PIANOS project plan

Group Linja

Helsinki 7th September 2005

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Joonas Kukkonen
Marja Hassinen
Eemil Lagerspetz

Client

Marko Salmenkivi

Project Masters

Juha Taina
Vesa Vainio (Instructor)

Homepage

`http://www.cs.helsinki.fi/group/linja`

Contents

1 Introduction

This section describes the project plan for the PIANOS project.

The project plan defines the timetable and the program to be developed, assesses risks, estimates the size and complexity of the program and introduces the project team and communication methods.

This document is used to manage the project. Also the customer and the project supervisor are able to track the project with this document since it is updated regularly.

1.1 Goals

The project's goal is to produce a program for the University of Helsinki's spatial data analysis research group. The program analyzes spatial data by using a prior distribution and a mathematical model to calculate the posterior distribution for a set of variables. Certain statistical measures are then calculated from the posterior distribution. The project must deliver by the 2nd of September 2005.

1.2 Current systems

The software is related to another currently in use, named Bassist: 'Bassist is a tool that automates the use of hierarchical Bayesian models in complex analysis tasks. Such models offer a powerful framework for modeling statistically complex real-world phenomena.' [Bass]

Currently the research group is using BASSIST to analyze spatial problems. BASSIST wasn't designed for problems with spatial dependencies. It's modeling language lacks features which would allow efficient definition of spatial problems. The goal is to concentrate on a small portion of spatial problems which can't be solved efficiently with BASSIST.

1.3 Version history

Version	Date	Modifications
1.0	27.05.2005	The reviewed version without modifications
1.1	01.06.2005	The final modified version

1.4 Overview of the project plan

1. Introduction	Describes this document and the program being produced.
2. Project organization	Describes the organization and people involved in this project.
3. Communication and data storage	Describes the communication methods that the project personnel use and the data storage methods used.
4. Risk analysis	Describes the risks involved in the project. Each risk has a probability and impact rating. The list is updated to show current risks.
5. Hardware and software requirements	Describes the working environment of the program.
6. Project schedule	Describes the phases, milestones and timetable of the project. The chapter gives an overall description of the software engineering process. The chapter also describes function points and a size estimate of the project.
7. Monitoring and reporting	Describes the ways the project is being tracked and documentation.
8. Quality assurance	Describes the measures taken to ensure the quality of the product.

2 Project organization

2.1 Interest groups

The project group

Name	Mobile phone	E-mail
Joonas Kukkonen	044-5500259	jkukkone(at)cs.helsinki.fi
Marja Hassinen	050-4914517	mhassine(at)cs.helsinki.fi
Eemil Lagerspetz	040-7605141	lagerspe(at)cs.helsinki.fi

The client: Marko Salmenkivi
The client's assistant and the project's technical advisor: Esa Junttila

The instructor: Vesa Vainio (Mobile phone: 040-5636966, E-mail: vainio(at)cs.helsinki.fi)

2.2 Responsibilities

Name	Role
Joonas Kukkonen	Project Manager
Marja Hassinen	Planning Manager
Eemil Lagerspetz	Technical Support Manager

Descriptions of the roles

- The Project Manager is the chairman of the group meetings. He represents the project group to the client, coordinates the work and composes the weekly reports.
- The Software Manager coordinates the planning of the software architecture and monitors the progress of the development.[1]
- The Planning Manager coordinates the schedule planning and observes the project's progress. She also controls the listings of the work hours and gathers statistics about them.
- The Quality Manager makes sure that the working habits of the group ensure good quality. Her responsibilities also include organizing the reviews and coordinating the test planning.[2]
- The Technical Support Manager takes care of the tools used in the project. The tools include the group's www-pages, the CVS version control system, the LaTeXpublishing system and the programming tools.

[1] The Software Manager will be chosen later; the former Software Manager left the project.
[2] The responsibilities of the Quality Manager have been handed over to the Planning Manager, since the Quality Manager left the project.

3 Communication and data storage

3.1 Weekly meetings

All of the important decisions concerning project matters are done in the weekly meetings. These meetings are carried out at the department of Computer Science. These are the formal, primary method of communication. The regular meetings are scheduled on weekdays throughout the project like this:

Mondays	13-15 in BK106
Wednesdays	12-14 in BK106
Fridays	14-16 in BK106

Extra meetings to be scheduled will be discussed during the meetings.

3.2 Other methods of communication

To supplement the weekly meetings the project has a web page on which all informally communicated but important information will be displayed; this page is located at 'http://www.cs.helsinki.fi/group/linja/'.
For purposes of internal communication the project members use a mailing list that contains everyone's e-mail address. The list has the address:
'ohtuv05-linja-list(at)cs.Helsinki.FI'.
Everything of crucial importance will be posted there and displayed on the project web page. Informal communication between project members is also carried out on other media, including irc, on the channel #ohtup_linja, and mobile phones.

3.3 Data storage

For data storage the project uses CVS, with the main repository on the CS computer system at '/home/group/linja/CVS/'. The project's files reside in the module 'project'. All modified files will be updated to that repository, and the dept. of CS's extensive backup system will ensure that the project data stays safe.

4 Risk analysis

This section deals with the risks involved in the project. Each risk is described and given probability- and seriousness ratings. Steps to counteract a risk are described after it. This list of risks will update as the project progresses and risks are dealt with. Newly found risks will be added to reflect the state of the project.

Each risk has a *name*, *status*, a longer *description*, *probability* and *seriousness* and some measures to prevent it and/or to minimize its effects if it comes true.

The *name* is a very short description of the risk. It should not be longer than a few words. Longer descriptions are placed in the risk's *description*-section. a risk's *Status* indicates how the preventive measures have been taken into effect. The status is one of: *Active* (The risk is waiting to happen), *Prevented* (The risk has been negated and cannot come true), *Mitigated* (Measures have been taken to reduce the probability of the risk), or *Realized* (The risk has come true)

The *Probability* of a risk can be *Low*, *Moderate* or *High*, where *Low* could be something like 0% to 10%, *Moderate* from 10% to 40% and *High* from 40% up. These are guidelines, of course the 'true probability' of a risk cannot be known.

A risk's *Seriousness* has similar values with the *Probability*. Here *Low* indicates that the risk, if it comes true, would mean a minor setback, say 1-2 days of more work. *Moderate* could be 2-4 days of extra trouble and *High* might mean even a week's additional work. Every risk includes preventive measures to take against it and a plan to minimize its effect if it comes true. Either of these may not be appropriate for all the risks; if so, the reason is indicated in its respective section. These are not comprehensive measures; more extensive prevention is advised. These are simple guidelines for actions to take.

4.1 Risk ratings

Overview of the rating system used:

Risk name	This is a very short description of the risk.
Status	This is the risk's status — one of 'Active', 'Mitigated', 'Prevented' or 'Realized'
Risk description	A longer description, maybe including an example case in a few words.
Probability	The likelihood of the risk coming true, on a scale of 'Low, Moderate, High'.
Seriousness	An estimate of impact on the project, 'High' being the most serious, 'Moderate' and 'Low' the other choices.
Possible prevention	Actions to take to reduce the likelihood of the risk or negate it altogether.
Minimizing the impact	Considerations as to how to proceed if indeed the risk comes true.

4.2 List of risks

4.2.1 Personnel related

Risk name	Temporary loss of work force
Status	Realized
Risk description	Someone can't take part in the project work for a short period of time. (For example someone gets ill.)
Probability	High
Seriousness	Low
Possible prevention	Schedule the work evenly for the work period. Minimizing stress.
Minimizing the impact	Keep the missing person up-to-date via e-mail or other communication methods. Delegate the missing person's duties among other workers.

Risk name	Permanent loss of work force
Status	Realized
Risk description	Someone must quit the project. (For example someone gets seriously ill.)
Probability	Low
Seriousness	High
Possible prevention	Document source code so that everyone else can understand how it works. Keep people informed on progress done with sufficient communication.
Minimizing the impact	Reconsider the limits of the project. Possibly drop some of the lower priority requirements.

4.2.2 Data issues

Risk name	CS network failure
Status	Realized
Risk description	The CVS or the Helsinki University servers are down, causing some data to be unreachable or inability to communicate.
Probability	Moderate
Seriousness	Moderate
Possible prevention	This is a project-independent matter.
Minimizing the impact	Everyone's CVS snapshots can be combined and a new repository can be easily built. In case of email failure, IRC can be used instead.

Risk name	Loss of project data
Status	Prevented
Risk description	Some of the group's files are lost. (For example because of a computer virus or physical damage).
Probability	Moderate
Seriousness	Moderate
Possible prevention	Use virtual workspaces and version control systems so that copies of the files will be available and backed up at a safe location. Effectively negates the risk.
Minimizing the impact	No reasonable means to minimize

4.2.3 Planning risks

Risk name	Delay
Status	Mitigated
Risk description	A part of the project is delayed.
Probability	Moderate
Seriousness	High
Possible prevention	Control the working hours of everyone. Plan the schedule with extra space for adjustment. Schedule enough time for every part of the project.
Minimizing the impact	Adjust the timetable. Re-evaluate the project limits. Work some more, if reasonable.

Risk name	Schedule pressure
Status	Active
Risk description	Time grows short, the schedule presses on. The project quality decreases.
Probability	Moderate
Seriousness	Moderate
Possible prevention	Plan the schedule with extra space for adjustment. Prioritize requirements so that at least the essential ones will be implementable in the given time frame.
Minimizing the impact	Re-evaluate the project limits, drop some of the lower priority requirements, if necessary.

Risk name	Evolution of requirements
Status	Active
Risk description	Requirements change extensively on-the-fly, altering plans and implementation.
Probability	Low
Seriousness	High
Possible prevention	Gather an exhaustive list of requirements for the requirements specification document, then freeze the essential requirements, planning for and implementing the requirements specified therein.
Minimizing the impact	Implement essential requirements, don't accept new essential requirements.

Risk name	Over-planning
Status	Active
Risk description	Extensive planning results in problems when it is discovered in the implementation phase that the plans must be altered to produce a functioning software.
Probability	Moderate
Seriousness	Moderate
Possible prevention	Don't get alienated from the implementation altogether; try planning solutions in practice sufficiently.
Minimizing the impact	Rationalize the plan, re-planning the parts in conflict with reality.

Risk name	Over-prototyping
Status	Active
Risk description	Testing everything in practice results in haphazard implementation and too little planning.
Probability	Moderate
Seriousness	Moderate
Possible prevention	Do enough planning to balance the prototyping.
Minimizing the impact	Concentrate on design for a while.

Risk name	Harder than expected
Status	Active
Risk description	A component is found to be harder to implement than expected.
Probability	Moderate
Seriousness	Moderate
Possible prevention	Prototype parts of the components to get a picture of their required internal complexity. Implement all the crucial data structures and algorithms in groups of people (opposed to people working alone).
Minimizing the impact	Implement the parts as described above.

Risk name	The problem is more extensive than expected
Status	Active
Risk description	The software is discovered to be too extensive to implement in the given time frame.
Probability	Moderate
Seriousness	High
Possible prevention	Analyze the problem thoroughly in the analysis phase.
Minimizing the impact	Implement only essential requirements, reducing the work to be done.

Risk name	Incompatibility
Status	Active
Risk description	Some parts of the software don't work with the other parts like they should.
Probability	Moderate
Seriousness	High
Possible prevention	Document component interfaces in a specific manner so that there is no chance of misunderstanding. Communicate with other group members so that everyone knows how the parts should fit together.
Minimizing the impact	Reimplement the parts by everyone's combined efforts.

4.2.4 Technical risks

Risk name	Fortran issues
Status	Active
Risk description	Problems with the Fortran compiler or the Fortran language.
Probability	Moderate
Seriousness	High
Possible prevention	Try other Fortran compilers in addition to the GNU Fortran compiler. Have project members study enough Fortran to understand how the language works.
Minimizing the impact	Change the Fortran compiler used. Try different structures in the language.

Risk name	Mis-implementation
Status	Active
Risk description	It is discovered that the product being built does not conform to the customer's idea of the software.
Probability	Moderate
Seriousness	Moderate
Possible prevention	Verification and Validation.
Minimizing the impact	Contact the customer and negotiate on further development.

Risk name	Wrong approach
Status	Active
Risk description	A problem in the technical level is being solved in the wrong manner. (For example: too inefficiently)
Probability	Moderate
Seriousness	High
Possible prevention	Implement all the crucial data structures and algorithms in groups of people (opposed to people working alone)
Minimizing the impact	Find the correct solution and reimplement the part.

Risk name	Lack of technical specification
Status	Active
Risk description	Critical concepts of the implementation have been left ambiguously defined.
Probability	Moderate
Seriousness	Moderate
Possible prevention	Use explicit definitions, avoid ambiguous words.
Minimizing the impact	Define the concept later on in specific terms, integrate the use of the concept to match the definition.

5 Hardware and software requirements

5.1 Hardware requirements

The software, once finished, will not pose any hardware requirements for itself. These requirements can be derived from hardware required by software that the program requires.

5.2 Software requirements

The software requires a functioning Fortran (90/95) compiler, so far we have planned for the open-source g95 to be this compiler. Other compilers may be used, and it is possible that the compiling can be left to the user for compliance with other compilers or environments without compilers.

The software is designed to run under a Linux-based operating system. Guarantees on other operating systems cannot be provided.

The requirements on processor speed of efficient running of the programs produced will depend heavily on the complexity of the problem a particular program was produced to solve. There are no guarantees of execution times on given processors. This may change as the project progresses.

6 Project schedule

The PIANOS project will follow the classical waterfall software development process. The software product life cycle consists of analysis, design, implementation and testing phases.

6.1 Phases

The analysis phase is centered around the software requirements specification (SRS). The analysis phase is complete when a software requirements specification is produced. In addition to the SRS, a general project plan including initial risk assessment and schedule are realized.

The design phase is focused on software architecture and testing design. Required deliverables are a software design document and a software testing document. Additional deliverables may include technical and user interfaces prototypes.

During the implementation phase the components and other features of the system are developed. At the end of this phase the software product is expected to be fully featured. Unit tests are run in accordance to the main components.

The testing and finishing phase concentrates on integration and system testing. User manual and release packaging are also produced during this phase. This phase is complete when the software and necessary documents are delivered to the client.

6.2 Milestones

In the software development process there are four major milestones. The milestones are scheduled at the end of each phase and they should be mostly complete[3] before moving to the next phase. Consequently, the progress of the project and the produced deliverables are reviewed when each milestone is reached.

Date	Deliverable
27.5.	Project plan (final)
10.6.	Software requirements specification (draft)
17.6.	Software requirements specification (final)
24.6.	Software design document (draft)
22.7.	Software design document (final)
19.8.	Software product (executable version)
2.9	Final software product (release)

[3] Strictly linear schedule is not required

6.3 Project size estimation

6.3.1 Function points

The function point method is used as the primary tool in size estimation. Size estimates are updated during the project according to available information.

Initial function point (FP) estimate can be obtained by considering data movements across application boundaries. The input of the application is likely to consist of a varying set of data and a possibly complex description of the dependencies between data attributes. The output of the application is likely to consist of a small number of characteristic values describing the data and a collection of samples based on the information provided by the model and the data set. A summary of the FP analysis is shown below.

	Simple	Average	Complex	Total
User inputs	3 * 3	2 * 4	0 * 6	17
User outputs	1 * 4	0 * 5	0 * 7	4
User inquiries	0 * 3	0 * 4	0 * 6	0
Files	1 * 5	1 * 10	1 * 15	30
External interfaces	0 * 5	0 * 7	0 * 10	0

At total: 51 function points

Simple user inputs include

- Some simulation parameters: number of iterations, number of burn-in iterations, the thinning (?) factor

Average user inputs include

- Some simulation parameters: proposal distributions to different parameters, preliminary values to different parameters

Complex user inputs include

- Some simulation parameters: the updating strategy

Simple user outputs include

- Samples from the posterior distribution

Simple files include

- The matrix describing which squares or cities are adjacent to each other

Average complexity files include

- The data (the observations and some other information - for example the observation rates)

Complex files include

- The model

Complexity adjustment factors

Each question is answered using a scale that ranges from 0 (not important) to 5 (very important).

1. Does the system require reliable backup and recovery?	0
2. Are data communications required?	0
3. Are there distributed processing functions?	1
4. Is performance critical?	4
5. Will the system run in an existing, heavily utilized operational environment?	?
6. Does the system require on-line data entry	?
7. Does the on-line data entry require ... multiple screens or operations?	?
8. Are the master files updated on-line?	?
9. Are the inputs, outputs, files, or inquiries complex?	5
10. Is the internal processing complex?	5
11. Is the code designed to be reusable?	3
12. Are conversion and installation included in the design?	0
13. Is the system designed for multiple installations in different organizations?	0
14. Is the application designed to facilitate change and ease of use by the user?	3

The sum of the answers is 21.

Now it's possible to calculate the FP estimate by using the formula

FP = count-total * (0.65 + 0.01 * answers) = 51 * (0.65 + 0.01 * 21) = 43.86

6.3.2 Lines of code

Lines of code (LOC) can be estimated based on the previous implementation of a similar system, Bassist. Since Bassist has approximately 12K lines of code and PIANOS is supposed to implement only a limited subset of the functionality available in Bassist, it is likely that PIANOS LOC will be around 2K - 6K.

Lines of code can also be estimated using function points. It is estimated that the average ratio when considering Fortran is 106 LOC / FP.

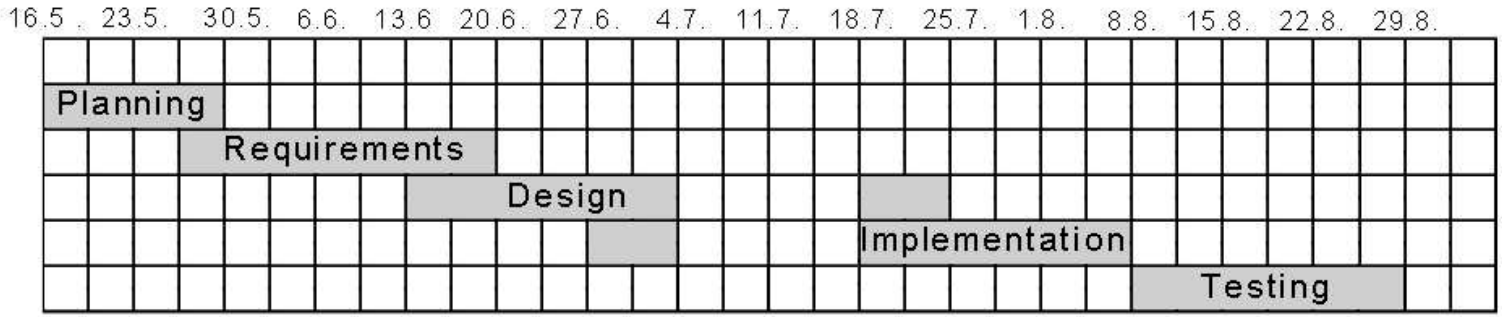
Using the FP calculation above the corresponding LOC estimate is about 4600 lines.

6.4 Project schedule

The project will run for 16 weeks, two of which are to be holidays. The holiday will be 4.7. - 17.7. (weeks 27 and 28).

Everyone should work at least for 240 hours, so the working rate should be at least 17 hours / week from the beginning.

The phases of the project are shown in the GANTT diagram at the end of this chapter.


16.5 .
23.5.
30.5.
6.6.
13.6
20.6.
27.6.
4.7.
11.7.
18.7.
25.7.
1.8.
8.8.
15.8.
22.8.
29.8.
Planning
Requirements
Design
Implementation
Testing

7 Methods of monitoring and reporting

7.1 Reporting working hours

All project group members will report their working hours in a file in the CVS. The file is /doc/reporting/hours/id.txt, where id is the member's user name in the CS computer system.

Working hours are recommended to be updated to the file daily. The absolute deadline is the next week's Monday, as they are required in the weekly report.

A program called Report.pl will be used to gather statistics about the working hours. For that reason, all the working hour reporting files must be in the following format:

- The first line is the member's name.
- Following lines are in the format 'date phase hours description'. The parts must be separated by a single tabulator.
- The file may include comments, those lines must begin with #.

The format for date must be dd.mm.yyyy, zeros in front of numbers may be skipped. For example, it's possible to write 23.5.2005 instead of 23.05.2005.

Phase is a four letter code of the phase on which the time was spent. The phase codes are:

PROJ	Project planning
REQU	Requirements analysis
DESI	Design
IMPL	Implementation
TEST	Testing
OTPR	Other deliverables (for example writing the user's manual or installation scripts)
KNOW	Getting to know the problem scope and environment
MEET	Meeting with the project group and preparing for the meetings
INST	Installing tools, learning to use them and maintaining them
OTHE	Anything that doesn't fall under the other categories

Hours are reported in half an hour's precision.

7.2 Weekly reporting

The group will produce weekly reports concerning the project's progress. They are written by Joonas, Marja will provide the working hour statistics. During the project, last week's report will be returned to the instructor by Tuesday morning.

A weekly report includes:

- Working hours during the week
- Tasks finished or agreed on during the week and all unfinished tasks
- Problems solved during the week and all unsolved problems

7.3 Reporting to the Software Engineering Project's information system

The Software Engineering Project has an information system where all groups must report some information about their projects.

The information system is located at http://db.cs.helsinki.fi/ tkt_ohtu/metrics/v0/index.php.

Every group member must report their working hours on the following dates:

- Monday 30.5.
- Monday 13.6.
- Monday 27.6.
- (Monday 11.7. - this is during the holiday)
- Monday 25.7.
- Monday 8.8.
- Monday 22.8.
- Friday 2.9.

Also some information must be reported about the project planning, requirements analysis, design and implementation phases. This will be done by Joonas. In all phases, the following dates are reported:

- When the phase started
- When the phase was supposed to end
- When the phase really ended

Additional information about project planning:

- The project's size estimates (lines of code and function points)

Additional information about requirements analysis:

- Number of requirements, use cases and words in the document
- The project's size estimates (lines of code and function points)

Additional information about the design phase:

- Number of operations designed according to the Requirements Specifications, operations changed and new operations
- Number of classes
- Number of pages, words, diagrams and pictures in the Design Document

Additional information about the implementation phase:

- Implementation languages
- Number of operations implemented according to the Design Document, operations changed and new operations
- The project's size estimates (lines of code and function points)
- Number of classes, methods per class, depth of inheritance hierarchy, number of immediate subclasses and connections between the classes

8 Quality assurance

8.1 Reviews

All documents will be reviewed by the project group and other appropriate interest groups. In addition to the documents, most relevant parts of the program code will also be reviewed.

A document or part of code will be frozen a few days before it's review. Between freezing and the review, no changes may be made to it.

Document	**Freezing date**	**Review**
Requirements specifications	13.6.	17.6.
Design document	18.7.	20.7.

Anni is responsible for organizing the reviews and she will prepare a checklist for the document to be reviewed by the freezing date. All participating to the review will read the frozen document in advance, looking for possible errors and comparing it to the checklist.

In the review, found errors will only be noted and written down. They will be assigned to someone to be fixed later, usually the person who wrote the part. The errors will not be discussed during the review, that way as many errors as possible may be found.

8.2 Coding style

The project group will follow a uniform coding style throughout the program.

- Java code will follow the standard convention for Java, described in [Java99].
- Fortran code will follow the convention proposed in [Fort01].

8.3 Testing

Testing is an integral part of the development of any software. It is a way to (1) prove in practice that the software meets the requirements it has been developed to fulfill. Also, (2) it verifies that the software works properly in the usual cases.

8.3.1 System tests

Tests of the former category are called *system tests*: They verify that the software, as a whole, meets the required criteria. System tests are, in a way, superficial: They don't concern the inner structure of the software, they only look at the input-output interaction and functionality. In specific efficiency matters they may measure the time it takes for a

function to complete. This enables us to design system tests as early as in the analysis phase.

8.3.2 Unit and integration tests

Tests of the latter category have two subcategories: *unit tests* and *integration tests*. The former ones check that individual parts of the software function as they were planned to, and the latter ones, integration tests, check that the parts function correctly with the other parts that they are planned to interact with. So, naturally, designing both of these tests requires the information brought by the design phase, where the internal structure of the software is planned.

8.3.3 Phases of testing

Any part should be tested before it is finished, and any part that is tested as functional is finished unless further requirements are set. The testing of individual parts is carried out in the implementation phase with the unit tests in this manner. Also, parts with common interfaces are tested with integration tests so that they function together in the expected manner. After the software is fully featured the system tests come in: They are to verify in the testing phase that all the requirements that were planned indeed are fulfilled. After the software passes the system tests it could be considered ready for release.

8.3.4 Testing practices used

The project group will use a standard testing practice for code in the Java programming language, such as JUnit. This will be decided later. For Fortran(90/95) another standard might be used or tests could be designed in a non-standard way if that better suits the case.
A *testing document* will be produced and updated as components are tested; The testing document will monitor the functionality implemented and that yet to be developed. The document will contain detailed information about each test, which part(s) of the software it is for, and in the case of system tests, the requirement that the test checks for.
The document lists bugs found and their status; these tell on a smaller scale about the status of different parts and the completeness of different requirements.
Each bug entry can have a status of:

Open	That is, discovered and not yet dealt with
Fixed	Found and fixed, but not reviewed by the one that reported the bug
Closed	Found, fixed and verified to have been fixed by the person who reported the bug

Bugs will be tracked and fixed in this manner through the course of the project.

9 References

Bass *Bassist: A Tool for MCMC Simulation of Statistical Models*
http://www.cs.helsinki.fi/research/fdk/bassist/

Fort01 Haataja, J., Rahola, J. and Ruokolainen, J., *Fortran 90/95*. Picaset Oy, Helsinki, 2001.
http://www.csc.fi/oppaat/f95/f95.pdf

Java99 *Code Conventions for the Java Programming Language*
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html