Software Test Plan

Swinbots - Swinburne RoboCup Team
SEP Group 1, 2004

Author(s):	Andrew Walker - 1262106
	James Fraser - 1261371
	Shane Harvie - 1262246
	David Reed - 1266659
	Luke Pitcher - 1262769
	Nickolas Wanke - 1229338
	Dejan Zigic - 1265636
	Michael Henderson - 1261487
Last Modified:	October 4, 2004
Version:	*Revision* : 1.17

Modification History Table

Date	Version	Description	Author
30/06/04	1.1	Added Content From Template	AW
30/06/04	1.2	Working furiously on the STP	AW
30/06/04	1.3	Filling in more sections	AW
02/07/04	1.4	Minor old changes	AW
05/07/04	1.5	Updated space requirements according to SQAP	DZ
18/07/04	1.6	Added to testing process	DZ
22/07/04	1.7	Added more detail	AW & DR
27/07/04	1.8	Updated Test plan	SH & DZ
27/07/04	1.9	Changed label for reference in section 4	LP
27/07/04	1.10	Updated the RoboCup Testing Tools	DZ
28/07/04	1.11	Editing changes	SH
28/07/04	1.12	Added automated tests section and figures	DZ
28/07/04	1.13	Updated diagrams and added dates to schedule	DZ
28/07/04	1.14	Added details of how requirements are to be tested	DZ
21/09/04	1.15	Post review modifications	DZ
02/10/04	1.16	Added abstract and intended audience	SH

Abstract

The testing plan of the 2004 Swinburne RoboCup project is presented in this document. It prescribes the scope, approach, resources and schedule of all testing activities. The plan identifies the items to be tested, the features to be tested, the types of testing to be performed and the risks associated with the plan.

Audience

The target audience for this document is:

- The Client — Dr Ali Bab-Hadiashar
- Project Team Members
- Project Supervisors
- Past and Future Team Members

Contents

1 Introduction **1**
- 1.1 Objectives . . . 1
- 1.2 Testing Strategy . . . 1
- 1.3 Scope . . . 1
- 1.4 Reference Material . . . 2
  - 1.4.1 Definitions . . . 2
  - 1.4.2 Acronyms . . . 2

2 Requirements to be tested **2**
- 2.1 Functional Requirements to be tested . . . 2
  - 2.1.1 RF Server . . . 2
  - 2.1.2 Embedded Software . . . 2
  - 2.1.3 Image Server . . . 3
  - 2.1.4 Robot Behaviour/Tactics . . . 4
  - 2.1.5 Tactics Debug Tool . . . 5
- 2.2 Performance Requirements to be tested . . . 5
- 2.3 Features Not To Be Tested . . . 6

3 Approach **6**
- 3.1 Unit Testing . . . 6
- 3.2 Build Testing . . . 6
- 3.3 Black Box Testing . . . 6
- 3.4 White Box Unit Testing . . . 7
- 3.5 Integration and System Testing . . . 7
  - 3.5.1 Substitutable Test Components . . . 7
  - 3.5.2 Functionality Scenario Walkthroughs . . . 7
- 3.6 Interface Testing . . . 7
- 3.7 Security Testing . . . 7
- 3.8 Recovery Testing . . . 7
- 3.9 Regression Testing . . . 7
- 3.10 Acceptance Testing . . . 8
- 3.11 Beta Testing . . . 8
- 3.12 Testing Procedures . . . 8
- 3.13 User and Operator Procedures . . . 8

4 Resources and Responsibilities **8**
- 4.1 Hardware . . . 8
- 4.2 Software . . . 9
- 4.3 Server Storage Space . . . 9
- 4.4 Boost::Test . . . 10

5 Risks **10**
- 5.1 Risk Mitigation Strategies . . . 11

6 Schedule **12**

7 Pass / Fail Criteria **12**
- 7.1 Approval Criteria . . . 12
- 7.2 Suspension and Resumption Criteria . . . 13

8 RoboCup Testing Tools **13**
- 8.1 Image Processing Calibration Tools . . . 13
- 8.2 Tactics Debug Tool . . . 13
- 8.3 Automated Tests . . . 13
- 8.4 Directory structure testing . . . 14

Appendices **18**

A Required "After Release" Changes **18**

List of Figures

1 Key for the following diagrams 15
2 Testing Process 16
3 All Building and Testing 17

1 Introduction

This Software Test Plan (STP) prescribes the scope, approach, resources and schedule of all testing activities. The plan identifies the items to be tested, the features to be tested, the types of testing to be performed and the risks associated with the plan.

1.1 Objectives

To identify problem areas of the RoboCup software system, and to isolate software which needs additional work.

1.2 Testing Strategy

The RoboCup team has decided to adopt both automated and manual testing processes. One of the aims was to attempt to automate as much of the process as possible, minimising the amount of time and effort required to run the tests. This allows team members to concentrate on analysis of results, and associated improvements of the system. Daily or weekly regression tests (depending on the component) will be performed on all stable and prototype code. Alongside this, the team members will complete unit, system, integration and performance tests to all prototype code before adding the code to the stable directory.

A major priority during test planning was to develop a reusable and extensible testing procedure, which produces test results which are both informative and efficient to analyse. Research on testing methods showed that the Boost organisation is leading the way in collection and presentation of regression test results (see `http://boost.sourceforge.net/regression-logs/`). Their methods served as a starting point for the design of our automated tests, and the presentation of results. This will be discussed further in 4.4.

The 2003 RoboCup team performed very little testing. They found it difficult to break the system into its components so that meaningful unit tests could be performed. This meant that integration testing (and associated debugging) was very difficult. Without confidence in the system components, it is very difficult to gain confidence in the system as a whole. With this in mind, the 2004 RoboCup team decided to produce a number of software tools to aid in unit testing. The extra time spent in development of these tools should result in more efficient testing, and better results.

1.3 Scope

Testing will be performed at several points in the life cycle as the product is constructed. Test planning is also an ongoing activity that will closely follow the construction of the product. As part of the planning, test check-lists must be developed for each level of product testing.

Testing of all functional requirements, as defined in the 2004 RoboCup SRS, will be performed.

Unit testing, integration testing, performance testing and regression testing will also be performed. The team will use a hybrid approach to testing, in that unit and integration testing will be performed in parallel. The reason for this choice is that the type of system to be developed utilises a “pipes and filters” structure (see SDD.pdf). The communication mechanisms (used for integration of the components) will be as important (and difficult) as the components themselves. As such, integration testing will be performed when the components are at a sufficient level of functionality.

1.4 Reference Material

SRS
Project Plan
SQAP
RoboCup Rules 2003
RoboCup Rules 2004

1.4.1 Definitions

Average propagation time: The time between image A being taken (by the overhead camera), and the receipt of the subsequent image (image B), showing the result of the command which was derived from image A.

1.4.2 Acronyms

GORGON – Swinburne Server, where all RoboCup web-site information and team statistics are located.

2 Requirements to be tested

2.1 Functional Requirements to be tested

The following are the tests to be conducted and the expected results produced by the functional requirements specified in the SRS (Refer to srs.pdf for details of the requirements):

2.1.1 RF Server

- **4.1 Receive robot instruction packet**
  Tactics Engine sends multiple types of data packets and upon successful operation the correct instruction should be available on the send buffer.
- **4.2 Broadcast robot instruction packet**
  Various instruction packets located on the send buffer will be available and upon successful operation they will be processed and transmitted to the robots, which will be evident by incorporating LED signals on-board the robots and by the return packets being sent by the robots.
- **4.3 Receive robot data packet**
  Once the robots receive instruction packets they will automatically send back a test packet to confirm the successful transmission. If the confirmation packet is successfully sent and received there should be a valid packet available in the RF Server's receive buffer.

2.1.2 Embedded Software

- **5.1 Receive robot instruction packet**
  The RF Server broadcasts multiple instruction packets and upon successful operation incoming packets will trigger interrupts, store the packets into the buffer, examine the packets, extract the data and execute the commands. Successful operation will be quite apparent with robots acting out the commands being sent.
- **5.2 Send robot data packet**
  Data packets are prepared by the robot (as mentioned, for test purposes once incoming packets are successfully received) and upon successful operation the data packets are sent to the RF Server and should be available in the receive buffer.

- **5.3 Stop**
  RF Server broadcasts a stop packet and upon successful operation the robot should halt all operations and wait for further instructions.

- **5.4 Reset**
  RF Server broadcasts a reset packet and upon successful operation the robot should halt all operations and reset all parameters to initial operating conditions.

- **5.5 Determine the required velocity for each wheel**
  The embedded software determines the wheel speed for each wheel in order to move to the desired position. Upon successufl operation, the robot will move to the desired position in a direct manner.

- **5.6 Control speed of a wheel**
  Upon successful operation the Embedded Software should read the motor encoder and determine the actual speed, use PID equations to compare this to the desired speed and set the wheel speeds accordingly. The successful operation will be evident if the robot reaches its desired location on the field.

- **5.7 Grab ball**
  RF Server sends a grab ball instruction which should result in the robot turning on the dribbler bar motor.

- **5.8 Determine whether ball is in possession**
  Once the ball is in possession, the relevant feedback device should alert the Embedded Software and as a result should create a possession packet and send the packet back to the RF Server. Upon successful operation the RF Server's receive buffer will have received a possession packet.

- **5.9 Shoot ball**
  RF Server sends a shoot packet to the robot and upon successful operation, and if the ball is in possession (determined in 5.8), the robot will process the packet and shoot the ball.

- **5.10 Pass ball**
  RF Server sends a pass packet to the robot and upon successful operation, and if the ball is in possession (determined in 5.8), the robot will process the packet and pass the ball.

- **5.11 Determine whether collision has occurred**
  The Embedded Software will poll the relevant feedback device and alert the Software once a collision has occurred. Upon successful operation the data should be analysed and sent to the collision response function.

- **5.12 Collision response**
  Receives a collision response message and processes the data. Upon successful operation the robot should move in the opposite direction to the impact for a small pre-set distance.

2.1.3 Image Server

- **6.1 Capture data from the camera**
  The camera is connected to the Image Processing and upon successful operation a filter graph should be running on the screen.

- **6.2 Input from the camera is received by DirectShow**
  Data arrives from the camera, and is placed through a sink filter and re-directed to the Image Server, which should then display the image.

- **6.3 Process image from the camera**
  Upon successful operation the image should be converted to the appropriate colour space and sorted by the relevant colours, and determine the important shapes on the screen, seen by a red outline.

- **6.4 Analyse shapes from the image processing stage**
  Once the regions have been identified, the image is analysed for the positions of the robots and ball, and the rotation angles of the robots. Upon successful operation the data gathered should be saved to a vector and then outputted to file, or steam data to client.
- **6.5 Client connection**
  The Image Server receives a control packet and upon authentication should send acknowledgement, receive place to stream, and store the client's team details.
- **6.6 Write to client**
  Upon successful operation a data packet should be transmitted to the client, which will also be written out to file.
- **6.7 Calibrate colours**
  An up to date image is supplied, whose colour parameters are altered until the image is at a decent standard, and the settings are saved. Upon successful operation the parameters should be saved to file.
- **6.8 Calibrate field area**
  An up to date image is supplied, whose area parameters are altered until the image is at a decent standard, and the settings are saved. Upon successful operation the parameters should be saved to file.

2.1.4 Robot Behaviour/Tactics

- **7.1 Stop robot**
  Tactics sends a stop packet to the RF Server and upon successful operation the stop packet should be processed by the RF Server and placed on the send buffer.
- **7.2 Move robot to set location**
  Tactics sends a position packet to the RF Server and upon successful operation the position packet should be processed by the RF Server and placed on the send buffer.
- **7.3 Stop all robots**
  Tactics sends a stop-all packet to the RF Server and upon successful operation the stop-all packet should be processed by the RF Server and placed on the send buffer.
- **7.4 Set goal direction**
  The direction at which the team is to be attacking is to be set, upon successful operation the correct direction is defined, which must be printed to be confirmed during testing.
- **7.5 Kick goal or shoot**
  Tactics will determine the best goal kicking orientation and send a shoot packet to the RF Server and upon successful operation the shoot packet should be processed by the RF Server and placed on the send buffer.
- **7.6 Kick to player - pass**
  Tactics will determine the best pass orientation and send a pass packet to the RF Server and upon successful operation the pass packet should be processed by the RF Server and placed on the send buffer.
- **7.7 Allocate role to robot**
  Upon successful operation each robot will be allocated a position on the field (Goalie, Attacker or Defender) which will be printed for confirmation during testing.
- **7.8 Select strategy**
  Tactics will analyse the game data received from the Image Server and upon successful operation store the allocated strategy for each robot for that particular frame.

- **7.9 Update robot according to strategy**
  Tactics will determine the robot positions according to the stored strategy and send the relevant packet to the RF Server and upon successful operation the packet should be processed by the RF Server and placed on the send buffer.

- **7.10 Get strategy**
  Upon successful operation the Tactics should retrieve the specified strategy from file according to the current position and send the packets accordingly as per function 7.9.

2.1.5 Tactics Debug Tool

- **8.1 Display visual representation of environment**
  Upon successful operation a 2-D plan view of the field, robots and ball should be available on the screen.

- **8.2 Display visual representation of team objective data**
  Tactics Debug Tool should retrieve environment data and upon successful operation should re-create the team decisions and display on the visual representation of the environment described in function 8.1.

- **8.3 Display visual representation of robot decision data**
  Tactics Debug Tool should retrieve environment data and upon successful operation should re-create the robot decisions and display on the visual representation of the environment described in function 8.1.

- **8.4 Update to next frame**
  Upon successful operation the data from the next frame should be displayed on the visual representation of the environment described in function 8.1.

- **8.5 Update to previous frame**
  Upon successful operation the data from the previous frame should be displayed on the visual representation of the environment described in function 8.1.

2.2 Performance Requirements to be tested

- **Image processing is to produce more than 20 frames-per-second.**
  Each time a new frame is created it will be added to a counter and at the end of transmission an average image processing time is to be printed, which should be at least 20 frames-per-second.

- **The Image processing is to be accurate to within 5 pixels.**
  Images produced by the field area calibration tool are to be compared in size to their actual physical size and should be accurate to within 5 pixels.

- **The RF server is to operate with a latency of less than 20ms (time between packet sent and valid data available for use).**
  All packets sent via the RF Server are to be timed and should have a latency if less than 20ms.

- **Packet loss in the RF should be less than 50%.**
  A count will be put in place to sum the number of packets sent, and at the end of transmission a total number of packets received by the robots will be sent back to the RF for comparison, intern there should be less than 50% packet loss.

- **The average propagation time less than 150ms.**
  Times will be recorded on both the RF Server and Robot to ensure that the total propagation time is less than 150ms.

- **Movement to a specified point on the field with IP feedback will be accurate to within 50mm.**
  RF Server will send multiple position packets to specifically marked locations (such as the corner of the goal box or the centre circle) and measurements will be taken which should be within 50mm.

2.3 Features Not To Be Tested

This Software Test Plan does not consider the testing of the hardware involved in the RoboCup system — however it does cover all software required to run such hardware, including the Embedded Software and communications software.
Build testing of documentation, although conducted will not be considered a formal part of the test process.

3 Approach

3.1 Unit Testing

A full list of the current unit tests, sorted by components for the entire RoboCup project will be maintained at http://mercury.it.swin.edu.au/hit3058_01/tests/out.html. Automated unit tests will be conducted using the boost::test framework available from http://boost.org. The test cases should include tests of all pre and post conditions as well as expected error conditions. Manual unit tests (those that require more subjective analysis of the test results), will have testing checklists that contain the following information:

- Description of test case
- Relevant input data
- Date tested
- Pass criteria
- Test result (pass/fail)
- Comments (optional)

3.2 Build Testing

The build tests will be maintained at http://mercury.it.swin.edu.au/hit3058_01/tests/buildlog.html and will list the results of compilation of all software components in all configurations. The build logs will be updated daily. This will enable detection of CVS update anomalies, such as failure to include a file from a team members' home PC.

3.3 Black Box Testing

Black box testing will test the standard usage of a component. Each non-GUI component should provide black box tests as a minimum. Tests should be repeated for input data that represents good, bad and boundary cases.

3.4 White Box Unit Testing

White box testing will be performed for some components — depending on the complexity of the code. Strong emphasis is to be placed on mathematically intensive operations such as the implementation of the Tactics potential field system.

3.5 Integration and System Testing

3.5.1 Substitutable Test Components

The RoboCup software has been designed using the pipes and filters pattern. We can use this to our advantage in testing by replacing components (or 'filters') with an equivalent test component. An example of this is the ProxyCam, which is a stand-in for the Camera class which provides the same module interface, but loads images from a data repository.

3.5.2 Functionality Scenario Walkthroughs

Many units, when fully integrated, have functionality that is difficult to test exhaustively in a 'Correct/Incorrect' fashion. In these cases 'scenarios' will be made to test usage of the system. They should cover both normal usage and possible abnormal usage.

If scenarios are required to test functionality, they are mentioned under the appropriate requirement in Section 2.1

3.6 Interface Testing

Interface testing will be conducted by a team of robotics students with an understanding of the problem domain, but no existing experience with the current software. Refer to the UID for further details.

3.7 Security Testing

No significant security requirements constrain the RoboCup project. The BlueTooth hardware encryption has been tested, and has passed. As no requirements in the SRS directly refer to security, no further security testing will be performed.

3.8 Recovery Testing

Recovery testing will be a manually performed test on the critical PC-side components only. Applications which crash with an exception should log the error message, and then restart. Communications connections should then be reset. This is not considered to be critical functionality — software which crashes on presentation day can be restarted with no repercussions.

3.9 Regression Testing

All automated unit, system, integration and performance tests will be performed on a daily basis, as part of our automated test tracking procedure.
Non-automated tests will only be conducted when a dependency is modified.
Automated Regression results and the most up-to-date non-automated test results will be used to identify ongoing patterns in software components at weekly team meetings.

3.10 Acceptance Testing

Acceptance testing will be conducted as a presentation to the client prior to the release of the product. Acceptance testing will be conducted by the client prior to the delivery deadline, and prior to the preparation to move to the release site on the 26th of August 2004. Acceptable performance will be determined by the client, but should include the three major criteria for assessment of the RoboCup project:

- Implementation of the 'go to' function (move a robot directly to a field position)
- Implementation of the 'kick to' function (robot kicks the ball to a field position)
- Implementation of the 'block' function (robot block an incoming ball, for goal keeper)

3.11 Beta Testing

The software from the RoboCup project will be used internally for the RoboCup project team, and as such a public beta will be unnecessary. If a public beta version was to be released, the time required to train an adequately large user community would be prohibitive.

3.12 Testing Procedures

Testing procedures will be created for complex subjective tests, which outline the test cases, the test method, and the expected results (including pass/fail criteria). They will be placed in the STP directory (robocup/docs/STP).

3.13 User and Operator Procedures

User procedures will be integrated into both the user interface testing for general users of the system, and into the user acceptance testing for the client. As such, it will not be conducted formally.

4 Resources and Responsibilities

The Testing Manager will be part of the Code QA team, and should attempt to attend all code reviews. They must ensure that all software development sub-teams are performing adequate testing throughout the development process.

Each team member is responsible for the testing of their code before moving it to a stable directory. If any team member is having difficulty with the amount of testing to be conducted, the Testing Manager will be notified as soon as possible and additional resources will be allocated to assist in the testing process.

4.1 Hardware

An IEEE-1394 compliant Firewire camera will be attached to the Firewire PCI.

An RF transmitter will be connected to a serial port and will use the RS232 protocol.

Daily automated unit tests will require approximately ten minutes of PC time, likely to be scheduled to run at 12pm on the shared team PCs.

The minimum system requirements to perform all tests are:

- Pentium 4 1.8 GHz (equivalent or above).
- 512 Mb RAM. (768 Mb Recommended)
- 1 Gb free hard disk space.
- SVGA Monitor capable of 1024x768 resolution.
- Graphics card with 16 Mb Memory
- Network Interface (Card or Built in)
- Firewire Interface (Card or Built in)
- 2 Serial Ports

4.2 Software

Documentation for the software tools is available in the RoboCup Tools document, which is stored in the repository as `robocup/docs/misc/tools.tex`. The following software will be required for testing:

- Microsoft Windows 2000
- Firewire drivers, including all other drivers required to list a camera as a video capture device
- OpenCV Libraries (Microsoft's Computer Vision Libraries)
- Microsoft Visual Studio .NET 2003
- Boost libraries
- Python 2.2.3
- ElementTree extensions for python
- Win32 Extensions for python
- Gnuplot.py
- Gnuplot
- Numeric Python
- NTGraph OCX
- pscp
- plink

4.3 Server Storage Space

Database storage is required to maintain the regression history for all tests. Some additional storage space for the build logs and test results will be maintained on the Swinburne GORGON server.

4.4 Boost::Test

The boost::test testing package will be used to run the unit tests for the RoboCup project. Integration, system and regression tests will be composed of compound tests constructed with the same framework. Refer to the SQAP for more detailed information on the management of tools.

The Boost Test Library provides a matched set of components for writing test programs, organizing tests into simple test cases and test suites, and controlling their runtime execution. Because the Boost Test Library is critical for porting and testing Boost libraries, it has been written to be conservative in its use of C++ features, and to keep dependencies to a bare minimum. The following Boost tools will be utilised during testing:

- The Execution Monitor - a basic exception and error detection and reporting facility for use in both production and test programs. The Execution Monitor calls a user-supplied function and reports all caught runtime exceptions. It is used internally by other Boost Test Library components. It could also be used in some production environments to make controlled calls of functions which might otherwise crash the program.
- The Program Execution Monitor - a simple helper facility to be used to monitor a program execution. The Program Execution Monitor provides the function main() and uses the Execution Monitor to control a program execution. It should be used in a production environment to produce uniform error reports. To control programs working in a test environment, use the Test Execution Monitor instead.
- The Test Tools - a toolbox for various testing needs. Test Tools are used by test programs working under control of the Test Execution Monitor or the Unit Test Framework.
- The Test Execution Monitor - causes a test program to run in a monitored environment. The Test Execution Monitor provides the main() function to control a simple test program execution and allows use of the Test Tools to implement test logic. It is intended to be used in a test environment. To control production code execution, use the Program Execution Monitor.
- The Unit Test Framework - a framework that simplifies writing and organising test cases. The framework supports test cases written as simple free functions or member functions and organises them into a tree of test suites. The framework allows use of the Test Tools to implement test cases and provides a facility to manage a log report level and a result report level.

As of boost 1.31.0, some of the XML output from boost::test doesn't produce well formed XML, which is expected by the outputs of the individual test runs. To compensate for this a customised formatter to produce well formed XML is required (this is especially important in C++ where &, ¡ and ¿ are significant characters).

Similarly, boost::test currently has a problem linking correctly under Microsoft Windows which requires a patch to be placed within each test case.

5 Risks

The general project risks, as defined in the Project Plan (PP.pdf) apply to the Testing stage of the project. Some risks which are specific to the testing process are shown below.

Risk: Faulty Equipment There exists the risk that equipment for some parts of the project may be defective. The microcontroller boards and RF equipment are particularly susceptible to this risk.

Risk: Power Outage An extended power outage at Swinburne University would place significant strain on the project.

Risk: Data Loss Computer system failure or malfunction often results in data loss. In the case of the RoboCup project, losses are most likely to occur through failure of computers in the lab, or a failure of mercury. Delays are to be expected whilst the system is being fixed or data is being reloaded/re-entered.

Risk: Rule Changes Major changes to the rules of RoboCup could require the team to redesign hardware and/or make substantial modifications to software. Such modifications would impose a cost, in terms of both money and time.

Risk: Internal Conflict If team members have problems with each other, and any such conflicts aren't dealt with appropriately, time could be wasted and the schedule would be adversely affected.

Risk: Team Member Absence If a team member is sick, absent or otherwise unable to perform his duty, any tasks assigned to this member are likely to be left incomplete. Depending on the importance of the team member, and the period of absence, significant project delays are possible.

Risk	Uncontrolled			Controlled		
	Likelihood	**Impact**	**Exposure**	**Likelihood**	**Impact**	**Exposure**
Faulty Equipment	4	6	24	3	4	12
Power Outage	9	2	18	9	2	18
Data Loss	7	8	56	1	8	8
Rules Changes	8	5	40	8	1	8
Internal conflict	8	5	40	6	4	24
Team Member Absence	8	6	48	6	4	24

Table 1: Identified Risks

5.1 Risk Mitigation Strategies

Mitigation: Faulty Equipment Supplied hardware should be tested immediately following delivery. When possible, equipment problems should be fixed by the team. Any problems requiring external attention are to be addressed by the affected sub-team and relevant leaders, who must organise for faulty equipment to be sent out for repair.

Mitigation: Power Outage Should an extended power outage (more than 2 days) fall upon Swinburne University, the team will assess the impact on the project and, depending on this impact, re-assess the scope of the project with the client. Reduction of this scope may allow completion of the project despite any setback caused by the power outage.

Mitigation: Data Loss Data loss will be minimised through the use of a CVS repository stored on the Swinburne server (mercury). Mercury is backed-up regularly, so any data losses will be minimal. However, team members are required to regularly checkout the repository onto their home machines, hence increasing redundancy in the backup 'chain'. In case of failure of the lab computers, they have been imaged onto the Swinburne network, hence speeding-up recovery times. The use of multiple computers in the lab also minimises the likelihood of a failure causing critical problems for the project.

Mitigation: Rule Changes If the rules for the RoboCup competition change significantly for 2004, the project will operate under as many of the new rules as is possible with the current hardware, but will ignore any other rules. Since the 2004 team isn't planning on competing with other universities this year, failure to meet the rule requirements will not be catastrophic.

Mitigation: Internal Conflict Internal conflict will be addressed by organising, as a 'relief', team events which are not purely focused on project activities; for example: a team dinner or drinks at the local pub. The Team Leader will monitor the number of hours each team member is logging, and will ensure that over-extended team members get some sort of break from too much project work. Other conflict resolution strategies are mentioned in the "Conflict Resolution Process" (see PP.pdf for details)

Mitigation: Team Member Absence Team member absence is only a major risk if the team member is significantly important, or if they are absent in the long term. Short-term absence will be covered by spreading residual tasks across the whole project team. Long term absence will likely require a re-evaluation of team goals, as personnel resources are already stretched. Modification of these goals may be required to ensure that the team goals are realistically achievable.

6 Schedule

The official team schedule, including testing milestones, can be found in the CVS repository `robocup/docs/schedule`.

- User Interface Testing is to be performed during weeks 2 and 3 of Semester 2 (2nd August 2004 - 13th August 2004).
- Unit test cases are to be defined before commencement of code. Testing is to be commenced as units are declared stable (from week 6 of semester 1 (29th March, 2004)). It will be conducted daily from the 7th of August, 2004.
- Performance testing will be performed after integration of components has been completed. This is planned for the week beginning 15th August, 2004.
- Regression testing will be conducted daily from the 7th of August, 2004.
- Integration testing - 18th July until 18th of August, 2004.
- System Testing - 12th August to 25th August, 2004.
- Acceptance Testing - 20th of August.

7 Pass / Fail Criteria

7.1 Approval Criteria

Each item for approval must initially pass a review consisting of at least two other team members (in addition to the author). The item must consist of properly documented code, samples (if appropriate), tests and any basic performance measurements that may be required. If the item is passed at review, or passed with modifications, it has one week to be rolled into the stable directory. In addition to the items mentioned above, the module must also conform to the following standards:

- no files contain tabs
- compile cleanly in all configurations
- be non-reliant upon code that has not been reviewed
- samples run in all configurations (if appropriate)
- tests run in all configurations
- documentation meets team standards
- code meets team standards
- files which have undergone substantial work must have revision histories

7.2 Suspension and Resumption Criteria

Any item which is substantially modified from the time that it was reviewed, should be re-reviewed. Re-reviews of code should be conducted no more than once a week. Code which does not comply with these guidelines will be rolled back, with the latest version moved into the prototypes folder (suspension). A successful (passed, or passed with modifications) review on suspended code will allow it to be resumed in the stable folder.

8 RoboCup Testing Tools

As mentioned earlier, the lack of testing performed by the 2003 RoboCup team made testing in general quite difficult. Without dividing the system into components, simple Unit testing proved an enormous task due to the complexity and intertwining nature of the code. Once it was time for integration testing, it was simply ignored due to the rush to complete the product.

The 2004 RoboCup team decided to design testing software from the beginning to avoid the complexities faced in 2003. The first step was to divide the software components into three major groups (Image Processing, Tactics Engine and Embedded/RF). The team then analysed the components and came up with the following tools to assist the unit and integration testing procedures described in the above sections:

8.1 Image Processing Calibration Tools

A number of tools were developed to reduce the excessive time needed for accurate calibration for the image processing system. Three major areas needing calibration are: areas of interest, colours, and camera parameters. By speeding up the process and improving the accuracy of calibration, integration and integration testing will be more successful and accurate.

Colour Calibration and Field Area Tool
The Colour Calibration Tool allows the user to set threshold values for the colours being analysed in the image data. By expediting this process, and allowing the user to see the spread of data in visual form, the set-up times for the colour calibration will be significantly reduced. The integration of the Field Area Tool User Interface will also allow the user to select the field area which is relevant to the data being analysed. The field area tool allows a user to select the areas of the field which are required, both during game play and during colour calibration.

8.2 Tactics Debug Tool

The Tactics Debug Tool was created to enhance a given Tactics Rules module. It is a GUI based system which will map the world data — that is the position of the team and opposition robots, and the ball — with a visual representation of the Tactics Rules. That way, if an undesirable decision is made, the tester can see why this decision was made (through the visual representation of the Rules Decisions), and go about trying to change the Rules such that a more desirable decision will be made.

8.3 Automated Tests

To further improve and assist the testing process, the 2004 team decided to implement several automated tests. As described above in section 4.4, unit, integration and system testing of most code will be automated via the Boost testing process. Document building and code building will also be an automated process to ensure that consistent and correct methods are being used by all team members. This also

allows a record of all complete builds to be stored so that they may be audited for correctness and compliance.

To track all of the automated (and non-automated) testing and building procedures, the 2004 team opted to store all of the test and build information onto the GORGON server as web pages. This allows all of the team members, and interested academic staff, to access the information.

The automated testing procedure is shown in detail in figure 2. The steps shown in figure 2 are as follows:

1. The automated unit, integration and system Boost tests are triggered.
2. The Boost tests compile the relevant code and produce executables.
3. The executables are run and XML test results are produced.
4. With the aid of Python, both HTML and SQL results are recorded.
5. The HTML results are automatically added to the GORGON server.
6. The SQL statements are uploaded to the RoboCup team Database (Swinburne's SQL platform), which holds all of the necessary test data that the team wishes to filter and sort.
7. The Robocup team Database is queried for all of the relevant test results over a time span and relevant information saved to a log file.
8. The log file transfers the information to HTML and GNUPlot (plotting language).
9. The HTML reports and graphs produced from the relevant test results are also uploaded onto the GORGON server.

Also shown on figure 2 are the non-automated test results, which are also added to the GORGON server for tracking and auditing purposes.

Figure 3 shows all of the automated and non-automated testing and building procedures that the 2004 RoboCup team has developed. These include static testing (manually performed tests), document building, code building and automated testing. As shown in the figure, all of the results are uploaded onto the GORGON server.

8.4 Directory structure testing

It was identified that the output and intermediate directory structures for most of the projects created in Visual Studio .NET were not identical since each system component was created individually. Adding to the complication is the fact that the directories have different modes of operations (Release and Debug), making it a tedious task to ensure every project conforms to the standards. This creates issues during integration. As such, the team decided that the most efficient means of resolving this was through an automated program which checks the project settings, and makes any required changes. Python was used to recursively sift through every directory in the RoboCup structure and source all of the project files. Once all the project files were identified, they were checked for conformance to the standard output and intermediate directory structure. Those that do not conform are automatically updated allowing integration to be made possible.

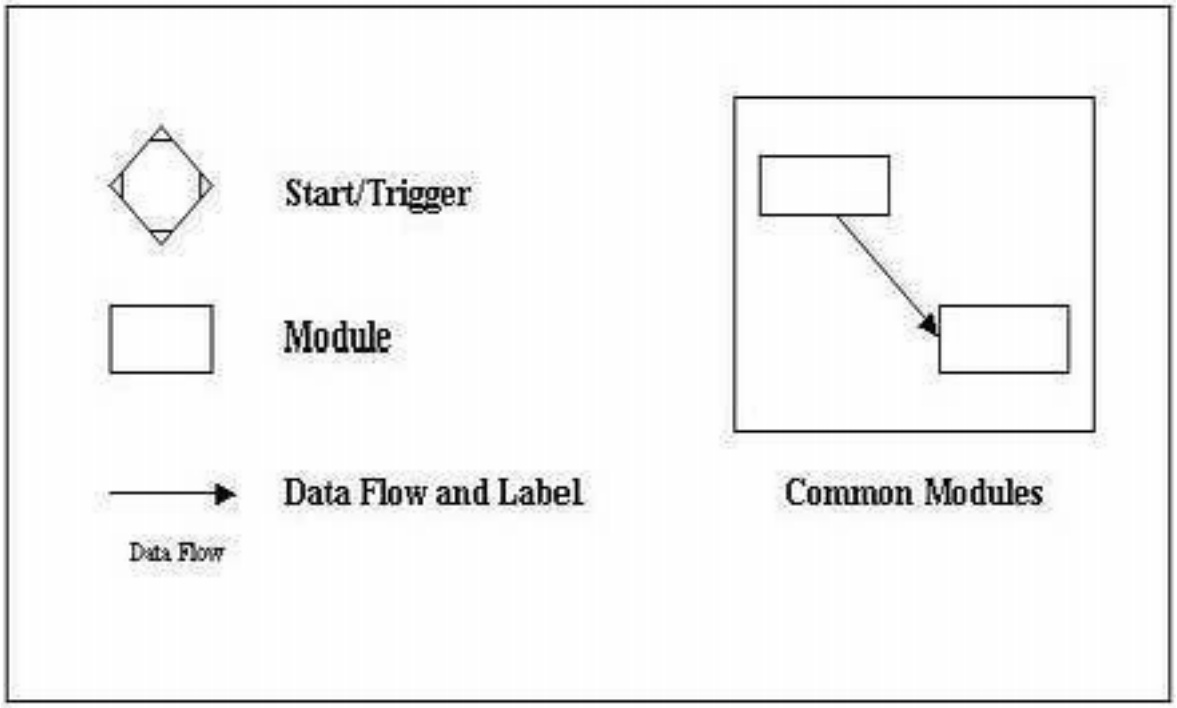

Figure 1: Key for the following diagrams

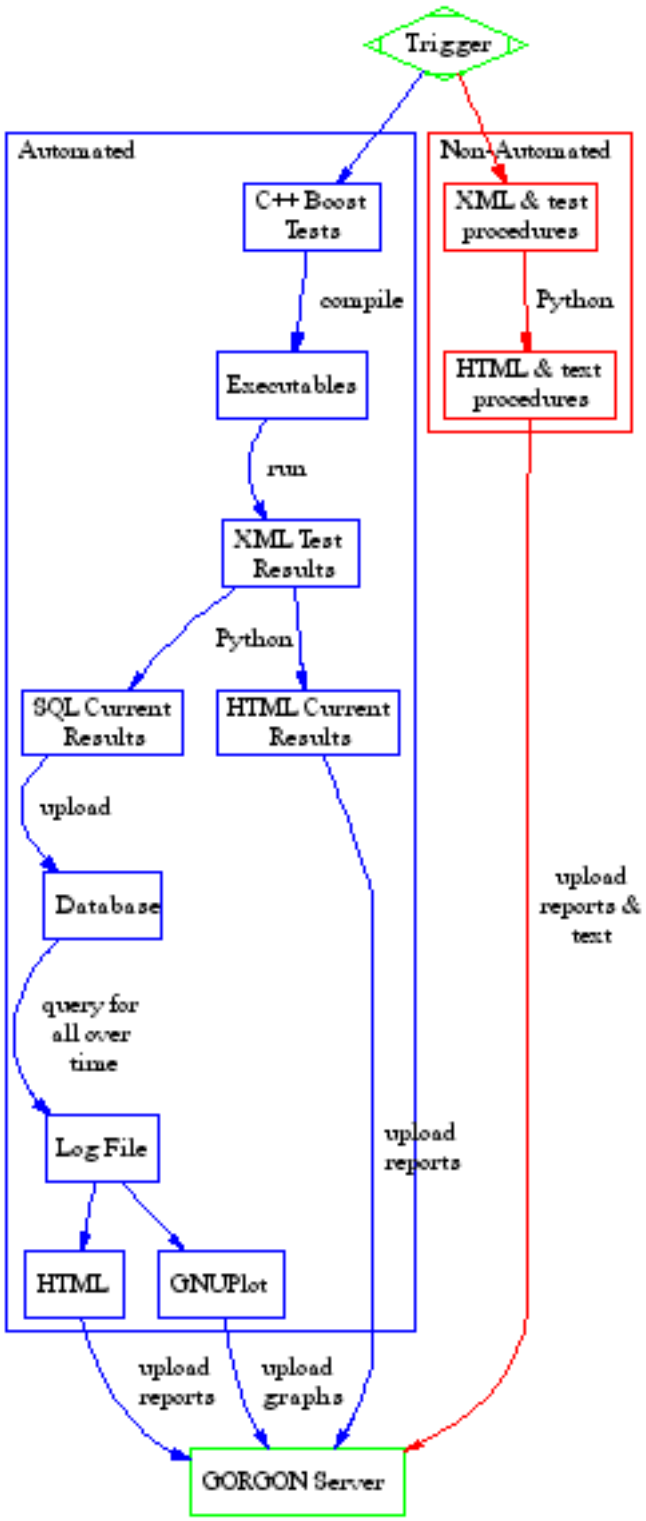

Figure 2: Testing Process

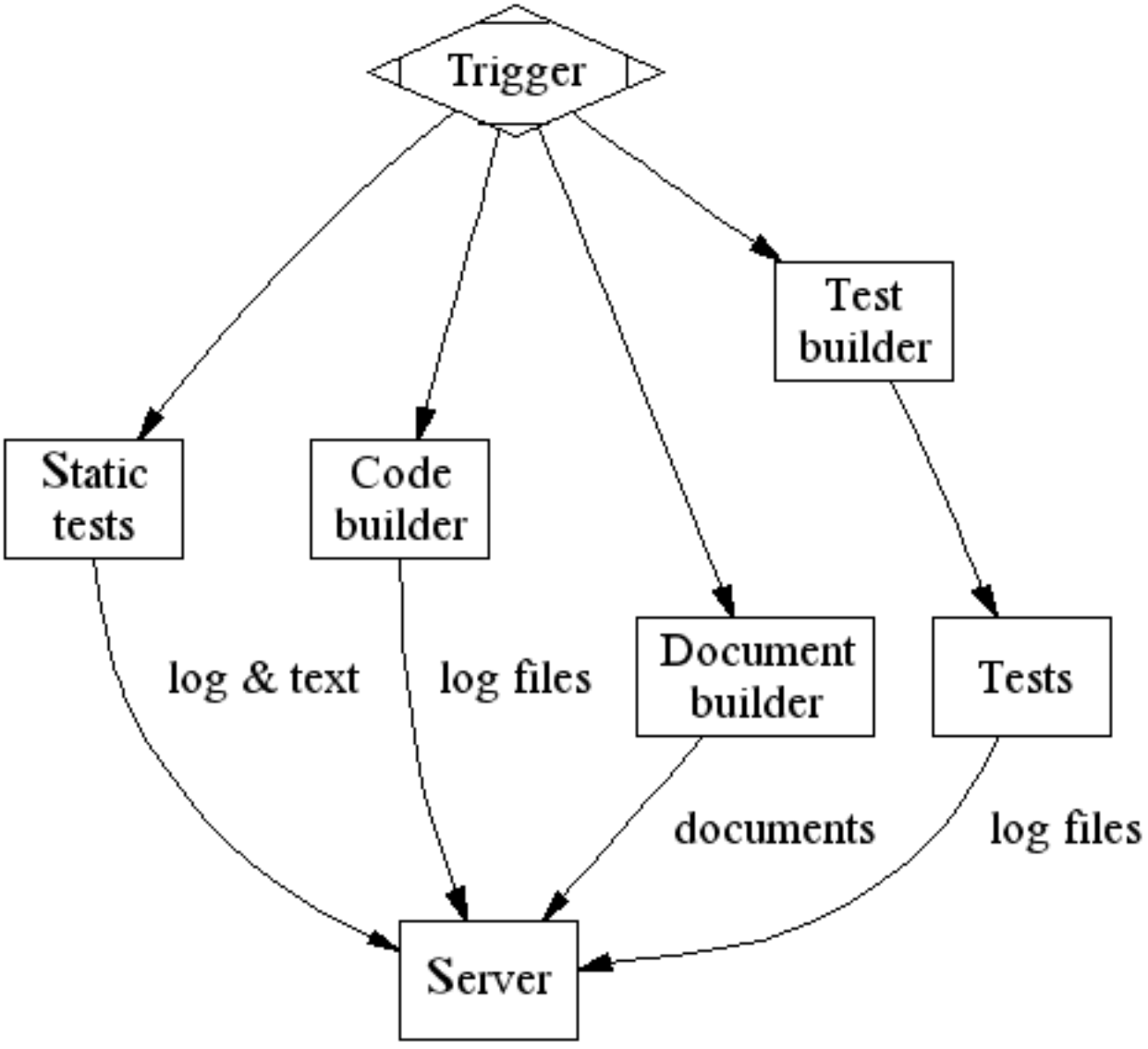

Figure 3: All Building and Testing

Appendices

A Required "After Release" Changes

No "after release" changes are required.