

Software Quality Assurance Plan  
Rate Adjustment by Managing Inflows (RAMI)  
Team M

Daniel Connor (connor)      Chris Edwards (caedwa)      Lars Yencken (lljy)  
Maya Muthuswamy (mayadm)      Rod Howard (rihoward)

May 27, 2003

Maintained By: Chris Edwards (caedwa@students.cs.mu.oz.au)  
Version: 2.01

**Abstract**

This document contains all the standards and procedures to be followed by Team M in the TCP flow control analysis project RAMI. These processes help to ensure the quality of the final product.

## Contents

<b>1 Purpose</b>	<b>6</b>
1.1 Purpose of this document . . . . .	6
1.2 Project aims and purpose . . . . .	6
1.3 Scope of this document . . . . .	6
1.4 Document Conventions . . . . .	6
1.5 People . . . . .	6
1.5.1 Client . . . . .	6
1.5.2 Team . . . . .	6
1.5.3 Supervisor . . . . .	7
1.5.4 External Reviewers/Auditors . . . . .	7
1.5.5 340 Reviewers . . . . .	7
1.5.6 341 Reviewers . . . . .	7
<b>2 Reference Documents</b>	<b>8</b>
2.1 Documents to be Produced . . . . .	8
2.2 Documents Used . . . . .	8
2.3 Glossary . . . . .	8
<b>3 Management</b>	<b>9</b>
3.1 Role Allocations . . . . .	9
3.2 Role Descriptions . . . . .	9
3.2.1 Supervisor . . . . .	9
3.2.2 Project Manager . . . . .	10
3.2.3 Risk Manager . . . . .	10
3.2.4 Librarian . . . . .	10
3.2.5 Client Liaison Officer . . . . .	11
3.2.6 340/341 Liaison Officer . . . . .	11
3.2.7 Web Site Manager . . . . .	11
3.2.8 Backup Manager . . . . .	11
3.2.9 Meeting Chair . . . . .	11
3.2.10 Audit Manager . . . . .	12
3.2.11 Document Managers . . . . .	12
3.2.12 Technical Leader . . . . .	12
3.2.13 Test Manager . . . . .	12
3.2.14 Internal Review Manager . . . . .	12
3.2.15 Release Manager . . . . .	13
3.2.16 External Reviewers/Auditors . . . . .	13
3.2.17 340/341 Reviewers . . . . .	13
3.3 Project Plan . . . . .	13
3.3.1 Major Milestones: . . . . .	13
<b>4 Documentation</b>	<b>14</b>
4.1 Software Requirements Specification (SRS) . . . . .	14
4.2 Software Architectural Design Description (SADD) . . . . .	14
4.3 Detailed Design Document (DDD) . . . . .	14
4.4 Test Plan (TP) . . . . .	14
4.5 Traceability Matrix . . . . .	15
4.6 User Documentation . . . . .	15
4.7 Design Notebook . . . . .	15

<b>5</b>	<b>Standards, Practices, &amp; Conventions</b>	<b>16</b>
5.1	Meeting Procedure . . . . .	16
5.1.1	General Meeting Procedures: applicable to all meetings . . . . .	16
5.1.2	Additional Procedures for Team Meetings . . . . .	16
5.1.3	Additional Procedures for Client Meetings . . . . .	17
5.1.4	Additional Procedures for Supervisor Meetings . . . . .	17
5.1.5	Informal Meeting Procedures . . . . .	18
5.2	Semester Break Procedure . . . . .	18
5.3	Review Procedure . . . . .	19
5.3.1	Entry Criteria . . . . .	19
5.3.2	Team Reviews . . . . .	19
5.3.3	Individual Reviews . . . . .	20
5.3.3.1	Posting a Change for Review . . . . .	20
5.3.3.2	Reviewing a Change . . . . .	21
5.3.4	Weekly Code Reviews . . . . .	21
5.3.5	External Reviews . . . . .	22
5.3.5.1	340 and 341 Reviews . . . . .	23
5.3.6	Exit Criteria . . . . .	23
5.4	Audit Procedure . . . . .	24
5.4.1	External Audits . . . . .	24
5.5	Task Allocation and Tracking . . . . .	25
5.5.1	Allocation Procedure . . . . .	25
5.5.2	Meetings . . . . .	25
5.5.3	Outside Meetings . . . . .	25
5.5.4	Progress on Tasks . . . . .	25
5.5.5	Missed Deadlines . . . . .	25
5.5.6	Tracking . . . . .	25
5.6	Decisions . . . . .	25
5.6.1	Recording Decisions . . . . .	25
5.6.2	Critical Appraisal . . . . .	26
5.7	Release Control . . . . .	27
5.8	Documentation Format Standard . . . . .	27
5.9	Coding Standard . . . . .	28
5.10	Channels of Communication . . . . .	28
5.11	Changes to the SQAP after External Review . . . . .	29
5.12	Changes to Roles . . . . .	29
5.13	Guidelines for professional development . . . . .	29
<b>6</b>	<b>Reviews and Audits</b>	<b>30</b>
6.1	Software Quality Assurance Plan (SQAP) . . . . .	30
6.2	Software Requirements Specification (SRS) . . . . .	31
6.3	Software Architectural Design Description (SADD) . . . . .	32
6.4	Test Plan (TP) . . . . .	32
6.5	Detailed Design Document (DDD) . . . . .	33
6.6	User Documentation . . . . .	33
<b>7</b>	<b>Verification, Validation and Testing</b>	<b>34</b>
7.1	Documents . . . . .	34
7.2	Code . . . . .	34
7.3	End Product . . . . .	34

<b>8</b>	<b>Problem Reporting &amp; Corrective Action</b>	<b>35</b>
8.1	Reporting of Errors . . . . .	35
8.1.1	Programming Errors . . . . .	35
8.1.2	Documentation Errors . . . . .	35
8.2	Corrective Action . . . . .	35
8.3	Bug Log . . . . .	35
<b>9</b>	<b>Tools, Techniques &amp; Methodologies</b>	<b>36</b>
9.1	Design Methodology . . . . .	36
9.2	Software . . . . .	36
9.3	Choice of language . . . . .	36
9.4	Tools for coding . . . . .	37
9.5	Tools for Diagrams . . . . .	37
9.6	Project Management . . . . .	37
9.6.1	Use of the Project Plan . . . . .	37
9.7	Version Control . . . . .	37
9.8	Records, Maintenance & Media Control . . . . .	38
9.8.1	Directory Structure . . . . .	38
9.8.2	Repository Structure . . . . .	39
9.9	File Naming Standards . . . . .	40
9.10	File Permission Standards . . . . .	40
9.11	Backup Procedure . . . . .	41
9.12	Task Log . . . . .	41
9.13	Team Web Site . . . . .	42
9.14	Email Standards . . . . .	42
<b>10</b>	<b>Risk Management</b>	<b>43</b>
10.1	Risks Identified . . . . .	43
10.1.1	High Probability Risks . . . . .	43
10.1.2	Medium Probability Risks . . . . .	43
10.1.3	Low Probability Risks . . . . .	43
10.2	Risk Mitigation . . . . .	43
10.2.1	Illness of Team Member(s) . . . . .	43
10.2.2	Inexperience with Tools . . . . .	43
10.2.3	Inexperience at Planning . . . . .	43
10.2.4	Design Not Producing Product that Meets Requirements . . . . .	44
10.2.5	Insufficient Technological Knowledge . . . . .	44
10.2.6	Difficulties Contacting Client . . . . .	44
10.2.7	Loss of Data . . . . .	44
10.3	Schedule Control . . . . .	44
10.4	Team Member Availability . . . . .	44
10.5	Changes to SRS after Client sign-off . . . . .	45
10.6	Technical Knowledge Risks . . . . .	45
10.7	Information from External Sources . . . . .	46
10.8	Conflict Resolution . . . . .	46
10.9	Risk Log . . . . .	46
10.10	Problem Log . . . . .	46
<b>A</b>	<b>Changes</b>	<b>47</b>

---

<b>B Coding Standards</b>	<b>51</b>
B.1 SAM Coding Standard . . . . .	51
B.1.1 File organisation . . . . .	51
B.1.2 Indentation . . . . .	51
B.1.3 Import statements . . . . .	51
B.1.4 Attribute initialisations . . . . .	51
B.1.5 Blank lines . . . . .	51
B.1.6 Whitespace . . . . .	51
B.1.7 Naming standards . . . . .	52
B.1.8 Comments . . . . .	52
B.1.9 Documentation strings . . . . .	52
B.1.10 Happydoc . . . . .	53
B.2 FCM Coding Standard . . . . .	53

# 1 Purpose

## 1.1 Purpose of this document

The purpose of the Software Quality Assurance Plan (SQAP) is to state the methods and procedures which will be used during the project to ensure the end product is of high quality, as well as identifying various people, documents, and tools that will be used during the project. The SQAP will be used primarily by the Team (see section 1.5.2) as a reference throughout the entire project. The Team will also update and refine it, to ensure the SQAP continues to reflect how the product is being developed (see the Changes Appendix, section A). The Supervisor (see section 1.5.3) and the External Auditors (see section 1.5.4) will also use the SQAP to make sure that the Team is following the processes specified and thereby gauge the quality of the project.

## 1.2 Project aims and purpose

The aim of this project is to develop the RAMI software product which will evaluate the performance of a new algorithm for calculating the TCP window size for low speed network links. This algorithm is to be implemented as two Linux kernel modules, and forms the Flow Control Module (FCM) part of RAMI. The second component of RAMI is SAM (Statistical Analysis Module), a statistical analysis package. This will be used to analyse the performance of the algorithm in a real world setting. The kernel modules are to be made publicly available under the GNU Public License.

## 1.3 Scope of this document

- This document applies to the quality assurance procedures undertaken by Team M during the entire project life-cycle.
- As a work in progress, this document will be constantly refined and updated as team procedures change and as new standards are decided upon (see the Changes Appendix, section A).

## 1.4 Document Conventions

For the conventions used in this document, see section 5.8.

## 1.5 People

### 1.5.1 Client

The Client is:

Lachlan Andrew  
 Dept of Electrical Engineering  
 The University of Melbourne

**Phone:** 8344 3816  
**email:** lha@unimelb.edu.au

### 1.5.2 Team

The Team doing this project is Team M, consisting of the following members:

Name	Username	Home Ph	Mobile Ph
Maya Muthuswamy	mayadm	XXXX XXXX	XXXXXXXXXX
Daniel Connor	connor	XXXX XXXX	XXXXXXXXXX
Chris Edwards	caedwa	XXXX XXXX	XXXXXXXXXX
Lars Yencken	lljy	XXXX XXXX	XXXXXXXXXX
Rod Howard	rihoward	XXXX XXXX	XXXXXXXXXX

### 1.5.3 Supervisor

The supervisor is:

Mark Ng  
 Dept of Computer Science & Software Engineering  
 University of Melbourne

**Phone:** 8344 9140

**email:** markn@cs.mu.oz.au

See section 3.2.1 for the role description and responsibilities of the supervisor.

### 1.5.4 External Reviewers/Auditors

The fourth year External Reviewers/Auditors are:

Name	Username	Mobile Ph
Peter Manolakos	pman	XXXX XXX XXX
Andrew Fox	adfox	XXXX XXX XXX
Jon Pak	jdpak	XXXX XXX XXX
Melissa Siah	msss	XXXX XXX XXX

See section 3.2.16 for the role description and responsibilities of the External Reviewers/Auditors.

### 1.5.5 340 Reviewers

The 340 Reviewers are Group N, consisting of:

Name	Username
Douglas McMeckan	mcmeckan
Edward Sholl	edwardas
Michael Grobelny	mdg
Sime Mardesic	sime
Andrew Lonsdale	lonsdale

See section 3.2.17 for the role description and responsibilities of the 340 Reviewers.

### 1.5.6 341 Reviewers

The 341 Reviewers are Group 13, consisting of:

Name	Username
Hrvoje Milekovic	hrvoje
Maisam Mirbagheri	maisam
Peter Haywood	phaywood
Travis Stapleton	travisrs
Naveen Kansara	naveen

See section 3.2.17 for the role description and responsibilities of the 341 Reviewers.

## 2 Reference Documents

### 2.1 Documents to be Produced

The documents which will be produced in the course of this project include:

- The Software Quality Assurance Plan (SQAP) [this document]
- The Software Requirements Specification (SRS)
- The Software Architecture Design Document (SADD)
- The Detailed Design Document (DDD)
- The Software Test Plan (TP)
- The User Documentation
- The Progress Report
- The Design Notebook
- The Traceability Matrix

### 2.2 Documents Used

The reference documents which will be used include:

- Arnold, T et al "Software Engineering Project Manual", 1993, Dep. Computer Science, Uni. of Melbourne.
- L<sup>A</sup>T<sub>E</sub>X web support <http://www.giss.nasa.gov>
- Mynatt, B "Software Engineering with Student Project Guidance", 1990, Prentice-Hall, London.

### 2.3 Glossary

The following terms are used in this document, and are defined to have the following meaning:

**CS** refers to The University Of Melbourne's Computer Science department, usually in reference to its computing facilities.

**SEECs** refers to the School of Electrical Engineering and Computer Science, of which CS is a part.

## 3 Management

### 3.1 Role Allocations

The following roles have been allocated:

<b>Name</b>	<b>Position</b>
Daniel Connor	Audit Manager Document Manager (TP) Test Manager
Chris Edwards	Backup Manager Document Manager (SQAP) Risk Manager Technical Leader Web Site Manager Release Manager
Rod Howard	Client Liaison Officer 340/341 Liaison Officer Document Manager (SRS) Internal Review Manager
Maya Muthuswamy	Project Manager
Lars Yencken	Librarian Technical Leader Document Manager (SADD) Document Manager (DDD)

### 3.2 Role Descriptions

This section defines the roles that will be used in this project and their associated responsibilities.

#### 3.2.1 Supervisor

The Supervisor is responsible for:

- Monitoring the progress of the team project (in a way that high level management would in a commercial setting)
- Monitoring the quality of all project deliverables (this will probably involve meeting with the team for reviews of each submission)
- Attending all agreed and arranged meetings with the Project Manager and the Team
- Assisting the Project Manager to ensure workloads are evenly distributed amongst Team Members
- Assisting the Team in relations with the Client
- Helping the Team develop project management strategies
- Assisting the Team in controlling slippage
- Assisting in resolving team conflicts which have been referred to the Supervisor for assistance
- Reporting to the coordinator any factors affecting the progress of the project

### 3.2.2 Project Manager

The Project Manager is responsible for:

- Functioning as the main point of contact between Supervisor and Team by:
  - Organising and attending regular Supervisor Meetings and Team Meetings (at least 1-2 hours per fortnight)
  - Providing adequate feedback both from the Team to the Supervisor and from the Supervisor to the Team
- Functioning as the main point of contact between the External Reviewers/Auditors and the Team, in particular by:
  - Organising External Reviews and meetings to discuss External Reviews with the External Reviewers and Team Members
  - Negotiating an audit and review plan with the External Reviewers/Auditors
- Coordinating the Team by:
  - Performing conflict resolution between Team Members
  - Maintaining the Project Plan and managing slippage
  - Ensuring tasks are evenly distributed amongst Team Members and are followed up
  - Ensuring that each Team Member submits a progress (regular) report after each submission
- Maintaining the Traceability Matrix
- Maintaining the Design Notebook
- Ensuring problems found during the Weekly Code Review (see section 5.3.4) are followed up

### 3.2.3 Risk Manager

The Risk Manager is responsible for:

- Determining the risks associated with the project, how to avoid each risk, and how to deal with each risk should it occur, to be detailed in section 10 of this document
- Maintaining a problem log (see section 10.10) of the problems arising and the strategies used to deal with each problem
- Maintaining a risk log (see section 10.9) of new risks identified and the strategies used to prevent risks occurring
- Creating contingency plans to be followed if risks occur
- Creating strategies for mitigation, monitoring and management of risks

### 3.2.4 Librarian

The Librarian is responsible for:

- Setting up and maintaining:
  - The CVS repository
  - READMEs for each team directory
- Ensuring that the External Auditors have access to the Team's documents
- Ensuring that all entries into the repository by Team Members are according to process (defined in section 9.7)
- Ensuring that all items in the group directory follow the file permission standards in section 9.10

### 3.2.5 Client Liaison Officer

The Client Liaison Officer is responsible for:

- Organising and attending meetings with the Client
- Acting as the primary point of contact between Client and Team Members
- Ensuring that queries/requests from other Team Members are communicated at meetings with the Client
- Organising for the Supervisor to be present at Client meetings when necessary
- Keeping the Client up to date on the Team's progress on the project

### 3.2.6 340/341 Liaison Officer

The 340/341 Liaison Officer is responsible for:

- Liaising with the Project Manager to come up with possible review dates for the 340 and 341 Reviews
- Acting as the primary point of contact between the 340/341 Reviewers and Team M
- Initiating contact with the 340/341 Reviewers
- Negotiating times for 340 and 341 Reviews with the 340 and 341 Reviewers
- Negotiating times for the meetings with the 340 and 341 Reviewers to be held after the reviews

### 3.2.7 Web Site Manager

The Web Site Manager is responsible for:

- Providing online access to:
  - Copies of documents, logs and meeting minutes and agendas from the repository
  - Posts from the mailing list
- Ensuring that the Web Site is updated nightly with all new/updated items from the mailing list/repository

### 3.2.8 Backup Manager

The Backup Manager is responsible for:

- Performing nightly backups, with archives of each night's backup for the past two weeks kept (following procedure in section 9.11)
- Extracting files and directories from these backups when requested by other Team Members
- In the case of data loss, restoring data from backups
- Recording each successful/unsuccessful backup in the backup log, to be kept on the Team Web Site

### 3.2.9 Meeting Chair

There is no set role of Meeting Chair, since the Meeting Chair is simply the person who organises the meeting (in most cases, the Project Manager). For any meeting, the Meeting Chair is responsible for:

- Preparing meeting agendas from ideas/issues raised by meeting participants in advance
- Starting and running the meeting according to procedure
- Emailing the agenda to all meeting participants by 10:00pm the night prior to the meeting
- Leading meeting discussions and ensuring all items on agenda are covered
- Allocating a Team Member to take minutes and ensuring minutes are adequately recorded

### 3.2.10 Audit Manager

The Audit Manager is responsible for:

- Monitoring group use of defined processes
- Organising External Audits and meetings to discuss External Audits with the External Auditors and the Team
- Ensuring responses to the External Audits are produced

### 3.2.11 Document Managers

A Document Manager is allocated for each major document produced by the Team. In his/her document, the Document Manager is responsible for:

- Ensuring that the document is compiled from contributions of Team Members
- Ensuring that the document is reviewed (where possible, both internally and externally) before submission
- Ensuring that the document is kept up to date
- Breaking down the task of document creation/modification into subtasks for the Project Manager to allocate to Team Members
- Ensuring that all criticisms and suggestions from reviews are acted upon (or not) with appropriate justification
- Ensuring that the review responses to any External Reviews are produced
- Submitting the document before the due date/time

### 3.2.12 Technical Leader

A Technical Leader is a source of knowledge and expertise that other Team Members are able to consult for help in solving technical problems. The responsibilities of a Technical Leader include:

- Aiding other team members in technical development
- Conducting and leading team research and skill development
- Leading the design phase of the project

### 3.2.13 Test Manager

The Test Manager is responsible for:

- Maintaining a set of tests and expected test results
- Ensuring regular testing of code is performed, as specified in the Test Plan
- Maintaining a log of all known bugs found in code

### 3.2.14 Internal Review Manager

The Internal Review Manager is responsible for:

- Organising the weekly code reviews

### 3.2.15 Release Manager

The Release Manager is responsible for:

- Preparing all releases of the project deliverables
- Organising in conjunction with the Project Manager, the allocation of tasks to ensure that releases of deliverables will be of a suitable quality
- Maintaining the quality of releases by deciding upon changes that will and will not be made to the repository immediately prior to the release

### 3.2.16 External Reviewers/Auditors

External Reviewers/Auditors are responsible for:

- Jointly creating (with Project Manager) a Review and Audit Plan
- Arranging reviews and meetings with the Team through the Project Manager (reviews) or Audit Manager (audits)
- Performing External Reviews and Audits according to the procedures defined in sections 5.3.5 and 5.4.1
- Providing advice to the Team on how to improve documents/processes through constructive comments

### 3.2.17 340/341 Reviewers

The 340/341 Reviewers are responsible for:

- Arranging reviews and meetings with the Team through the 340/341 Liaison Officer
- Each performing one review of the SRS following procedures defined in section 5.3.5.1

## 3.3 Project Plan

The major milestones for this project, which dictate the Project Plan, correspond with the following project submissions.

### 3.3.1 Major Milestones:

Milestone	Date
SQAP Submission	27/3/02
SRS Submission	26/4/02
SADD and Test Plan Submission	24/5/02
DDD Submission	9/8/02
Mid-Year Presentation	22/8/02
Preliminary Demonstration	9/9/02
Final Demonstration	17/9/02
Final Documentation Submission	25/10/02
Final Repository Submission	1/11/02

Details about the Project Plan can be found in section 9.6.

## 4 Documentation

This section describes all the major documents that will be produced. The audiences for all documents are at least Team M, Supervisor and Auditors, unless otherwise specified.

### 4.1 Software Requirements Specification (SRS)

The aim of the SRS is to describe each of the essential features to be implemented in the software without mentioning how they are implemented. It is determined by negotiation with the Client and acts as a contract for sign-off. An additional audience for the SRS is the Client.

The key objectives of the SRS are to:

- Specify software requirements (functions, performances, design constraints, external interfaces and attributes), enabling each requirement to be objectively verified/validated by the prescribed processes
- State all constraints that affect the system
- Establish customer sign-off

### 4.2 Software Architectural Design Description (SADD)

The aim of the SADD is to clearly depict how the high level structure will be designed in order to satisfy the requirements in the SRS. An additional audience for the SADD is future developers and maintainers of the RAMI product.

The SADD will consist of the following:

- The major modules in the system
- Relationships between modules
- What each module does, down to the function/method level (without describing how they do it)
- The traceability from the SRS to the design

### 4.3 Detailed Design Document (DDD)

The DDD will expand the architectural design, specifying the algorithms to be used and data types of components, and any other additional information needed to give a complete picture of the design of the system. An additional audience is future developers and maintainers of the RAMI product.

### 4.4 Test Plan (TP)

The aim of the Test Plan is to specify the procedures which will be used to verify and validate the RAMI software produced.

The different types of testing that will be used are:

- Unit
- Integration
- System
- Function
- Acceptance and Installation
- Documentation
- Graphical User Interface (GUI)

## 4.5 Traceability Matrix

The aim of the Traceability Matrix is to clearly show the traceability of requirements from the SRS through to the SADD, DDD, code, TP and testing results.

For each requirement in the SRS, there will be a reference to the module, class, method and test case (where relevant) that implements (or tests) the requirement. Also, for each requirement, the status of implementation of the requirement and the status of testing the requirement will be recorded.

The Traceability Matrix will be a Microsoft Excel file. Each time it is updated, the version will be stored under the `traceability` directory of the Team's home directory, with the name `traceabilityYY-MM-DD.xls`, where YY-MM-DD is the date the version was modified. This ensures there is a sufficient record kept of the Team's progress through implementation and testing of each requirement.

## 4.6 User Documentation

The aim of the User Documentation is to enable the user to properly install and use the software. The User Documentation will include:

- An installation and user manual for FCM, as required in the SRS
- An installation and user manual for SAM, as required in the SRS

The SAM User Documentation will also be contained in the SAM software system, and can be accessed via the *Help* menu, thus providing online help.

The FCM User Documentation will be made available as a text file. Included in the Installation Manual for FCM will be instructions on compiling the kernel modules and FCM logger, as well as instructions on loading the kernel modules.

The end users and Client are an additional audience for the User Documentation.

## 4.7 Design Notebook

The aim of the Design Notebook is to keep a record of everything that has contributed to the design and decisions of the project. It will be maintained in the `doc/design_notebook` folder in the Team's repository as much as possible, but where this is not possible (for example, design sketches) in a loose-leaf folder.

The Design Notebook will contain (or contain a reference to):

- Minutes and agendas of all meetings
- Documentation of all decisions (both design decisions and procedure decisions, see section 5.6 for more details)
- Critical appraisals of all decisions made (see section 5.6 for more details)
- Hard copies of the Project Plan (see section 9.6) which have been used and amended by the Team in meetings
- Results of reviews:
  - Physically amended copies of documents resulting from Team Reviews (see section 5.3.2)
  - Review reports (both from Team Reviews and External Reviews, see section 5.3.5)
  - Responses to reviews
  - Supervisor comments on reviews
  - Submitted documents and their results
- Sketches and diagrams which are used in the design process (including any from Client Meetings)
- Material used in presentations/demonstrations given by the Team
- Any research documents used in the design process

## 5 Standards, Practices, & Conventions

The following sets out the standard procedures to be followed by the team at meetings and during reviews and audits, as well as in documentation, communication, coding and testing.

### 5.1 Meeting Procedure

#### 5.1.1 General Meeting Procedures: applicable to all meetings

- The Meeting Chair for any meeting is the organiser of that meeting.
- If Team Members have any ideas/issues for the meeting, they must email these to the Meeting Chair at least 48 hours before the meeting.
- The Meeting Chair will then prepare the agenda following the template in the `doc/templates` folder of the repository, making it available by 10pm on the night before the meeting in the appropriate folder (`doc/mins/supervisor`, `doc/mins/general` or `doc/mins/client`).
- It is a guideline that the Meeting Chair brings a hard copy of the agenda to the meeting.
- At the start of the meeting, the Meeting Chair allocates a team member to take minutes. This allocation will be performed on a rotational basis to be fair and give everyone a chance. The person who takes the minutes is also responsible for typing up the minutes.
- The Meeting Chair must ensure the agenda is strictly followed. Any other business arising will be discussed at the end of the meeting, or added to the agenda for the next meeting.
- Every Team Member's opinion on decisions made and design issues plays a very important role in reaching the best solutions, hence all decisions will be made democratically - that is, three team members must agree for a decision to be made. For more details on decisions, see section 5.6.
- In the case where a decision cannot be agreed upon even after a full Team vote, the Supervisor will be called upon to help make the decision.
- Minutes must be typed up following the minutes template in the `doc/templates` directory of the repository, and added to the appropriate `doc/mins` directory (see above) by 24 hours after the meeting.
- All minutes and agendas can also be viewed at Team M's Website:  
<http://www.cs.mu.oz.au/SE-projects/s340gm/meetings/>
- A meeting may be cancelled or moved if all meeting participants agree by phone or email anytime before the scheduled meeting time.

#### 5.1.2 Additional Procedures for Team Meetings

- **Semester 1**
  - Regular Team Meetings will be held on Fridays at 1pm at Redmond Barry Courtyard during Semester 1.
  - However, if the Friday meeting is cancelled, the Project Manager must ensure that at least one other Team Meeting has been held or will be held that week, to ensure the Team is kept up-to-date on general progress.
- **Exams and Semester Break**
  - No Team Meetings will be held during the examination period (from 31st May to 17th June).
  - During the semester break, Team Meetings will be held as needed. This is due to the intensive programming sessions held (see section 5.2).

- **Semester 2**

- Regular Team Meetings will be held on Mondays at 12pm at Redmond Barry Courtyard during Semester 2.
- However, if the Monday meeting is cancelled, the Project Manager must ensure that at least one other Team Meeting has been held or will be held that week, to ensure the Team is kept up-to-date on general progress.

- Other Team Meetings may be organised as necessary by the Project Manager or any other Team Member.
- The quorum for a Team Meeting is 3 to ensure sufficient opinions and viewpoints on issues to be discussed. If the quorum is not present, the meeting shall be cancelled, or proceed informally (see section 5.1.5).
- Agendas must be named `gagendaYY-MM-DD.tex`, and minutes named `gminsYY-MM-DD.tex`.
- Any Team Member unable to attend a meeting must notify the Project Manager or some other attending member as soon as possible before the meeting. The absent Team Member must also read the minutes of the meeting before the next meeting.

### 5.1.3 Additional Procedures for Client Meetings

- The Client Liaison Officer organises Client Meetings with the Client, at least 4 days prior to the nominated time.
- If the Client is unable to attend a meeting, then email should be used to communicate with the Client.
- Client Meetings will be attended by at least 2 people, including the Client Liaison Officer and one Technical Leader to ensure different interpretations are available.
- The Supervisor will attend at least the first Client meeting.
- During the initial Client Meetings where requirements are being gathered, minutes will be taken by at least 2 people to ensure as much information is recorded as possible, or whenever possible, a dictaphone will be taken instead to make this even easier.
- The Client Liaison Officer will type up the collated minutes.
- Agendas must be named `cagendaYY-MM-DD.tex`, and minutes named `cminsYY-MM-DD.tex`.

### 5.1.4 Additional Procedures for Supervisor Meetings

- **Semester 1**

- Supervisor Meetings will be held on Tuesdays at 12pm at SEECS at least once a fortnight.

- **Exams and Semester Break**

- No Supervisor Meetings will be held during the examination period (from 31st May to 17th June).
- During the semester break, one Supervisor Meeting will be held at the start of the semester break (to discuss the plan for semester break), and one will be held at the end of semester break (to discuss the plan for semester two).
- Other meetings may be held as needed.

- **Semester 2**

- Supervisor Meetings will be held on Thursdays at either 3:15pm or 4:15pm at SEECS at least once a fortnight, except during the Midsemester break.

- The Project Manager will act as Meeting Chair.
- The agenda must be emailed to the team and Supervisor by 10pm two days before the meeting, instead of one day.
- At least 3 Team Members must be present for sufficient team representation (unless the Supervisor and Team agree otherwise).
- Agendas must be named `sagendaYY-MM-DD.tex`, and minutes `sminsYY-MM-DD.tex`.

### 5.1.5 Informal Meeting Procedures

- This procedure applies to any discussions which take place in person, but outside of an organised meeting.
- There are no formal agendas, hence no Meeting Chair.
- The requirement that minutes must be taken is waived.
- The purpose of Informal Meetings is to discuss ideas and concepts rather than make decisions, since the whole team may not be present.
- Therefore, any decisions made should not significantly affect the roles, responsibilities or tasks of those not present. One example would be an architectural design decision, which only affects those Team Members involved in design.
- However, when a decision is made, it will be recorded by a Team Member present, and posted to the Team Mailing List before the end of the day. This post will serve as a record of any such decisions.
- For specific details of decisions, see section 5.6.

## 5.2 Semester Break Procedure

The semester break (17th June to 28th July) is the time when Team M will complete the majority of the coding of RAMI, to ensure a lighter workload for Team Members during second semester. Intensive all-day group programming sessions will be held every weekday, so Team Members can benefit from working with others, and productivity can be increased.

- These sessions will be held at, depending on availability of the locations:
  - SEECS
  - Melbourne University Engineering faculty computer labs
  - Team Members' houses (with networked computers set up)
- Each Team Member will get one week's holiday (the date to be negotiated with the Project Manager at least one week before, to ensure adequate planning can be carried out).
- At the start of each session, Team Members shall discuss together the tasks that need to be done, who will do them, and come up with goals for the day, and also for the week.
- Due to the close contact during sessions, the Task Log (see section 9.12) does not serve any useful purpose, and therefore will only be used when necessary (for example, a Team Member not being able to attend a session, or when the Team is split up across more than one location).

## 5.3 Review Procedure

### 5.3.1 Entry Criteria

The entry criteria for a Team Review are:

- The document contains all material that is expected, possibly in rough form.
- There is sufficient detail in all areas of the document.
- The document compiles without fatal errors.

The entry criteria for an External Review are:

- The document has undergone at least one Team Review <sup>1</sup>.
- All criticisms arising from the Team Review(s) have been addressed.
- The External Reviewers have agreed to perform an External Review.
- A spell check has been performed.

### 5.3.2 Team Reviews

- Team Reviews are organised by either the the Project Manager or the Document Manager for the document to be reviewed, when they are satisfied the document has passed the entry criteria for a Team Review.
- The Document Manager must ensure that the document is reviewed by at least 3 Team Members.
- Team Members will usually get 1-2 days to review the document.
- During the reviewing time, Team Members must review the document and create a review report (text file) with name `review-report-DOC-NUM-LOGIN.txt`, where DOC is the acronym of the document being reviewed, NUM is the number of the Team Review, and LOGIN is the login of the Team Member. <sup>2</sup>
- By the end of the reviewing time, Team Members must have committed their review report into the `doc/team_review` directory of the repository, and also must have sent a copy of their review report to the Team's mailing list (with the keyword `[team]`).
- A Team Review meeting will be scheduled to discuss the Team Review, and will occur during a normal Team Meeting (see section 5.1.2).
- The agenda will therefore include the discussion of the Team Review.
- It is a guideline that Team Members bring a copy of their review report to the meeting.
- During the meeting, the Team must decide on the changes to be made.

---

<sup>1</sup>For the first External Review of the DDD, the Team decided against doing a Team Review prior to the DDD review, since the DDD was very similar to the SADD, and the changed sections had already been discussed at a Team Meeting. Refer to `smins02-08-08.tex` in `doc/mins/supervisor` of the repository for more details.

<sup>2</sup>Prior to the 25th August, the review consisted of:

- Each Team Member making notes on a hard copy of the document instead of creating a text file
- The Document Manager bringing a hard copy of the document to the meeting, which was used to record all the decided changes (or in the case where the decided changes were all changes by all Team Members, all the individual reviews were used by the Document Manager instead)

This was changed due to the overhead of printing out large documents and the ease of accessing soft copies instead. These hard copies are stored in the Design Notebook.

- The Document Manager must ensure these changes are acted upon (either by personally doing all the changes, or by delegating changes to other Team Members), within a decided time frame discussed at the meeting.
- The minutes for the meeting need not detail all the changes to be made to the document, but only need to contain a reference to the review responses (see below).
- While doing changes to the document, Team Members must annotate the relevant sections of each review report with the action taken for each point to produce the review responses.
- The review responses shall be named `review-response-DOC-NUM-LOGIN.txt`, where DOC is the name of the document being reviewed, and NUM is the number of the External Review, and shall be stored in the repository in the `doc/team_review` directory.

### 5.3.3 Individual Reviews

Most changes to the repository require an Individual Review before they can be committed to the repository. This section describes how an Individual Review is conducted.

Individual Reviews require at least two people: the author of the change, in this section called the Author, and at least one reviewer, in this section called the Reviewer. The Reviewer need not be known or allocated ahead of time.

For details on when an Individual Review is unnecessary, see section 9.7.

#### 5.3.3.1 Posting a Change for Review

To have a change reviewed, the Author must, in the following order:

1. Initially update the workspace by running `cvs update` from within the workspace, and resolve any conflicts generated in the process.
2. Post the proposed change to the mailing list in the following format:
  - The subject line should begin with `[review]` and follow with a short title for the change.
  - The email may begin with any general comments. If a particular Team Member is desired as Reviewer, he/she should be specified here. If the change requires more than one Reviewer, the number of reviewers required should also be specified here.
  - A general description of the change should follow, along with a specific comment for each group of files modified in the change, following the format of the file `cvslog` in the `doc/templates` directory of the repository.
  - The end of the email should include a diff of the workspace, performed by running `cvs diff -u` from within the workspace. Where a very large file has been added or modified, for example a postscript, PDF or binary file, it should be attached to the email rather than included in the diff (if the file is too large to send as an attachment, the Author should put the document on the web and post the URL of the website).
3. Attend to any criticisms from the Reviewer until the change is accepted; a particularly difficult change may need to be reviewed and fixed several times before being accepted.
4. Update the workspace again, by the same procedure as earlier.
5. Commit the change only after it has been accepted by the Reviewer and the workspace is current and contains no conflicts.

### 5.3.3.2 Reviewing a Change

A Reviewer is chosen to review a change posted to the mailing list in the following way:

- Within 78 hours of the change being posted, any Team Member may act as Reviewer and post a review of the change.
- After 78 hours, if no review has been posted, then the Author must inform the Project Manager by email, who in turn will allocate a Team Member the task of reviewing the change.
- This review must be completed within 24 hours.

The format of a review of a change is as follows:

- The review must be an email reply to the original post containing the change.
- The reviewer must comment on the change, and either:
  - Indicate in the review if the change is acceptable to commit to the repository, and if not, what must be fixed before it will be acceptable, or
  - For a complicated change, indicate in the review items to be fixed and request that the change be reposted for review afterwards.

To ensure that changes are reviewed in a timely manner, it is encouraged that Team Members review any changes they receive by the mailing list voluntarily.

### 5.3.4 Weekly Code Reviews

Each week, a review of all changes made to code will be conducted.<sup>3</sup> This review will be divided evenly across all Team Members unless otherwise agreed upon by the Team. The review will occur as follows:

- Each Thursday evening, the Internal Review Manager must check the Team Website and, using the CVS statistics, allocate in a roughly equal fashion, each file which has been modified in the last week to a Team Member. When possible, files should not be allocated to a Team Member who has made a change on the file during the previous week.
- The Internal Review Manager will then email to the Team Mailing List the list of files which have been allocated to each Team Member. This email will have the keyword [wr] in the Subject line. This email is to be sent by 11am of the next day (Friday).
- Team Members then have until 9am on the following Monday to conduct their review of the given files.
- In conducting their review, Team Members will:
  - Check the online CVS repository viewer on the Team's Web Site to view the changes which have been made in the previous week. Particular attention must be paid to those changes.
  - Read over the entire file, using the following as a code verification checklist:
    1. Inconsistent documentation
    2. Off-by-one errors
    3. Loops which never terminate
    4. Non-adherence to templates
    5. Confusing variable names
    6. Insufficient commenting
    7. Low quality code

---

<sup>3</sup>This procedure was introduced on 24th July, when the implementation phase had been started and the previous review procedure was not as effective as it had been for documents. See the `procedure_log` in `doc/design_notebook/decisions` for more details. This procedure ended on 4th October, when implementation had almost finished.

- 8. Insufficient error-checking
- 9. Throwing incorrect exceptions
- Send a review report to the Team Mailing List, listing all problems found with the files (as a reply to the original list of allocated files). This will take the format of the usual CVS log messages, with changes replaced by problems found. If an error is also corrected as part of the review, this should be noted.
- When all reviews have been completed, the Project Manager will allocate each problem found to a Team Member. How these tasks are assigned is up to the Project Manager's discretion, and tasks may be delayed or ignored if appropriate.

### 5.3.5 External Reviews

The following documents: SQAP(2), SRS(1), SADD(1), DDD(2), TP(2) will undergo External Reviewing by the External Reviewers (see section 3.2.16). The number in brackets indicates how many External Reviews are to be performed on the document by the External Reviewers. The process for External Reviews is as follows:

- An email will be sent by the Project Manager to the External Reviewers, notifying them of:
  1. The document the Team wants to be reviewed
  2. A proposed reviewing time (which will start at least 6 - 7 days after the email has been sent, and will be of 2 - 3 days duration, including weekends)
  3. 2 or 3 proposed meeting times and places for the Team to meet the Reviewers to go over the review (approximately 2 days after the end of the reviewing time)
  4. Where to find the document to be reviewed (in the **review** folder of the Team's home directory)
- The External Reviewers will reply to the email within 48 hours, notifying the Team of their preferred reviewing time and meeting time. If a time cannot be agreed on, it will be renegotiated by email ASAP.
- Once the reviewing time has been agreed upon, the Document Manager must place a copy of the document in the **review** directory of the Team's home directory before the start of the reviewing time.
- During the reviewing time, External Reviewers must review the document and create a review report (text file).
- By the end of the reviewing time, External Reviewers must have sent a copy of the review report to the Team's mailing list.
- The review report will be stored in the mailing list archive, on the Team's website, and also in the **review** directory of the Team's home directory with the name **review-report-DOC-NUM-LOGIN**, where DOC is the acronym of the document being reviewed, NUM is the number of the External Review, and LOGIN is the login of the External Reviewer.
- The review meeting will be conducted as per normal Team Meetings (see section 5.1.2).
- During the meeting, any unclear points made by Reviewers will be discussed, and changes needed will be decided upon.
- The minutes for the meeting need only contain reference to the review response (see below).
- Following the meeting, the Team will update the document with changes agreed to during the meeting.
- It is the Document Manager's responsibility to ensure these changes are made, either by him/herself, or by delegating changes to Team Members.

- After the changes have been made, the Document Manager must ensure that a response to the review is produced, which will be stored in the **review** directory of the Team's home directory, and will also be made available on the Team's Web site.
- The response will be a text file and will present the team's view and comments made, and any action taken or changes made.
- The review response shall be named **review-response-DOC-NUM.txt**, where DOC is the name of the document being reviewed, and NUM is the number of the External Review.
- If requested, the review response shall also be sent via email to the External Reviewer.
- It is also the Document Manager's responsibility to ensure that the process of updating the document and producing a review response (the seven above points) will take no longer than two weeks<sup>4</sup> measured from the review meeting (or in the case of numerous/few changes, another time period as agreed to by the Team).

Should the Reviewers or the Team wish to postpone the review after details of the reviewing time have already been finalised, they will contact the other party to make new arrangements. These will be made using the procedure for organising a review.

If the Team or Reviewers are unable to attend the Review Meeting, they will email the other party to postpone the meeting. If this occurs less than 24 hours before the meeting, or no response has been received by then, phone calls should be made to make arrangements.

If a Reviewer is absent from the meeting without notice, they will be contacted by email to arrange another meeting, or if deemed suitable by the Team, answers can be sought by email instead.

#### 5.3.5.1 340 and 341 Reviews

The SRS will also undergo one review performed by a 340 Team (see section 1.5.5), and one review performed by a 341 Team (see section 1.5.6). These reviews are a special case of External Reviews, and follow the same process as listed in section 5.3.5, with the following differences:

- 'External Reviewer' now refers to either '340 Reviewer' or '341 Reviewer'.
- Instead of the Project Manager organising the review, it will be done by the 340/341 Liaison Officer (after coming up with possible review dates in conjunction with the Project Manager).
- The document to be reviewed is always the SRS, hence need not be included in the email sent to the 340/341 Reviewers.
- Due to the relative inexperience of the 340 and 341 Reviewers, the reviewing time is expected to be 6 - 7 days, instead of 2 - 3 days.

#### 5.3.6 Exit Criteria

The exit criteria for Team and External Reviews are:

- All criticisms from the review have been addressed.
- The document conforms to standards laid out in this document.

---

<sup>4</sup>For the first DDD review, it was decided to not set a deadline as the DDD was continually being worked on at that time.

## 5.4 Audit Procedure

### 5.4.1 External Audits

External Audits involve the team and the External Auditors (see section 3.2.16). The process for External Audits is as follows:

- The External Auditors will email the Audit Manager at least 7 days before the planned freezing time. This email will contain:
  1. A proposed freezing date and time of the group directory (freezing time will be 3 - 4 days, including weekends)
  2. 2 or 3 proposed meeting times and places for the Team to meet the Auditors to discuss the audit (approximately 2 - 3 days after the end of the freezing time)
- The Audit Manager will reply to the email within 48 hours, notifying the Auditors of the preferred meeting time. If a time cannot be agreed on, it will be renegotiated by email ASAP.
- The SQAP will be frozen an additional 48 hours before the start of the freezing time, and will be placed in the `audit` folder of the Team's home directory by the Audit Manager (and will also be available from the website).
- This enables the External Auditors to read it and come up with an audit plan before the audit, and also enables them to ask for any hard copies of documents/Design Notebook before the audit starts.
- During the freezing time, the Team cannot alter the CVS repository.
- By the end of the freezing time, External Auditors must have sent a copy of the audit report to the Team's mailing list.
- The audit report will be stored in the mailing list archive, on the Team's website, and also in the `audit` directory of the Team's home directory with the name `audit-report-NUM-LOGIN.txt`, where *NUM* is the number of the External Audit, and *LOGIN* is the login of the External Auditor.
- The audit meeting will be conducted as per normal Team Meetings (see section 5.1.2).
- During the meeting, any unclear points made by the Auditors will be discussed and changes needed will be decided.
- The minutes for the meeting need only contain reference to the audit response (see below).
- Following the meeting, it is the Audit Manager's responsibility to ensure all criticisms from the Audit are acted upon.
- The Audit Manager must ensure a response to the audit is produced, which will be stored in the folder `audit` of the Team's home directory, and will also be made available on the Team's Web site.
- The audit response will be a text file and will present the Team's view and comments, and any action taken or changes made.
- The audit response shall be named `audit-response-NUM-LOGIN.txt`, where *NUM* is the number of the External Audit and *LOGIN* is the login of the External Auditor.
- If requested, the audit response shall also be sent via email to the External Auditors.
- It is also the Audit Manager's responsibility to ensure that the process of addressing criticisms and producing the audit response (the above five points) shall take no longer than ten days, measured from the meeting with the Auditors (or in the case of numerous/few changes, another time period as agreed to by the Team).

During an audit, the Team cannot alter the CVS repository. When notified of the audit, arrangements will be made such that every Team Member has work to do which can be completed without access to CVS, with minimal overlap. This reduces any problems with merging into the repository afterwards.

## 5.5 Task Allocation and Tracking

### 5.5.1 Allocation Procedure

Task allocation will be performed by considering the suitability of a team member for a particular task (for example, a task relating to CVS would be assigned to the Librarian). If the workload of the chosen team member is too large, the task will be reallocated to another member. The date for completion of the task will be negotiated between the Team Member and the Project Manager.

### 5.5.2 Meetings

The main time that tasks will be allocated is at team meetings, when input can be made by everyone. During meetings, tasks arising will be allocated and recorded in minutes. It is the Project Manager's responsibility to ensure the Task Log is updated within two days (see section 9.12 for details on the Task Log).

### 5.5.3 Outside Meetings

If tasks come up between meetings, it is the Project Manager's responsibility to delegate these tasks to team members, without a Team Meeting, as per the above procedure. It is also the Project Manager's responsibility to let the other team members know of the task allocation, including ensuring the Task Log is updated by the end of the day (see section 9.12).

### 5.5.4 Progress on Tasks

After working on a task, it is the responsibility of the team member working on the task to update the Task Log with their progress on the task. This should be done whenever progress can be reasonably estimated.

### 5.5.5 Missed Deadlines

Also, if the Team Member cannot finish the task within the time allocated, either more time must be negotiated with the Project Manager, or help sought. The Team Member responsible for the task must then update the Task Log accordingly.

### 5.5.6 Tracking

The Project Manager will track progress of tasks by frequent communication with team members and also by closely monitoring the Task Log.

## 5.6 Decisions

### 5.6.1 Recording Decisions

A record will be kept of all major decisions made during the project. The types of decisions are:

- Design decisions
- Procedure decisions (including role decisions)

These decisions will be recorded in either the Design Log, `design_log`, or Procedure Log, `procedure_log`, both to be stored in the `decisions` folder of the `doc/design_notebook` folder of the repository (see section 4.7 for more details of the contents of the Design Notebook).

For each decision made (either procedural or design), the following information will be recorded:

- **Date:** The date the decision was made
- **Agreed by:** Who has agreed to the decision (see below)
- **Context:** The background of the decision (for example, what part of the design it is relevant to)

- **Decision:** A description of the decision
- **Alternatives:** Any alternatives considered
- **Justification:** Why this decision was chosen over others
- **Critical Appraisal:** A critical appraisal of the decision (see 5.6.2)
- **Critical Appraisal Date:** The date the critical appraisal was performed (see 5.6.2)

The process for adding in decisions to either decision log is as follows:

1. A decision is made (either at a meeting or outside a meeting) by a group of people, or an individual.
2. The people involved in making the decision will decide amongst themselves who is to post the decision to the mailing list.
3. The poster of the decision must add the decision to the relevant decision log in CVS, filling in all fields except 'Critical Appraisal' and 'Critical Appraisal Date', and adding in the Team Members who have so far agreed to the decision under the 'Agreed by' field.
4. If there are at least three names (three is the majority for decision making) under the 'Agreed by' field, the poster can commit the decision to CVS, and proceed to step 6.
5. If there are less than three names under the 'Agreed by' field, the poster must wait until at least three people have agreed to the decision before committing to CVS.
6. The poster must then post this decision to the mailing list with the keyword [decision], so all Team Members have an opportunity to view the decision.
7. Team Members can either reply in agreement, or object to the decision (or parts of it, for example, a Team Member's name may have been added to 'Agreed by' when they have not actually agreed).

If three Team Members cannot agree on a decision, the people involved in making the decision are responsible for ensuring it gets added to the agenda for the next Team Meeting to be further discussed.

### 5.6.2 Critical Appraisal

All decisions made will be critically appraised and assessed by the Team. This critical appraisal will include:

- An evaluation of how successful, effective and useful the decision was
- The impact of the decision on the project
- Whether any of the alternatives would have been better
- Whether any changes are needed as a result of the decision

Critical appraisals for each decision will be recorded below each decision made in the design and procedure decision logs, as detailed in 5.6.1.

The procedure for performing critical appraisals is as follows:

1. After enough time has passed for the Team to gauge the impact of a decision, the Project Manager will include a critical appraisal of the decision as an item on the agenda for the next Team Meeting.
2. The Team will discuss the points noted above, and a Team Member will be allocated to record the discussion (may well be the same person who is taking minutes).
3. The Project Manager will allocate a Team Member to add in the critical appraisal (and date) to the relevant decision log, and will negotiate a time frame for this to be completed by.
4. After adding the critical appraisal, the Team Member may commit the relevant decision log to CVS without posting for Individual Review (see 5.3.3 and 9.7).

## 5.7 Release Control

In order to ensure that releases of deliverables are of a high quality, these procedures will be used:

- For a week preceding a release (or another time period as decided by the Release Manager), no new features will be added to the code of RAMI.
- This will ensure that the Team focuses on bug fixes rather than new functionality which may not get tested adequately before the release.
- It is up to the Release Manager's discretion as to what a 'new feature' is defined as, but will generally be a feature for which no existing test case exists.
- During the time when no new features can be added, any changes that are made to source code (not including test cases) must be posted for Individual Review (see section 5.3.3), even if the Weekly Code Review procedure (see section 5.3.4) is in place.
- This is to ensure that the changes made are in fact bug fixes and will not cause the stability of RAMI to decrease.
- Furthermore, the Release Manager must act as reviewer for all the changes posted, since the Release Manager knows which features will and will not be in the release.
- The Release Manager is then responsible for creating the release (in whatever form is needed) before the release.
- When producing a release, the Release Manager will produce a document containing "Release Notes", which will be included with the release. These release notes will describe any changes in the new release from any previous ones, including, but not limited to:
  - Known bugs
  - The purpose of the release
  - Any new functionality included in the release

## 5.8 Documentation Format Standard

All major documents are to be in  $\text{\LaTeX}$  format, and should also be available at the team web site in HTML format.

All major documents will have the following:

- Title Page
  - Document title
  - Group name (Team M)
  - Name and logins of group members
  - Abstract, briefly describing the document
  - Maintainer of document
  - Version and date
- Table of Contents
  - Lists all sections, sub sections, sub sub sections, and sub sub sub sections where necessary
- Table of Figures (if figures are included in the document)

Formatting standards for each document should be the  $\text{\LaTeX}$  default standard where applicable. The following standards must also be adhered to:

- All sections must begin on a new page.
- Lists should always begin with a capital letter and only end with a full stop if they make a complete sentence.
- Every word in a section heading will begin with a capital letter, except for common prepositions (and, of, the, etc).
- Directory names will be presented in the `texttt` style of  $\text{\LaTeX}$

The following conventions will apply to  $\text{\LaTeX}$  source files:

- URLs will be presented using the `url` command
- All lines in  $\text{\LaTeX}$  source files will have a maximum line length of 80 characters
- Major documents will have each section split into a separate file to be included by the main source file
- Items of lists must be indented one level
- Where a change will be needed later, a comment will be created containing `FIXME` to ease locating such areas later
- Cross-references to other parts of a document will be indicated in this manner: (see section 4.2).
- Team members will be referred to by their position titles rather than their personal names. The list of positions is given in 3.1.
- All relevant file and directory references will be relative to the Team M's group directory `/home/case/340/s340gm/`.

## 5.9 Coding Standard

The coding standard to be followed is that described in the Coding Standard appendix (see section B) of this document. The aim of the coding standard is to reduce required development/maintenance time through code that is readable, efficient and well documented.

## 5.10 Channels of Communication

Email will be used as a primary source of communication between Team Members, the Supervisor and the Client. Copies of all emails will be kept by the Librarian, however, all Team Members are encouraged to also keep copies.

In the case that the CS systems are unavailable, each team member has an alternate email address, as follows:

Username	Alternative Email Address
connor	kojin@alphalink.com.au
caedwa	chris@edwards.mine.nu
lljy	lyencken@froggy.com.au
mayadm	falgar@bigpond.net.au
rihoward	rian@bigpond.net.au

After the outage is over, the Project Manager must ensure all email is forwarded to the Team's address so it may be archived.

In the case where email is too slow or unavailable, communication will be by phone.

## 5.11 Changes to the SQAP after External Review

Once SQAP has been externally reviewed (see section 5.3.5) and all changes resulting from the review have been completed, then the SQAP itself is considered completed. This means the SQAP has been approved and will be used for the Team's processes. However, after this time, the SQAP may still change. Some possible reasons for changing the SQAP are:

1. It may contain deficiencies.
2. Processes may need to be rethought if they are ineffective.
3. Changes in project structure may make sections of the SQAP obsolete.
4. Spelling/grammatical errors are picked up.
5. Sections which could not be completed by the time the SQAP was externally reviewed need to be filled in.

If a change to the SQAP does not fall under number 4 or 5 in the list above, then:

- The change must be accompanied by a description in the Changes appendix of the SQAP. These changes are to be listed with the most recent at the beginning.
- This description must include what was changed, the date of the change and the reason for the change.

## 5.12 Changes to Roles

Changes to roles may occur either because a Team Member wishes to change one of their roles, or because either the Project Manager or Risk Manager decides a change is required. Any change must be approved by at least 4 Team Members at a meeting, or alternatively, by 3 other Team Members by email. If voted on at a meeting, the reasons for the change must be documented in the Minutes, and the proposed change must be on the Agenda. To change a role by email, the suggested change must be posted for review, and 3 other Team Members must approve the change. The email postings will serve as sufficient record of the change, and should document the reason for the change.

It is the responsibility of any Team Member affected by a role change to ensure they fully understand any new or modified role. Any queries should be raised with other Team Members either by email or at the Team Meeting. Once a change has been made it will be assumed that the role is understood by the Team Member assigned to it.

## 5.13 Guidelines for professional development

This section gives guidelines that the Team should follow. These are only guidelines (and not procedures) since they are unverifiable.

- Team Members are expected to check their email at least once a day to keep up-to-date.
- Team Members are expected to check and update the Task Log regularly, to ensure the Project Manager and rest of Team know about their progress.
- Before committing any source files that are able to be compiled (for example, latex source, code), Team Members are expected to compile the source file to ensure no compilation bugs enter the repository.

## 6 Reviews and Audits

This section outlines the reviews and audits which are to be conducted, detailing the purpose of each of these reviews or audits, and the criteria which will be used in reviewing or auditing. Each review or audit will be conducted at a time negotiated between the Team and the reviewer/auditor. All reviews/audits will involve at least the following personnel (unless otherwise noted):

- All Team Members
- External Reviewers/Auditors

### 6.1 Software Quality Assurance Plan (SQAP)

Initially the SQAP will be internally reviewed, as soon as the first draft has been completed, according to the following criteria:

- The SQAP contains correct and agreed upon material
- The SQAP contains enough detail in all areas for the submission
- The SQAP meets the necessary criteria laid out in the Project Manual

The SQAP will also be externally reviewed. The External Reviews will be conducted according to the following criteria:

- The project and its aims have been described and team identified
- All necessary documents are referenced
- Team Members have been allocated roles and their responsibilities have been made clear
- The responsibilities of the Supervisor, External Reviewers and External Auditors have been explained
- The project plan shows:
  - The phases of the project and submission dates
  - The schedule of completed tasks for the Concept Phase
  - The projected schedule for the Specification, Architectural Design, Detailed Design, Implementation, Integration and Testing Phases
- All planned quality assurance documents are listed and described
- There is a standard meeting procedure and meeting time
- Method for task allocation has been described
- Method for specification has been described
- Standards have been set for documentation and coding
- Each major review has been described with:
  - Types of review for each phase
  - Items to be reviewed
  - Personnel involved in each review
- Document verification methods have been described
- Procedures for problem reporting and correction have been described

- Design methodology has been described
- Tools and systems for documenting decisions and designs have been described
- Media control detailed
- Risk management described

## 6.2 Software Requirements Specification (SRS)

The SRS will initially be internally reviewed once the first draft is complete, according to the following criteria:

- The SRS is feasible and contains only necessary requirements
- The SRS meets criteria listed below for SRR
- Requirements are objective, clear and testable
- The SRS is complete and all constraints are described
- There are no incomplete sections in the SRS
- Requirements are understandable
- The SRS follows the standards in the SQAP for documentation

The Document Manager for the SQAP should also ensure that changes needed to the SQAP as a result of the SRS are made. The SRS will additionally need to be reviewed by the Client during the Software Requirements Review (SRR). The SRR is held to ensure the adequacy of the requirements stated in the SRS, and is held between the team and the Client.

The criteria that will be examined at this review are:

- The purpose of the project is given
- The existing system is described
- The proposed system is described
- The functional requirements are explained, numbered for traceability and ranked in order of importance
- The non-functional requirements are described, including:
  - Identifying the type of user and what they will use the project for
  - Manual(s) needed
  - Hardware needed
  - Performance constraints
  - Error handling and defaults
  - Interfaces between hardware and software
  - Likely modifications
  - Constraints on the physical environment
  - Security needs
  - Design constraints
  - Implementation constraints
- The non-core requirements are ranked by the Client in order of importance

- The user interface is described
- Documentation and training requirements are described
- Acceptance criteria is specified
- Examples of system behaviour are given
- Glossary for words/acronyms specific to project
- There is a section for the Client and team to sign off the SRS

There will also be External Reviews of the SRS, conducted according to both sets of criteria listed above.

### 6.3 Software Architectural Design Description (SADD)

The SADD will initially be internally reviewed, in which the following criteria will be assessed:

- The structure designed meets all the requirements set out in the SRS and cross-references to relevant parts of the SRS are included.
- The SADD contains:
  - The design description (including overview of system structure, each module in the module design, the name and/or ID of every module, clear and complete diagrams)
  - The module design descriptions (including each module appearing in the design, all information which is available to that module including any references to specifications in the SRS)
  - All major data structures are described
  - Modules and module breakdown chosen are appropriate
  - Diagrams conform to style guide

The Document Managers for the SQAP and SRS should also ensure that any changes needed to the SQAP and SRS as a result of the SADD are made. The Project Manager should ensure that the Design Notebook contains design decisions, their justification and the trade-offs that have been made at this point.

There will also be an External Review held to evaluate the technical adequacy of the preliminary design of the software as depicted in the SADD. The criteria for this review are those listed above.

### 6.4 Test Plan (TP)

The TP will initially be internally reviewed, and the following criteria will be examined:

- The TP describes
  - Test plan for the code
  - Unit test strategy
  - Integration test strategy, including order of integration
  - System test strategy
  - Function test strategy
  - Acceptance and Installation test strategy
- There is some reference to regression testing
- The TP specifies all tests needed to produce a quality product
- All the functionality described in the SRS is fully tested

The Document Manager for the SQAP should also ensure that any changes needed to the SQAP as a result of the TP are made. There will also be an External Review held to evaluate the adequacy and completeness of the testing methods defined in the TP. The criteria for this review are those listed above.

## 6.5 Detailed Design Document (DDD)

The DDD will initially be internally reviewed, according to the following criteria:

- Algorithms for every module are chosen (and described if necessary by pseudocode)
- Data structures are finalised
- The DDD is amended so that:
  - Design description includes any new modules and any changes to relationships between existing modules
  - Module designs include each module appearing in description and complete module headers for each module (possibly in the form of automatically generated online documentation)
  - Major data structures include name of each user-defined data type and how to access its sub-parts, and name and value of each user-defined constant
  - All diagrams conform to style guide

The Document Managers for the SQAP, SRS and TP should also ensure that any changes needed to the SQAP, SRS and TP as a result of the DDD are made. The Project Manager should ensure that the Design Notebook has been updated with any new design decisions.

There will also be an External Review held to determine the acceptability of the DDD in satisfying the requirements laid out in the SRS. The criteria for this review are those listed above.

## 6.6 User Documentation

The User Documentation will undergo a User Documentation Review, which will be a Team Review only (will not involve External Reviewers). The criteria examined at this review are:

- Contains an accurate description of the system (that is, it only describes functionality that is in the system and not non-existent functionality)
- Contains a complete description of the system (that is, it describes all functionality in the system)
- Is written in a clear and concise form to enhance usability (that is, it is easy to follow and understand)
- Contains clear and useful diagrams, figures and screen shots to guide the user through difficult steps of the program, thus further enhancing usability
- Contains a Table of Contents to enable easy navigation
- Contains a brief overview of the system at the beginning of the document
- Contains standard navigation facilities (Next, Previous, Up, Home) as well as links within the text to other relevant sections
- Contains no spelling or grammatical errors

This review will be held within the Team only.

## 7 Verification, Validation and Testing

This section describes the procedures used to verify and validate documents and code produced during this project.

### 7.1 Documents

The documents produced during this project will be verified and validated by:

- Undergoing Individual Reviews before being committed to CVS, when a document changes (see section 5.3.3)
- Undergoing one or more Team Reviews (see section 5.3.2)
- Undergoing one or more External Reviews (see section 5.3.5)
- Undergoing a final detailed readthrough and review by at least one Team Member prior to the final proofread
- Undergoing a final proofread by the Document Manager

The Team Reviews and External Reviews are to be completed following the criteria detailed in section 6.

The SRS will additionally need to be reviewed by the Client in the SRR, detailed in section 6.2.

### 7.2 Code

The code produced during this project will be tested following the procedures outlined in the Test Plan, and also by tracing the requirements through the Traceability Matrix (see section 4.5). Also, during the major implementation phase, all changes to code will undergo a Weekly Code Review (see section 5.3.4) to ensure the quality of code remains at a high standard.

Once the amount of changes to code per week becomes sufficiently small (as decided by the Team), Weekly Code Reviews will stop, and code will undergo the same Individual Review procedure (see section 5.3.3) as documents before entering the repository.

### 7.3 End Product

The end product will be verified using the Traceability Matrix (see section 4.5), which ensures all requirements specified in the SRS have been implemented in the end product.

## 8 Problem Reporting & Corrective Action

### 8.1 Reporting of Errors

Errors fall into two major types:

- Programming/Technical errors
- Documentation errors

#### 8.1.1 Programming Errors

Programming errors will be handled differently to Documentation errors. When a bug is discovered, either by testing or code inspection, it should be reported by email to the mailing list with the subject keyword [bug]. The Test Manager should also add the bug to the Bug Log (see section 8.3).

#### 8.1.2 Documentation Errors

Documentation errors found should be fixed by the finder of the error, and posted for Individual Review as usual (see section 5.3.3). The finder may alternatively email the Document Manager for that document describing the error, in which case the Document Manager is responsible for ensuring that the error is fixed. Special cases are changes to the SQAP after External Review (see section 5.11) and changes to the SRS after Sign Off (see section 10.5).

### 8.2 Corrective Action

When a bug has been reported, the Team Member responsible for that section of code should attempt to correct it. If they are unable to do so, they should post to the mailing list seeking help. If they are still unsuccessful in correcting the bug, the Test Manager must be notified, and the responsibility for ensuring the bug is fixed falls to the Test Manager (who may assign another Team Member or Technical Leader to fix it).

### 8.3 Bug Log

The Testing Manager will be responsible for keeping a log of all known bugs. This will be a simple text file, and will include:

- Description of bug
- Date bug was found
- Description of problem and impact
- Who was assigned to fixing the bug
- What was done to fix the bug
- Date bug was fixed

This log shall be stored in CVS, in the `doc/bug` directory.

## 9 Tools, Techniques & Methodologies

### 9.1 Design Methodology

For implementation and submission of documents, the waterfall model will be used. This means that the team will produce a SQAP, an SRS, an SADD, an DDD and a TP in that order. However, within each phase, an iterative process will be used, whereby the document is refined through reviews. The documents will also be reviewed after the next phase commences, and phases will overlap.

### 9.2 Software

The following software will be used to complete the project:

**Microsoft Project** - Used by the Project Manager to plan the project, including the production of Gantt charts

**L<sup>A</sup>T<sub>E</sub>X** - Used for all major documents, and other minor documents where appropriate

**procmail** - Used to sort mail, in particular to automatically place posts to the Team Mailing list into the Web Site Mailing list archive

**hypermail** - Used to produce an HTML Web site Archive of Mailing list posts

**rsync** - Used to backup group directory to off-site backup while using very little bandwidth by only transferring changes

**tar, gzip** - Used together to produce archive copies of group directory every night

**CVS** - Tool chosen for version control of Team documents and code

**dia** - The GNOME diagram editor, to be used for diagrams within documents

**Python** - The Python programming language, to be used for writing the GUI component of the project

**Glade** - A GUI design program, to be used for graphically designing and implementing the GUI component of the project

**Viewcvs** - A web-based viewer for the Team's CVS repository

**Pychecker** - A static analysis tool for Python, to be used as part of testing for any Python components

**Happydoc** - A tool for extracting documentation from a Python program, to be used to create self-documenting code.

**Kernel-doc** - A tool for documenting Linux kernel source code, to be used to create self-documenting code.

This list will be updated as new software packages are required.

### 9.3 Choice of language

The language chosen for the Statistical Analysis Module (SAM) part of RAMI is Python. The reasons for this choice have been described in the minutes from Team Meeting number 10, which are on the Team's website at: <http://www.cs.mu.oz.au/SE-projects/s340gm/meetings/general/gmins02-05-06/>. Also, the decision has been recorded in the Procedure decision Log, `procedure_log`, stored in the `decisions` folder of the `doc/design_notebook` folder of the repository.

The language that the Flow Control Module (FCM) will be written in is C. The reason for this is that FCM will consist of kernel modules, which must be written in C.

## 9.4 Tools for coding

Team Members should use available static analysis tools, including compiling with all warnings enabled. For python, the static analysis tool that should be used (and is automatically used when using the testing script) is `pychecker`. The programming environment for the project is Linux, hence shell scripting tools may be constructed to aid in development.

## 9.5 Tools for Diagrams

Where possible, diagrams should be drawn using the GNOME diagram editor, `dia`. They should be saved in `dia`'s native format, but exported to Encapsulated Postscript for inclusion into documents.

## 9.6 Project Management

Microsoft Project (on the Project Manager's home computer) will be used to create a Gantt Chart representing the Project Plan, which will assist the Project Manager (and Team) in planning the project. To ensure that the whole Team participates in planning the project, the Project Plan will regularly be brought to Team/Supervisor Meetings, so the Team can discuss the plan together and come up with an agreed plan that suits everyone. These modified copies of the Project Plan are stored in the Design Notebook.

The Project Plan will regularly be updated by the Project Manager, and each version of it (named `projectplanYY-MM-DD.mpp`) will be stored in the `plan` directory of Team M's group directory as a `.mpp` file. It will also regularly be converted to GIF formats to be accessed from Team M's website.

### 9.6.1 Use of the Project Plan

The Project Plan will not only be used as a planning tool, but will also be used to aid in motivation.

By providing a visualisation of the entire project, the Project Plan enables the Team to actually see the current status of the project: all tasks that have been completed, that are currently being completed and that are planned for the future. Also, the Project Plan is invaluable in planning for external events (reviews and audits), since it enables the Team to not be at a disadvantage and to prevent any conflicts (for example, planned major coding development would not be placed during an audit). In this way, the Project Plan is a vital tool to aid in planning for the future, since the placement and duration of future events can be effectively estimated.

The Project Plan also aids in allocating time for slippage. By colouring events that have slipped or events that could cause slippage in red, this alerts the Team to work extra hard to ensure slippage does not occur. Also, the use of dependencies between tasks means that if one event slips, it could cause many more (dependant) tasks to slip as well. To counteract this, task estimates are always on the higher side, and extra time is left after important tasks so that if slippage does occur, time has been sufficiently allocated for this.

Finally, it has been found that the Project Plan aids in motivation. It helps the Team to stay motivated because the end of the project is always 'in sight' on the Project Plan. By providing a graphical picture of the tasks completed, the Team can gauge the amount of work done, and see that their tasks have not been futile and have in fact been helping the Team reach the final goal. The pressure of seeing external events in the near future (a different colour to normal, internal tasks) ensures the Team stay motivated to work to bring the document/code up to standard before the submission/review/audit, since these external events are (somewhat) fixed and cannot be altered.

## 9.7 Version Control

CVS will be used for version control. The Librarian will maintain the repository, ensuring that all Team Members have access to the project files and documents.

The items to be stored in CVS for version control are:

- All documentation developed by the Team, including:
  - The 5 major documents (SQAP, SRS, SADD, DDD and TP)

- All user documentation
  - All developer documentation
  - The agendas and minutes of all meetings for which they are recorded
  - Any online items of the Design Notebook
  - Any templates used by the Team
  - The logs used by the Team (task, risk, bug, problem)
- All source code for the product itself
  - Optionally, the source code of any developer tools created that require version control (as decided by the creator of that tool)

The procedure for making a change to the repository depends on the type and complexity of the change. All changes to the repository must undergo an Individual Review (see section 5.3.3), except for the following cases. Changes that can be committed without review are:

- All changes to source code within the `src` directory when the Weekly Code Review (see section 5.3.4) procedure is in use
- The initial draft of a document (or part of a document)
- Changes to a document which fix either spelling mistakes or grammatical expression only without changing the meaning of a document
- Changes to a document's formatting only by the Document Manager of that document
- Changes to the Task, Risk, Problem and Bug Logs
- Changes to the procedure Decision Log or design Decision Log (see section 5.6)
- Changes to the overall test report
- Changes to the source code of a developer tool by the author of the tool
- The addition of meeting agendas or minutes to the repository
- Changes to README files

## 9.8 Records, Maintenance & Media Control

The Project Manager will keep a Design Notebook, containing design decisions (see 5.6 for more details on decisions), as well everything described in section 4.7. The hard copy items are to be kept at the Project Manager's house, and made available to Team Members, Supervisor or Auditors on request. The rest of the Design Notebook will be stored under CVS, as detailed below.

### 9.8.1 Directory Structure

The Team M directory `/home/case/340/s340gm` has the following top-level directory structure:

- `audit` - location where the current version of the SQAP is made available for External Auditors, and where external audit reports and their responses are stored, and also the midyear feedback report
- `bin` - executable tools for use by all Team Members
- `mail-archive` - archives of email from the mailing list
- `plan` - Gantt charts
- `reference` - reference documents for RAMI

- **repository** - the CVS repository, see below for contents
- **review** - location where documents are made available for External Reviews, and where external review reports and their responses are stored
- **test\_reports** - all testing reports for RAMI
- **traceability** - versions of the Traceability Matrix
- **www\_public** - the Team web site

### 9.8.2 Repository Structure

The directory `/home/case/340/s340gm/repository/` has the following directory structure:

- **doc** - documentation of all forms
  - **bug** - the Bug Log
  - **critique** - the critique of Team N's SRS
  - **design\_notebook** - the online parts of the Design Notebook
  - **devel** - developer documentation
  - **man** - user manual files
  - **mins** - minutes and agendas of meetings
    - \* **client** - client meetings
    - \* **general** - general meetings
    - \* **supervisor** - supervisor meetings
  - **problem** - the Problem Log and Risk Log
  - **SQAP** - Software Quality Assurance Plan
  - **SRS** - Software Requirements Specification
  - **SADD** - Software Architectural Design Document
  - **DDD** - Detailed Design Document
  - **team\_review** - review reports and responses from Team Reviews
  - **templates** - templates for documents, reports, code and submissions
  - **TP** - the Test Plan
- **src** - source files for the project and project modules
  - **fcm** - source files for the FCM component of RAMI
    - \* **logger** - source files for the logger component of FCM
    - \* **receiver** - source files for the receiver component of FCM
    - \* **router** - source files for the router component of FCM
  - **prototype** - source files for the prototype of SAM
  - **sam** - source files for the SAM component of RAMI
  - **test** - test cases for RAMI and a test progress report
    - \* **fcm** - test cases for the FCM component of RAMI
      - **logger** - test cases for the logger component of FCM
      - **receiver** - test cases for the receiver component of FCM
      - **router** - test cases for the router component of FCM
      - **stress** - test cases for Stress Testing of FCM

- \* **function** - test cases for Functional Testing
- \* **sam** - test cases for the SAM component of RAMI plus an automated script for running SAM's test cases
  - **GUI** - test cases for the graphical user interface component of SAM
  - **data** - test data for SAM
  - **integration** - integration test cases for SAM
  - **unit** - unit test cases and stubs for SAM
- **tools** - developer tools created for the project
  - **config** - configuration files for developers
  - **scripts** - executable tools for developers
  - **tasks** - the Task Log

## 9.9 File Naming Standards

All file names must be less than 30 characters. The following additional constraints apply to the file name but not the file extension.

For documents:

- Any document file must be named after the largest part of the document that it contains, e.g. `SRS.tex` for the main file of the SRS, or `Tools.tex` for a section called Tools.
- Names must be either an acronym in capitals, or one or more words beginning with capital letters and without spacing, e.g. `SQAP.tex` or `FurtherProblems.tex`.
- If any other file represents a numbered section of the document, the filename should be prefixed with that number, e.g. `04Management.tex`. All sections of the same document should have the same number of digits in the filename.
- If a file is an appendix to the document, it should be further prefixed with A, e.g. `A1Changes.tex`. Note that appendices use their own numbering separate from the rest of the document.

For code:

File naming Standards for code are listed in section B.1.7.

## 9.10 File Permission Standards

Files should have their permissions set according to the following rules by their owner (whoever created them). Permissions for the owner should be set identically to group permissions as set out below.

1. All directories should be readable and executable by all users, and writeable to the creator and to the group (`chmod 775`).
2. If a file is not to be edited further, then it should be read-only to all users (`chmod a+r-w`).  
*Note:* This includes files which may be replaced by updated versions but not edited themselves, for example, a `dvi` file generated by  $\text{\LaTeX}$ .
3. If a file is to be edited further, then either:
  - (a) The file is under CVS, and the permissions should be left as handled by CVS; or
  - (b) The file is under an alternate form of version control, and the permissions used should be specified and justified in the README for that directory by its maintainer; or

- (c) The file is not under version control, and should be made writable to all group members, and readable to all (`chmod g+rw, chmod o-w`).

*Note:* This is allowed, but not recommended. It is better to place files under version control of some sort.

4. Additionally, if a file is executable, then it should be executable by all group members but not by all users (`chmod g+x, chmod o-x`).

The maintainer of a directory may choose to use alternate file permissions, however these must be explicitly documented and justified in the README file for that directory.

## 9.11 Backup Procedure

Nightly backups will be performed by the Backup Manager, on the group's directory `/home/case/340/s340gm/` (see section 3.2.8). The following procedure will be followed.

- A nightly cron run will update the backup directory on the Backup Manager's home computer using `rsync`. The Backup Manager's home computer is located at his house, and is permanently connected to the Internet, allowing the procedure to take place automatically.
- This backup directory will then be archived as a gzipped tar file in another directory. The backup directory will not be deleted, as only changes will be transferred the following day.
- The backup log will be updated, and transferred to the Team's Web Site for viewing.

## 9.12 Task Log

A Task Log will be used to list tasks to be completed and their progress. The data used to generate reports will be stored in the `tools/tasks` directory of the repository to maintain a reliable history, and from this an online version of the Task Log will be produced for the Team Web Site, and will be updated hourly, to ensure Team Members are kept up to date on the progress made. For details on the task allocation procedure, see section 5.5.

The Task Log will record the following for each task:

- Task name
- Team Member it has been assigned to
- Percentage completed, or cancellation if appropriate
- Date assigned
- Date to be completed
- A longer description of the task, with any extra information required to complete it
- The estimated number of hours the task was expected to take
- The actual number of hours the task took or has taken to that point

The Project Manager can add in tasks for any member to do, whereas other Team Members can only add in tasks for themselves to do (unless the Team Member has consulted with the Project Manager).

### 9.13 Team Web Site

The Team's Web Site provides easy access to information for Team Members, and also serves as a record for auditing purposes. The address of the Web site is <http://www.cs.mu.oz.au/SE-projects/s340gm/>. The following will be made available at this address:

- An archive of Mailing List posts, broken down by month, with review messages separated from others, and the ability to sort by author, date or subject
- HTML versions of all Agendas and Minutes from all meetings, as found in the Team's CVS repository
- A copy of each major document (SQAP, SRS, SADD, DDD, TP) from at least the latest External Review
- Copies of each External Review and External Audit, as well as their corresponding response
- The Task Log (updated hourly)
- CVS statistics for the percentage and number of files committed by each Team Member for the past week, and also the overall statistics for the `doc` and `src` directories
- An online viewer of the CVS repository
- The Backup Log
- A GIF version of the latest Project Plan
- Documentation for all source code (generated by `Happydoc` for SAM and `Kernel-doc` for FCM) which forms the major part of the DDD
- Internal Documentation for training purposes
- A Debian package page giving instructions on how to install the RAMI Debian package
- Links to any other web pages appropriate for the project

### 9.14 Email Standards

The `hypermail` program will be used by the Web Site Manager to archive mailing list posts on the Team Web Site. All emails will be sent to all Team Members through the team email address `s340gm@cs.mu.oz.au`. An appropriate subject line must be included, which shall describe the contents of the email such that it is clear to Team Members whether they need to read it or not. One or more keywords should also be included at the beginning of the subject line, taken from the following list:

- `[client]` (Client communication)
- `[review]` (code/documentation to be reviewed for entry into the repository)
- `[team]` (team related matters: meeting agendas, minutes, etc) This is the default if no keyword is specified.
- `[super]` (supervisor communication)
- `[devel]` (development discussion)
- `[bug]` (discussion of bugs found in code)
- `[risk]` (discussion of risks to be placed in the Risk Log, see section 10.9)
- `[problem]` (discussion of problems to be placed in the Problem Log, see section 10.10)
- `[username]` (communication directed at a specific Team Member)
- `[wr]` (Weekly code review assignments)
- `[decision]` (decisions made, see section 5.6)

## 10 Risk Management

### 10.1 Risks Identified

This section outlines the main risks involved with the project, and the processes to follow to avoid or deal with them. It is expected that other risks will be identified as they arise throughout the year, however these will not be added to the SQAP, they will be added to the Risk or Problem Logs (see sections 10.10 and 10.9).

#### 10.1.1 High Probability Risks

- Schedule slippage due to illness of Team Member(s)
- Schedule slippage due to inexperience with tools being used, including programming languages
- Schedule slippage due to inexperience at planning

#### 10.1.2 Medium Probability Risks

- Design not producing a product which meets requirements set out in SRS
- Essential requirements being missed in SRS
- Schedule slippage due to insufficient knowledge of technologies involved

#### 10.1.3 Low Probability Risks

- Schedule slippage due to inability to contact Client
- Loss of data
- Schedule slippage due to delay in finding suitable project

### 10.2 Risk Mitigation

#### 10.2.1 Illness of Team Member(s)

It is expected that at some point, and probably more than once, a Team Member will become ill and be unable to fulfill their tasks and roles in the project. To mitigate this risk, the Project Manager will allow extra time in the Project Plan, and this will protect against illness lasting a few days. Should the illness last for such a time that Project Plan deadlines cannot be met, the Project Manager and Risk Manager must reallocate tasks to ensure the ill Team Member's tasks are completed as soon as possible.

#### 10.2.2 Inexperience with Tools

Not all Team Members will be experienced and comfortable with all tools used to complete the Project. It is the responsibility of the appropriate Technical Leader to ensure that Team Members are proficient at any tools necessary. Should a Team Member still have trouble, and it will affect the Project Plan due to slippage, the appropriate Technical Leader shall assist the Team Member, and if necessary, perform the task for them. To ensure this risk does not happen, time will be allocated on the Project Plan for training and learning new tools.

#### 10.2.3 Inexperience at Planning

The many tasks involved with the project will require estimates to be made of the time required for completion in order to produce the Project Plan. These times are only estimates however, and some will vary considerably from the actual time taken. To ensure this does not cause slippage, the Project Manager will allow extra time, as for Team Member illness. Also, the whole team will regularly be involved in planning, when the Project Manager brings the Project Plan to Team Meetings to discuss.

#### 10.2.4 Design Not Producing Product that Meets Requirements

It is important that the Design phase leads to a product which fulfills the requirements set out in the SRS. To mitigate this risk, a Traceability Matrix will be produced such that every point in the SRS has at least one corresponding point in the design documents. This will then show if requirements are missed.

However, the design may fulfill all the functional requirements and still be unacceptable. This could be due to performance problems. Appropriate prototyping should be conducted during the requirements and design phases to test design decisions if performance or other non-functional requirements are important.

#### 10.2.5 Insufficient Technological Knowledge

Particularly during the design phase, an understanding of how the technologies involved in the project work will be crucial to the project's success. The Technical Leaders are designated as the authorities on such matters, and Team Members should consult them when necessary. The Technical Leaders should perform extensive reading if necessary and ensure that no important details are missed.

#### 10.2.6 Difficulties Contacting Client

During the initial requirements phase, and also later, it is important that contact is made regularly with the Client. If the Client is unavailable, producing the SRS will be difficult or impossible, and so schedule slippage will occur. The Client should be queried as to their availability, and then any periods of unavailability can be worked around in the Project Plan. If even this proves impossible, the Supervisor should be consulted.

#### 10.2.7 Loss of Data

Because the data contained within the group's directory on the CS machines are the record and product of the project, it is important that great care is taken to ensure no data is lost. In addition to the nightly backups made by the CS administrators, nightly off-site backups will be made by the Backup Manager. This prevents against data loss in the case of a failure of either the CS machines or the Backup Manager's backups. The chance of failure of both of these is deemed to be low enough that any process to mitigate against it would be unnecessary.

Because backups are made of this location, Team Members should ensure that data is kept outside of this location for as little time as possible. This includes workspaces being used to make changes. Since changes will be regularly checked in, the CS nightly backups will be sufficient given the amount of data kept by Team Members in their home directories.

### 10.3 Schedule Control

It is expected that some deadlines will be missed, and the following procedure must be followed in such a case.

If it is missed by under a week, the Project Manager will review other upcoming deadlines and if necessary, reallocate resources. The Agenda for the next meeting must include a discussion of the delay.

If the deadline will be missed by over a week, the responsibility falls to the Risk Manager. This will be in cooperation with the Project Manager who will update the Project Plan as necessary. If the next scheduled meeting is more than three days away, the Risk Manager must call a meeting for the purpose of controlling the delay.

If a submission deadline can not be met using available resources, the Project Manager must organise an appropriate extension with the Supervisor.

### 10.4 Team Member Availability

If a Team Member knows in advance that he/she will be unavailable to work on the project in a given period, he/she must discuss this with the Project Manager, whose responsibility it is to schedule and allocate tasks around this obstacle.

It is understood however that a Team Member may be unexpectedly unavailable, e.g. due to sickness. In such cases, the Project Manager will attempt to make contact with the Team Member and estimate how long the Team Member will be unavailable for. If the estimate is less than one week, then the Project Manager must perform task reallocation so as to minimise deadline slippage.

If one of the following conditions occur, then the Risk Manager must call a Team Meeting to discuss measures to minimise deadline slippage, which may include task reallocation and temporary role reallocation.

- The Project Manager's initial estimate is greater than one week.
- The actual time spent unavailable is greater than one week.
- More than one Team Member is unexpectedly unavailable at the same time.
- Repeated and problematic Team Member unavailability occurs. This is defined as unavailability which causes schedule slippage.

## 10.5 Changes to SRS after Client sign-off

Two types of non-trivial change (where a non-trivial change is defined as one that affects the meaning of a requirement, and hence, the outcome of the end product) are to be considered.

1. Changes wanted by Team Members. These changes must be posted to the Team Mailing list, and also to the Client. If the Client approves the change, or approves with modifications, and the Team agrees with the changes at the next Team Meeting, then the change can be made. If the Client is opposed to the change, but the team wishes to pursue it, the Client Liaison Officer will organise a meeting with the Client.

Once the change is made, a footnote will be made at the appropriate heading, with a link to the URL of the Mailing List archive copy of the post detailing the change.

2. Changes requested by the Client. When the Client requests a change (by Email/Phone/Meeting) the team must approve the change at a meeting, or by Email if the Client's request is urgent.

The Document Manager will make the change requested by the Client and then follow the procedure for Changes wanted by Team Members.

After either type of change, a letter detailing the change will be sent to the Client. This letter must then be signed and returned by the Client. This acts as a record that the change was authorised by the Client. Alternatively, a meeting may be arranged with the Client, and the entire document, including the change, re-signed.

## 10.6 Technical Knowledge Risks

During development, lack of technical knowledge will often be an issue. Where foreseeable, the Project Manager must allocate time/resources to the learning of new technologies either directly by allocating a Team Member the task of learning material, or indirectly by taking this issue into account when deciding the development time of a task. This will also be detailed on the Project Plan.

It is understood however that people learn new skills and gain technical knowledge at different speeds, hence it is inevitable that Team Members will be unable to meet a deadline due to lack of technical knowledge.

Where an individual faces this problem, they should contact a Technical Leader for aid in their learning of the material. Where a Technical Leader has this problem, it should be brought up in a Team Meeting, where an appropriate strategy for learning the material will be discussed and decided on. This strategy could take the form of doing online tutorials, performing research, seeking help from the Supervisor or posting to mailing lists for help on the topic.

## 10.7 Information from External Sources

Information from external sources, e.g. news groups, mailing lists, External Reviewers, is not expected to be perfect. This information or advice may contain flaws, hence it should be considered critically before being taken into account. Notable exceptions include reference documents, e.g. by the maker of a product, which are assumed to be precise.

Note that this is not a strict process, rather it is considered part of professional development, and is encouraged in Team Members.

## 10.8 Conflict Resolution

If a member of the team wishes to make a complaint about another member, they should do so to the Project Manager. The Project Manager will then discuss the matter with both members (if necessary) and resolve the conflict. At the Project Manager's discretion, the Supervisor may be contacted.

If the complaint is about the Project Manager, it should be made to the Risk Manager, who will follow the same procedure as above for the Project Manager.

## 10.9 Risk Log

A Risk Log will be created, which will be maintained by the Risk Manager. It will be stored in the repository, in the `doc/problem` directory. It shall contain details of new risks which have been identified.

If any Team Member identifies a new risk, they must inform the Risk Manager who will then add it to the Risk Log. The Risk Manager is to be informed by an email, with the keyword `[risk]`.

The Risk Log will contain the following information for each risk:

**Risk:** A short description of the risk

**Date:** The date the risk was added to the Risk Log

**Cause:** A description of possible reasons why this risk could occur

**Impact:** A description of the impact on the Team and Project that this risk would have if it occurred

**Solution:** How the problems associated with the risk will be solved

## 10.10 Problem Log

A Problem Log will be created, which will be maintained by the Risk Manager. It will be stored in the repository, in the `doc/problem` directory. It shall contain details of any problems which have actually occurred.

If any Team Member identifies a new problem they must inform the Risk Manager who will then add it to the Problem Log. The Risk Manager is to be informed by an email, with the keyword `[problem]`.

The Problem Log will contain the following information for each risk:

**Problem:** A short description of the problem encountered

**Date:** The date the problem was added to the Problem Log

**Cause:** A description of possible reasons why this problem occurred

**Impact:** A description of the impact on the Team and Project that the problem had

**Solution:** Any strategies that could be put in place to prevent further problems of this nature occurring in the future

## A Changes

This section gives details of changes made to the SQAP after the first external review.

- **Date:** 2002-10-23  
**Change:** Made some minor changes resulting from a final review by the Supervisor. The following changes were made:
  - Changed Review Manager role to Internal Review Manager
  - Added that FCM user manual will be written
  - Added requirement for release notes to be produced as part of release procedure
  - Added description of location of Backup Manager’s computer

The version number was also incremented to 2.01

- **Date:** 2002-10-19  
**Change:** Performed the Document Manager’s final proofread. Made changes to correct spelling and grammatical errors, as well as the following changes:
  - Added alternative method of re-signing SRS with Client
  - Changed Coding Standard for SAM to allow `import` statements to be included in locations other than the beginning of files under some circumstances.

Most sections of the SQAP were changed, however most changes were trivial changes for spelling or grammatical reasons.

- **Date:** 2002-10-12  
**Change:** Performed the final walkthrough/review of the SQAP before the Document Manager’s final proofread. Changes made were mainly for clarity of expression and consistency, for example:
  - Adding references to other parts of the document to aid in understanding
  - Making sure the same term was used consistently (for example, “User Documentation” / “The User Manual” could prove confusing)
  - Updating roles to ensure the right people take credit for the things they have actually done
  - Updating submission dates
  - Adding in details about where to find each document/log that were missing
  - Adding in exceptions to where processes had not been followed (for example, weekly Supervisor meetings did not occur during the Midsemester break, which had been planned that way)
  - Added in the date when the Weekly Code Review process finished
  - Relaxing processes slightly if they had not been followed (for example, posting the weekly code review was changed from 9am to 11am)
  - Added extra verification process of two final readthroughs of each document (this is the first)
  - Added that the whole Team participates in project planning
  - Updated directory structure and what is on the website

Most sections of the SQAP were changed.

- **Date:** 2002-10-03  
**Change:** Added an extra requirement for Release Control (section 5.7) - that changes to code must be posted for Individual Review, even if the Weekly Code Review is in place, and that the Release Manager must be the one who reviews these changes. This is to ensure that no bugs creep into a release, and that the person most familiar with features knows exactly what goes in.

- **Date:** 2002-09-27  
**Change:** Modified coding standard for python to reflect what we are actually doing. Modified the requirements for docstrings for tests/stubs since for these files, it is not necessary to have such strict requirements since Happydoc will not be used on them. Coding Standard appendix was changed.
- **Date:** 2002-09-21  
**Change:** Changes were made to most sections of the SQAP as suggested by External Audit 3. Since most sections were modified, please refer to the audit response, `audit-response-3-adfox.txt` in the `audit` directory of the Team's home directory for details of changes made and reasons for changes.
- **Date:** 2002-08-31  
**Change:** The file naming conventions in section 9 (Tools) was removed, and a reference to Appendix 2 was added in its place so as to preserve single point of control. Also, the Linux kernel coding standard was added to Appendix 2.
- **Date:** 2002-08-27  
**Change:** The procedure for recording decisions made and critically appraising these decisions was added, after the midyear feedback report suggested it. New Decision section was added to section 5 (Standards). Also more detail was added to Design Notebook sections, as suggested by midyear feedback report.
- **Date:** 2002-08-24  
**Change:** Changes were made to all sections of the SQAP as suggested by External Review 2. Since all sections were modified, please refer to the review response, `review-response-SQAP-2.txt` in the `review` directory of the Team's home directory for details of changes made and reasons for changes.
- **Date:** 2002-08-15  
**Change:** Added a new role of "Release Manager" to conduct and control the software releases made by the Team. This was added after a request by the Supervisor for a stable preview version of the software. This role was assigned to Chris Edwards. Section 3.1 was changed.
- **Date:** 2002-08-14  
**Change:** Added some extra software packages being used for the project. They are: Python, Glade, Viewcvs, Pychecker, Happydoc, Kernel-doc. The use of these packages was only decided upon during the design phase and hence they could not be included earlier.
- **Date:** 2002-08-13  
**Change:** A general consistency check of the SQAP was performed before the second SQAP external review to pick up deficiencies and update the document. Changes include:
  - Updating document section for the SADD, TP and DDD
  - Relaxing requirement of a Team Review being the only item on the agenda (since it was found that this did not usually take up a whole meeting)
  - Enabling all Team Members to add tasks to the Task Log as long as they consult with the Project Manager (since often there are too many tasks for one person to add in)
  - Including a section on how the Project Plan is used by the Team (as suggested by Supervisor)

Most sections of the SQAP were modified.

- **Date:** 2002-08-02  
**Change:** The definition of 'non-trivial' change was clarified in the 'Changes to SRS after signoff' section, since it would be wasteful to have to contact the Client every time a small change was made to the SRS. Section 10.5 was changed.
- **Date:** 2002-07-28  
**Change:** The external review and audit procedure was changed as directed by the 4th year reviewers/auditors so that the actual review/audit reports are stored to facilitate marking. Sections 5.3.5 and 5.4.1 were changed.

- **Date:** 2002-07-24  
**Change:** A new procedure was added for Weekly Code Reviews (see section 5.3.4). This was introduced to replace the existing Individual Review procedure, which did not seem to work as well with code as it did with documents, due to the difficulty of understanding code as opposed to documents. A Review Manager position was also added, whose responsibility it is to organise the Weekly Code Reviews.
  - **Date:** 2002-07-23  
**Change:** The working procedure and meeting procedure that was followed by the Team during the semester break was added in, to make these procedures formal. Sections 5.1 and 5.2 were updated.
  - **Date:** 2002-05-18  
**Change:** The external audit procedure was updated as suggested in External Audit 1, to enable auditors access to the SQAP 2 days before the audit starts so they can ask for hard copies of documents and come up with an audit plan. Section 5.4.1 was updated.
  - **Date:** 2002-05-18  
**Change:** A guideline that files must be compiled before they are committed was added in. This is because people often forget to run latex, which can result in bugs in latex source entering the repository. Section 5.13 was updated.
  - **Date:** 2002-05-12  
**Change:** A description of the Problem Log was added (see Section 10.10). It now includes any new risks identified, and a process is included for when a new risk is identified. A new email keyword [risk] was added for this purpose.
  - **Date:** 2002-05-10  
**Change:** These changes resulted from the first External Audit. The following were changed:
    - Process for emailing ideas for meetings to the Meeting Chair was changed since it is optional, not mandatory.
    - Process for recording minutes at Client Meetings was changed since a dictaphone has proved more useful than two people taking minutes.
    - Supervisor Meeting process was updated since these regular meetings are only for semester 1.
    - A deadline for when an Individual Review must be completed was added in to complete this procedure.
    - A note was added to External Review process to say where the review report could be found to clarify this process.
    - Process for Internal Audits was clarified, and a deadline for the Audit Response was added in.
    - Procedure for archiving email when CS is down was added in.
- These changes resulted in Section 5 being changed.
- **Date:** 2002-05-09  
**Change:** The situations for when not to post for Individual Review were relaxed so that any first draft of a document may be committed without review, since the first draft is usually not important and is usually simply a template. Section 9.7 was updated.
  - **Date:** 2002-05-01  
**Change:** The 5 major documents were moved from the base repository folder to the doc folder to remove clutter from the repository. Section 9.8 updated to reflect this.
  - **Date:** 2002-04-27  
**Change:** Added in more reference to the Project Plan. This change was actually made before the External Review was completed, but only committed after the External Review was completed. The reason for this change was that there was not enough information about who modifies the Project Plan, where it is stored and what its purpose is. This information was added to sections 3 and 9.

- **Date:** 2002-04-25

**Change:** Added in a more detailed process for what happens after an external review. The reason for this addition is that it was noticed we took a long time to update the SQAP after the first external review of it. So, a deadline of a week (or another negotiated time) was added in and the process was detailed further. Section 5.3.5 was changed.

## B Coding Standards

### B.1 SAM Coding Standard

All code written in Python will conform to the coding standards described in this section. These standards have been based on two Python Enhancement Proposals - the ‘Style Guide for Python Code’ (<http://www.python.org/peps/pep-0008.html>) and ‘Docstring Conventions’ (<http://www.python.org/peps/pep-0257.html>).

#### B.1.1 File organisation

- All Python source files shall use “.py” as a suffix.

#### B.1.2 Indentation

- One level of indentation shall be one tab character, which is replaced by 4 spaces. All Python source files shall have the following line to ensure this: `vim: ts=4 sw=4 noexpandtab`:
- Line lengths shall be no longer than 80 characters. Longer lines shall be broken and indented one level.

#### B.1.3 Import statements

- The form of import statement used shall be: `from Library import Package1, Package2`, or `import Library` if the Library is too big and thus would pollute the namespace.
- Import statements shall be on separate lines.
- Import statements shall be before any functions or class definitions, unless the Library is only needed in a specific part of the code.

#### B.1.4 Attribute initialisations

Python does not require variables to be declared before they are used. However, this is detrimental to modifiability of classes - so, all attributes (private and public) of a class shall be initialised in the constructor of the class.

#### B.1.5 Blank lines

A blank line shall be used:

- Between methods
- Before and after import statements
- Before and after blocks of comments
- Within a method to separate sections to improve readability (if necessary)
- Before and after documentation strings (see B.1.9)

#### B.1.6 Whitespace

A single space character shall be used:

- Immediately after a comma
- Between all binary operators except ‘.’ and their arguments (except when an ‘=’ sign is used to indicate a default value or keyword argument)

### B.1.7 Naming standards

- Classes shall use the `CapWords` filenaming convention (all capital words).
- Methods shall use the `mixedCase` filenaming convention (first word lowercase, then the rest uppercase).
- Attributes shall use the `lower_case_with_underscores` convention (all lowercase words separated by underscores)
- Files shall have the same name as the highest level public class (or function) they hold, with a “.py” file extension.

### B.1.8 Comments

- Comments fall into two categories - documentation comments (see section B.1.9) and implementation comments.
- Implementation comments are used within the code to explain the implementation used and to aid in understanding.
- Implementation comments can be on the same line as a statement (separated by at least two spaces from the statement), or on a separate line.

### B.1.9 Documentation strings

- Documentation comments describe the purpose and application of the code without describing implementation details, and documentation strings (docstrings) shall be used for all documentation comments. This forms an Application Programming Interface (API) for the code, and is thus extremely important. This also forms the major part of the DDD.
- Docstrings shall be written for all classes and methods.
- The docstring shall be the first statement in a class or method definition.
- Triple double quotes shall be used around docstrings.
- The triple double quote that ends a multi-line docstring shall be on a line by itself.
- The docstring for a class shall contain, in this order, on separate lines:
  - A one line summary of the class
  - Purpose: Intended action
  - Used by: List all the classes or functions which call/activate this class
  - Uses: List all the classes or functions which are called/activated by this class
  - Attributes: List of all the public attributes (in most cases, this will be None, since private attributes are preferred since they promote information hiding)
  - SRS Refs: List of the references to the SRS
  - Author: Name of Initial author
  - Date: Date of creation
- The docstring for a function or method shall contain:
  - Purpose: Intended action
  - Used by: List all the classes or functions which call/activate this function/method
  - Uses: List all the classes or functions which are called/activated by this function/method
  - Input: Describe the format, type and order of the input parameters

- Output: Describe the format and type of the output parameters
- Preconditions: List assumptions made about the input
- Postconditions: List assumptions that can be made about the output
- SRS Refs: List of SRS references for the method
- Author: Name of Initial author
- Date: Date of creation
- For test cases of SAM written in Python, docstrings for a class need only contain:
  - Purpose: Intended action
  - Uses: List all the classes or functions which are called/activated by this class
  - Author: Name of initial author
  - Date: Date of creation
- For test cases of SAM written in Python, docstrings for a function or method need only contain:
  - Name: Name of the test case
  - Purpose: Intended action
  - Uses: List all the classes or functions which are called/activated by this function/method
  - Preconditions: List assumptions made about the input
  - Postconditions: List assumptions that can be made about the output
  - Author: Name of initial author
  - Date: Date of creation
- For stubs, drivers or other programs (eg scripts or data generators) written to assist testing, there is no formal requirement for docstrings.

Refer to the template `doc/templates/Template.py` for a specific example for source files.

### B.1.10 Happydoc

Happydoc will be used to automatically generate API documentation for the Python code. Happydoc uses docstrings, hence conventions for docstrings (described above) must be conformed to. Happydoc will be used with the default settings, outputting the documentation to a HTML file with the name *PYTHON-FILENAME\_CLASS.py.html*. The files will be stored in the subdirectory `src/sam/doc`.

## B.2 FCM Coding Standard

Coding standards for FCM shall follow the Linux kernel coding style, which can be found under the `Documentation` directory of the Linux kernel source tree in the file `CodingStyle`.

The following is a copy of the Linux kernel coding style:

### Linux kernel coding style

This is a short document describing the preferred coding style for the linux kernel. Coding style is very personal, and I won't `_force_` my views on anybody, but this is what goes for anything that I have to be able to maintain, and I'd prefer it for most other things too. Please at least consider the points made here.

First off, I'd suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it's a great symbolic gesture.

Anyway, here goes:

#### Chapter 1: Indentation

Tabs are 8 characters, and thus indentations are also 8 characters. There are heretic movements that try to make indentations 4 (or even 2!) characters deep, and that is akin to trying to define the value of PI to be 3.

Rationale: The whole idea behind indentation is to clearly define where a block of control starts and ends. Especially when you've been looking at your screen for 20 straight hours, you'll find it a lot easier to see how the indentation works if you have large indentations.

Now, some people will claim that having 8-character indentations makes the code move too far to the right, and makes it hard to read on a 80-character terminal screen. The answer to that is that if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program.

In short, 8-char indents make things easier to read, and have the added benefit of warning you when you're nesting your functions too deep. Heed that warning.

#### Chapter 2: Placing Braces

The other issue that always comes up in C styling is the placement of braces. Unlike the indent size, there are few technical reasons to choose one placement strategy over the other, but the preferred way, as shown to us by the prophets Kernighan and Ritchie, is to put the opening brace last on the line, and put the closing brace first, thusly:

```
if (x is true) {
we do y
}
```

However, there is one special case, namely functions: they have the opening brace at the beginning of the next line, thus:

```
int function(int x)
{
body of function
}
```

Heretic people all over the world have claimed that this inconsistency is ... well ... inconsistent, but all right-thinking people know that (a) K&R are *\_right\_* and (b) K&R are right. Besides, functions are special anyway (you can't nest them in C).

Note that the closing brace is empty on a line of its own, *\_except\_* in the cases where it is followed by a continuation of the same statement, ie a "while" in a do-statement or an "else" in an if-statement, like this:

```
do {  
body of do-loop  
} while (condition);
```

and

```
if (x == y) {  
..  
} else if (x > y) {  
..  
} else {  
....  
}
```

Rationale: K&R.

Also, note that this brace-placement also minimizes the number of empty (or almost empty) lines, without any loss of readability. Thus, as the supply of new-lines on your screen is not a renewable resource (think 25-line terminal screens here), you have more empty lines to put comments on.

### Chapter 3: Naming

C is a Spartan language, and so should your naming be. Unlike Modula-2 and Pascal programmers, C programmers do not use cute names like `ThisVariableIsATemporaryCounter`. A C programmer would call that variable "tmp", which is much easier to write, and not the least more difficult to understand.

HOWEVER, while mixed-case names are frowned upon, descriptive names for global variables are a must. To call a global function "foo" is a shooting offense.

GLOBAL variables (to be used only if you *\_really\_* need them) need to have descriptive names, as do global functions. If you have a function that counts the number of active users, you should call that "count\_active\_users()" or similar, you should *\_not\_* call it "cntusr()".

Encoding the type of a function into the name (so-called Hungarian notation) is brain damaged - the compiler knows the types anyway and can check those, and it only confuses the programmer. No wonder MicroSoft makes buggy programs.

LOCAL variable names should be short, and to the point. If you have some random integer loop counter, it should probably be called "i".

Calling it "loop\_counter" is non-productive, if there is no chance of it being mis-understood. Similarly, "tmp" can be just about any type of variable that is used to hold a temporary value.

If you are afraid to mix up your local variable names, you have another problem, which is called the function-growth-hormone-imbalance syndrome. See next chapter.

#### Chapter 4: Functions

Functions should be short and sweet, and do just one thing. They should fit on one or two screenfuls of text (the ISO/ANSI screen size is 80x24, as we all know), and do one thing and do that well.

The maximum length of a function is inversely proportional to the complexity and indentation level of that function. So, if you have a conceptually simple function that is just one long (but simple) case-statement, where you have to do lots of small things for a lot of different cases, it's OK to have a longer function.

However, if you have a complex function, and you suspect that a less-than-gifted first-year high-school student might not even understand what the function is all about, you should adhere to the maximum limits all the more closely. Use helper functions with descriptive names (you can ask the compiler to in-line them if you think it's performance-critical, and it will probably do a better job of it that you would have done).

Another measure of the function is the number of local variables. They shouldn't exceed 5-10, or you're doing something wrong. Re-think the function, and split it into smaller pieces. A human brain can generally easily keep track of about 7 different things, anything more and it gets confused. You know you're brilliant, but maybe you'd like to understand what you did 2 weeks from now.

#### Chapter 5: Commenting

Comments are good, but there is also a danger of over-commenting. NEVER try to explain HOW your code works in a comment: it's much better to write the code so that the `_working_` is obvious, and it's a waste of time to explain badly written code.

Generally, you want your comments to tell WHAT your code does, not HOW. Also, try to avoid putting comments inside a function body: if the function is so complex that you need to separately comment parts of it, you should probably go back to chapter 4 for a while. You can make small comments to note or warn about something particularly clever (or ugly), but try to avoid excess. Instead, put the comments at the head of the function, telling people what it does, and possibly WHY it does it.

## Chapter 6: You've made a mess of it

That's OK, we all do. You've probably been told by your long-time Unix user helper that "GNU emacs" automatically formats the C sources for you, and you've noticed that yes, it does do that, but the defaults it uses are less than desirable (in fact, they are worse than random typing - a infinite number of monkeys typing into GNU emacs would never make a good program).

So, you can either get rid of GNU emacs, or change it to use saner values. To do the latter, you can stick the following in your .emacs file:

```
(defun linux-c-mode ()
  "C mode with adjusted defaults for use with the Linux kernel."
  (interactive)
  (c-mode)
  (c-set-style "K&R")
  (setq c-basic-offset 8))
```

This will define the M-x linux-c-mode command. When hacking on a module, if you put the string `-*- linux-c -*-` somewhere on the first two lines, this mode will be automatically invoked. Also, you may want to add

```
(setq auto-mode-alist (cons '("/usr/src/linux.*/*\.[ch]$" . linux-c-mode)
  auto-mode-alist))
```

to your .emacs file if you want to have linux-c-mode switched on automatically when you edit source files under /usr/src/linux.

But even if you fail in getting emacs to do sane formatting, not everything is lost: use "indent".

Now, again, GNU indent has the same brain dead settings that GNU emacs has, which is why you need to give it a few command line options. However, that's not too bad, because even the makers of GNU indent recognize the authority of K&R (the GNU people aren't evil, they are just severely misguided in this matter), so you just give indent the options `"-kr -i8"` (stands for "K&R, 8 character indents").

"indent" has a lot of options, and especially when it comes to comment re-formatting you may want to take a look at the manual page. But remember: "indent" is not a fix for bad programming.

## Chapter 7: Configuration-files

For configuration options (arch/xxx/config.in, and all the Config.in files), somewhat different indentation is used.

An indentation level of 3 is used in the code, while the text in the config-options should have an indentation-level of 2 to indicate dependencies. The

latter only applies to bool/tristate options. For other options, just use common sense. An example:

```
if [ "$CONFIG_EXPERIMENTAL" = "y" ]; then
    tristate 'Apply nitroglycerine inside the keyboard (DANGEROUS)' CONFIG_BOOM
    if [ "$CONFIG_BOOM" != "n" ]; then
        bool ' Output nice messages when you explode' CONFIG_CHEER
    fi
fi
```

Generally, CONFIG\_EXPERIMENTAL should surround all options not considered stable. All options that are known to trash data (experimental write-support for file-systems, for instance) should be denoted (DANGEROUS), other Experimental options should be denoted (EXPERIMENTAL).

## Chapter 8: Data structures

Data structures that have visibility outside the single-threaded environment they are created and destroyed in should always have reference counts. In the kernel, garbage collection doesn't exist (and outside the kernel garbage collection is slow and inefficient), which means that you absolutely *have* to reference count all your uses.

Reference counting means that you can avoid locking, and allows multiple users to have access to the data structure in parallel - and not having to worry about the structure suddenly going away from under them just because they slept or did something else for a while.

Note that locking is *not* a replacement for reference counting. Locking is used to keep data structures coherent, while reference counting is a memory management technique. Usually both are needed, and they are not to be confused with each other.

Many data structures can indeed have two levels of reference counting, when there are users of different "classes". The subclass count counts the number of subclass users, and decrements the global count just once when the subclass count goes to zero.

Examples of this kind of "multi-reference-counting" can be found in memory management ("struct mm\_struct": mm\_users and mm\_count), and in filesystem code ("struct super\_block": s\_count and s\_active).

Remember: if another thread can find your data structure, and you don't have a reference count on it, you almost certainly have a bug.