than the numbers they refer to?

2.3 NAMING CONVENTIONS

- Does the convention distinguish among local, module, and global data?
- Does the convention distinguish among type names, named constants, enumerated types, and variables?
- Does the convention identify input-only parameters to routines in languages that don't enforce them?
- Is the convention as compatible as possible with standard conventions for the language?
- Are names formatted for readability?

2.4 SHORT NAMES

- Does the code use long names (unless it's necessary to use short ones)?
- Does the code avoid abbreviations that save only one character?
- Are all words abbreviated consistently?
- Are the names pronounceable?
- Are names that could be mispronounced avoided?

2.5 COMMON NAMING PROBLEMS: HAVE YOU AVOIDED...

- ...names that are misleading?
- ...names with similar meanings?
- ...names that are different by only one or two characters?
- ...names that sound similar?
- ...names intentionally misspelled to make them shorter?
- ...names that are commonly misspelled in English?
- ...names that conflict with standard library-routine names or with predefined variable names? Note: Overloading is permitted and sometimes encouraged in Ada.

3.0 LAYOUT CHECKLIST

3.1 GENERAL

- Is formatting done primarily to illuminate the logical structure of the code?
- Can the formatting scheme be used consistently?
- Does the formatting scheme result in code that's easy to maintain?
- Does the formatting scheme improve code readability?
- Is the formatting consistent when viewed with a text editor in both UNIX and MS-DOS?
- Have all tabs been eliminated?

3.2 CONTROL STRUCTURES

- Does the code avoid double indented begin-end ({ }) pairs?
- Are complicated expressions formatted for readability?
- Are single-statement blocks formatted consistently?
- Are case statements formatted in a way that's consistent with the formatting of other control structures?

3.3 INDIVIDUAL STATEMENTS

- Are continuation lines indented sensibly?
- Are groups of related statements aligned?
- Are groups of unrelated statements unaligned?
- Does each line contain at most one statement?
- Is there at most one data declaration per line?

3.4 COMMENTS

- Are the comments indented the same number of spaces as the code they comment?
- Is the commenting style easy to maintain?

3.5 ROUTINES

- Are the arguments to each routine formatted so that each argument is easy to read, modify, and comment?
- In C, are new-style routine declarations used? (Compiler dependant)

3.6 FILES, MODULES, AND PROGRAMS

- Does each file hold code for one and only one module?
- Are routines within a file clearly separated with blank lines?
- In Ada, avoid using the "use" clause which creates naming ambiguities.

4.0 SELF-DOCUMENTING CODE CHECKLIST

4.1 ROUTINES

- Does each routine's name describe exactly what the routine does?
- Does each routine perform one well-defined task?

- Have all parts of each routine that would benefit from being put into their own routines been put into their own routines?
- Is each routine's interface obvious and clear?

4.2 DATA ORGANIZATION

- Are extra variables used for clarity when needed?
- Are references to variables close together?
- Are data structures simple so that they minimize complexity?
- Is complicated data accessed through abstract access routines (abstract data types)?

4.3 CONTROL

- Is the nominal path through the code clear?
- Are related statements grouped together?
- Have relatively independent groups of statements been packaged into their own routines?
- Does the normal case follow the if rather than the else?
- Are control structures simple so that they minimize complexity?
- Does each loop perform one and only one function, as a well-defined routine would?
- Is nesting minimized?
- Have boolean expressions been simplified by using additional boolean variables, boolean functions, and decision tables?

4.4 DESIGN

- Is the code straightforward, and does it avoid cleverness?
- Are implementation details hidden as much as possible?
- Is the program written in terms of the problem domain as much as possible rather than in terms of computer-science or programming language structures?

Unit Header Format

```
Unit(including main)

/*********************************************************

Module Name: The name of the unit being documented

Program: The name of the unit
```

```
Purpose: A detailed description explaining what the unit does and any
particular reason why a certain design was chosen.

Inputs: A list of inputs (parameters) to the unit and their purpose

Outputs: A list of outputs generated by the unit and their purpose

Date Created: The date the unit was created

Modified: List of dates and reasons the unit was modified

**********************************************************/
```

File Header Format

```
File Header

/*********************************************************

FILE: @(#) @(#)codestan.txt    3.4 - 08/04/2010

PURPOSE: This file consists of all functions that provide and process
         the command line interface for the Recon3 instrumentor for C/C++.

SYSTEM: Recon3

HISTORY:
VER     DATE           AUTHOR         DESCRIPTION
1.0     08 APR 10      MTA  Amin      Created for T001 - Add r3 command
                                      line interface
==========================================================================*/
*****************************************************/
```

Field Descriptions

```
FILE:  The name of the file and the SCCS keywords in this format.

PURPOSE: An overall description of why this file was created and what it
does.

SYSTEM: The software product that this file is a component of.

VER:  This field is used to record the SCCS version number this file will
have when you check it in.

DATE: The date the file was created in the form DD MMM YY.
AUTHOR:  This is the first initial and last name of the person making the
change.

DESCRIPTION:  A brief explanation of why the file was created or modified.
```

Appendix B: Sample QMS Software Development Folder

Date(s) of Assessment: _______________ Project: _______________________________________

Assessor(s): _________________________ Process Assessed: ______________________________

____

		Y, N, NA	F, O	Comments
ASSESSMENT PREPARATION				
1	Have standards been identified to clearly define the process assessment?			
2	Were guidelines used to prepare for the assessment?			
3	Has the project submitted any request for deviations or waivers to the defined process?			
4	Have entrance and exit criteria been established for the assessment?			
5	Were the appropriate stakeholders identified for this SDF assessment?			
6	Was the assessment process addressed, including the method for capturing Requests for Action (RFAs), risks, or issues?			

SOFTWARE DEVELOPMENT FOLDER (SDF) CONTENT				
7	Does the SDF list or reference all software requirements that are mapped to the software element?			

8	Does the SDF include:			
8a	All applicable Requirements Documents?			
8b	An updated Requirements Matrix?			
8c	Functional specification(s)?			
8d	Interface definitions?			
8e	Data structure definitions?			
9	Are all action items (RFAs/RIDs) resulting from a milestone review (e.g., SRR, PDR, and CDR) that affect this software element or its requirements maintained in the SDF?			
10	Was the documentation reflecting the (RFAs/RIDs) action item's resolution provided in the SDF?			
11	Was the applicable milestone review package identified in the SDF?			
12	Was the design inspection/peer review package(s) for this software element inserted (or referenced) in the SDF?			
13	Were the resulting (design/peer review) action items and documentation of their resolution included in the SDF?			
14	Are there code inspection/peer review package(s) for each software element?			

15	Were the resulting (code inspection/peer review) action items and documentation of their resolution included in the SDF?			
16	Were the following items located in the SDF:			
16a	Current listing(s) for the each software element?			
16b	PDL (Program Design Language as applicable)?			
16c	S/W Change History?			
16d	Compiled Source Code?			
17	Are specific tools identified that are required to maintain each software element: (e.g., one-of-a-kind compilers or commercial/government developed tools necessary to recompile, update, or execute the software)?			
18	Are the following items located or referenced in the SDF :			
18a	Unit test plans/procedures?			
18b	Test data and source code for any test drivers?			
18c	Summary of unit test results?			
18d	Discrepancy reports or change requests that necessitate modification of the software element?			

18e	Documentation of each discrepancy/change’s resolution?			
19	Was the date noted when the SDF was delivered to CM or otherwise archived (if applicable)?			
POST REVIEW ACTIVITIES				
20	At the conclusion of the assessment is an understanding reached on the validity and degree of completeness of the Development Folders?			
21	Did all designated parties concur in the acceptability of the Development Folders?			
22	Are there any risks, issues, or request for actions (RFAs) that require follow-up?			
23	Is there a process in place for reviewing and tracking the closure of risks, issues, or RFAs?			
24	Were Lessons Learned addressed and captured?			
REFERENCE ITEMS/DOCUMENTS				
580-CK-017-01, ISD Software Development Folder Checklist				

Date(s) of Assessment: _________________ Project: ___________________________________

Assessor(s): ___________________________ Process Assessed: __________________________

____

COMMENTS PAGE _____ of _____

#	Comments from assessment

Appendix C : Sample QMS Software Unit Test

Date(s) of Assessment: _______ Project: ________________________________

Assessor(s): _______________ Process Assessed: ________________________
____

		Y, N, NA	F, O	Comments
PROCESS ASSESSMENT PREPARATION				
1	Do standards and guidelines exist that clearly define the process?			
2	Has the project submitted any request for deviations or waivers to current standards or guidelines?			
4	Have entrance and exit criteria been established for the process assessment?			
5	Are processes documented and under configuration control?			
6	Was documentation required for the implementation of this process made available to the participants with ample time to review and prepare?			
7	Is there evidence that all stakeholders/participants were involved in the implementation of the process?			

		Y, N, NA	F, O	Comments
8	Have all parties involved in the implementation of the assessed process received training on the process?			
9	Were there any constraints/limitations associated with the implementation of the process identified?			
UNIT TEST CRITERIA\COMPLIANCE				
10	Were the objectives of the unit test established:			
10a	The strategies to be employed			
10b	The coverage requirements,			
10c	Reporting and analysis,			
10d	Close-out of anomalies?			
11	Has the unit test been designed to be a test that executes all of the code in the unit? *Tip: Is there evidence that the unit test executed every statement in the unit, including all branches of conditional statements?*			
12	Does the unit test satisfy the requirement for full path coverage and boundary value testing?			

		Y, N, NA	F, O	Comments
13	Is there sufficient documentation on the unit test to make it clear what is being tested and the general test approach?			
14	Has it been confirmed that anomalies during unit test are software anomalies, and not problems detected for other reasons?			
15	Have comments in the source code been paired with comments in the unit test code to verify that all conditional branches have been tested and paths have been covered?			
16	Was each conditional branch in the unit executed?			
17	Were all operations that might cause erroneous execution (i.e., divide by zero, taking square root of negative number, etc.) proved impossible?			
18	Were all parameters and inputs to subprograms tested with nominal values and with values at the extremes by the algorithm, compiler, and CPU?			
19	Were changes to the module source code required to run unit test?			
20	Is there documentation regarding the test environment the unit was tested on?			
21	Is the unit test repeatable, and will identical results be produced?			
22	Can the unit test be run automatically without user interaction?			

		Y, N, NA	F, O	Comments
23	Have the data files used by the unit test been treated as source code for the purpose of Configuration Management?			
24	Do distinct elements of input vectors and matrices have distinct values for the purpose of catching indexing errors?			
25	Do inputs have distinct values? (*If input's order to an operation matter, the input's should have distinct values to catch order errors.)*			
POST ASSESSMENT ACTIVITIES				
26	Are unit tests and test results stored in the software development folders or other artifact files?			
27	At the conclusion of the assessment, is an understanding reached between development, test, system engineering, QA, & CM on the validity and degree of completeness of the Unit Test process?			
28	Did all designated parties concur in the acceptability of the Unit Test process (i.e., was there a legitimate reason to deviate from the process)?			
29	Have all artifacts been placed under formal configuration control (e.g., unit test results, unit test logs)?			
30	Were Lessons Learned addressed and captured?			
REFERENCE ITEMS/DOCUMENTS				
FSW Unit Test Standard, Flight Software Branch-Code 582, Version 1.03 – 09-25-03, 582-2000-002				

	Y, N, NA	F, O	Comments
National Institute of Standards and Technology (NIST) Special Publication 500-223, A Framework for the Development and Assurance of High Integrity Software, dtd12/94			
Mil-Std-498 DID			
***NPR 7150.2,* NASA Software Engineering Requirements (SWE-062)**			

Date(s) of Assessment: _________________ Project: __________________________________

Assessor(s): _______________________ Process Assessed: ___________________________

____

COMMENTS PAGE _____ of _____

#	Comments from assessment

Appendix D: Sample QMS Configuration Management

Date(s) of Assessment: _______________ Project: ______________________________________

Assessor(s): ________________________ Process Assessed: _____________________________

____

		Y, N, NA	F, O	Comments
ASSESSMENT PREPARATION				
1	Have standards been identified to clearly define the process being assessed?			
2	Were guidelines used to prepare for the assessment?			
3	Has the project submitted any request for deviations or waivers to the defined process?			
4	Have entrance and exit criteria been established for the assessment?			
5	Were the appropriate stakeholders involved in the implementation of this process?			
CONFIGURATION MANAGEMENT INFRASTRUCTURE				
6	Has the project identified those persons or groups with authority to approve baselines and authorize changes?			
7	Has a CM Plan been established?			
8	Have CM Procedures been developed to implement the plan?			
9	Has a CCB been established to approve all formal baselines and modifications that affect configured software products?			
10	Has a centralized software library been established for retention and controlled retrieval of software support documentation and change control records?			
11	Have CM tools been identified to manage the project's software work products?			

		Y, N, NA	F, O	Comments
12	Have CM training requirements been identified?			
13	Is there evidence of higher level management review of the CM process?			
14	Has the project obtained commitment from the stakeholders responsible for performing and supporting the CMP's execution?			
CONFIGURATION IDENTIFICATION				
15	Has the project identified all configuration items (CI) and related work products to be placed under configuration control?			
16	Does the CM process identify product baselines for major stages in the project's life cycle (e.g., requirements baseline, design baseline, build/release baselines, acceptance baselines)?			
17	Does the CM process assign unique identifiers (i.e., naming and labeling conventions) to configuration items?			
18	Does the CM process specify when each configuration item is placed under configuration management?			
19	Does the CM process identify each configuration item owner?			
CONFIGURATION CONTROL AND BASELINE MANAGEMENT				
20	Is there evidence that the CM Process establishes and maintains a configuration management and change management system for controlling software work products? If so, is the CM & change management system(s) capable of the following:			
20a	Managing multiple control levels of CM?			
20b	Storing and retrieving configuration items?			
20c	Storing and recovering archived versions of configuration items?			
20d	Maintaining the accuracy and integrity of the configuration items?			

		Y, N, NA	F, O	Comments
20e	Accepting change requests from all project members?			
21	Is there evidence that the CM system has created or released a baseline for internal use and/or delivery to the customer?			
22	Is there evidence of CCB authorization for creating, modifying, or releasing a baseline?			
23	Is there evidence that the impact of changes and proposed fixes are analyzed and recorded?			
24	Are backups made to ensure retrieval of previous baselines or work products?			
25	Is there off-line storage of master copies of software media, software documents, etc.?			
26	Does the Software Library provide controlled access and distribution of software configuration items and work products to authorized persons only?			
CONFIGURATION STATUS ACCOUNTING				
27	Does the CM system maintain records of the status and contents of the software throughout the project’s life cycle?			
28	Does the CM system record and monitor all change requests (and the reasons for change) to controlled software products to assure that the configuration of all identified items is known at all times?			
29	Does the CM system track the status of change requests to closure?			
30	Does the CM system report the status of approved configuration items and the status of approved changes?			
31	Can the CM system report the latest version of the baselines?			
32	Are CM records readily available to affected groups and/or individuals?			
CONFIGURATION AUDITS				
33	Is there evidence that configuration audits have been performed?			

		Y, N, NA	F, O	Comments
34	Do these CM audits confirm that the configuration records correctly identify the configuration items in the baseline?			
35	Do the CM audits confirm the completeness and correctness of items in the CM system?			
36	Do the CM audits confirm compliance with applicable CM standards & procedures?			
37	Are CM audit reports/ records available?			
38	Is there evidence that action items from CM audits have been tracked to closure?			
MEASUREMENT				
39	Is there evidence of measures, measurement results, and improvement information derived from planning and performing the CM process to support the future use and improvement of the organization's processes and process assets?			
POST REVIEW ACTIVITIES				
40	At the conclusion of the assessment, is an understanding reached on the validity and degree of completeness of the CM Process?			
41	Did all designated parties concur in the acceptability of the CM Process?			
42	Are there any risks, issues, or request for actions (RFAs) that require follow-up?			
43	Is there a process in place for reviewing and tracking the closure of risks, issues, or RFAs?			
44	Were Lessons Learned addressed and captured?			

REFERENCE ITEMS/DOCUMENTS
CMMI Version 1.1, Guidelines for Process and Integration and Product Improvement
NPR 7120.5B, NASA Program and Project Management Processes and Requirements
ISD Software Configuration Management Process, 580-PC-019-01

Date(s) of Assessment: _________________ Project: ____

Assessor(s): ___________________________ Process Assessed: ________________________________

____

___

COMMENTS PAGE _____ of _____

#	**Comments from assessment**
