THE UNIVERSITY OF ADELAIDE

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING
ADELAIDE, SOUTH AUSTRALIA 5005.

ELEC ENG 4039 A/B HONOURS PROJECT

A study of the pleiotropy versus redundancy trade-off, using evolutionary computation

Supervisor: **Dr. Derek Abbott**

Moderator: **Mr. Andrew G. Allison**

Students: **Zhiyang Ong (1085011)**

Andy Hao Wei Lo (1077531)

Date Submitted: 19 August 2004

Declaration

The authors, Zhiyang Ong (student number 1085011) and Andy Hao Wei Lo (student number 1077531), declare the following to be of our own work, unless otherwise referenced, as defined by The University of Adelaide's policy on plagiarism.

Acknowledgements

The authors would like to thank Associate Professor Derek Abbott for his supervision and for stimulating the minds of the authors. The authors also wish to express their gratitude to Mr. Matthew Berryman for his guidance.

1. Abstract

This document constitutes the proposal for the project with the aim of determining an optimal balance between pleiotropy and redundancy in a network using evolutionary computation. It outlines the technical background of the project, and establishes the requirements and milestones of the project. Subsequently, a project management plan is described to explain how are we commence and conduct our research and software development. In addition, a discussion about the adopted verification, validation and testing methodologies, and a proposed budget for this project is also included.

Table of Contents

List of Tables

List of Figures

2. Project Aim

The aim of this project is to investigate the optimum ratio of pleiotropy and redundancy for telecommunication networks to perform efficiently under different network conditions. Being a continuation of the work done by former students, it will concentrate on defining additional cost and reliability (fitness) functions. It will also involve modifying and extending Genetic Algorithms (Gas) currently employed in the existing software. Thus, the current software will be modified and enhanced during the course of this project.

2. Technical Background

2.1.Importance of the Project

Genetic algorithm (GA) is a stochastic approach to solve complex optimisation problems, where solutions are simulated and measured by a fitness functions. It has been known to play major roles in fields such as construction, computer hardware design, resource management, distributed computer network topology analysis, and stationary and mobile communications infrastructure optimization [1][2]. By simulating such networks with GA, one is able to explore a large proportion of the solution space in a random and systematic way at the same time. Randomness, due to the nature of the GA's mutation function, is to massively extend the solution space; systematism arises from the crossover operation as it searches the solutions at hand more thoroughly [1][3][4]. Since solutions generated from GA are capable of adapting to changes in requirements or environments, and failures and defects, together with the ever increasing computing power, more accurate network solutions can be discovered with GA in minimal time.

2.2.Applications of the Project

Exercising GA on network optimisation is a significant advance in the modern society, as cost and reliability are always of concerns. In terms of resource management, one may wish to carry

out a set of task as quick as possible at a lowest price. By employing such an algorithm, we can allocate resources or man power to maximise efficiency. When designing any type of network, it is desired to minimize the cost of operating the links, and to maximise the reliability, which is also critical. In this case, lower cost of links may indicate lower quality. Consequently in order to find such an optimum level, we can use GA to solve this problem. Thus the outcome of the project may have a great degree of commercial and social value, especially in the industry of telecommunication networks, where reliability is just as important as the cost [5].

2.3. Research History

The most important component of GA is how you measure the goodness of a solution. Such metric is commonly known as the fitness function. As the name suggests, the better solution will have a better fitness. The previous solution has a fitness function defined as the ratio between reliability and cost (F=R/C). This approach combines two separate metrics, which conceals much information of the solution, because the ratio is just a proportional relationship between two metrics. In addition, a fundamental problem arises where a network with no links will be infinitely fit due to the zero cost. Although the previous research has taken into account many aspects of a telecommunication network (capacity of nodes and links), the results are trivial. Whilst the documentation of the previous software was insufficient, the underlying software architecture, including the graph data structure, will serve as the starting point for our research. Our major focus will be on further developing and refining of the previous algorithms [5].

2.4. Literature Review

Although GA is a radically different and modern approach to problem solving, since design of fitness function is a complex task requiring knowledge of the problem domain, very few research elaborate on networking simulations. However, there has been much effort invested in enhancing the methodologies, such as selection and mutation, in evolutionary computation, which are also fundamental in GA. These research efforts are beneficial for our project development [5].

On the other hand, the Graph data structure is very generic, so sufficient information can be gathered regarding graph traversal, minimum spanning tree construction, shortest path

algorithms, and connectivity test. Each of the above may be used in determining a fitness function for each graph solution, and to verify the correctness of each graph [6].

Furthermore, various academic resources provide detail specification of metrics in telecommunication systems and telecommunications-traffic modelling [7][8][31]. This information is crucial to our task, as we are simulating networks. Since investigation in this field began much earlier than evolutionary computation, a large amount of background and supplementary information are available relating this aspect.

2.5. Critical Issues

Following the literature review, it is realised that there are a number of critical issues encountered by previous researchers, and may be subjects of concern for this project.

Firstly, defining the structure of the chromosome and fitness functions has always been a major issue when measuring intangible entities. Each fitness function is application specific. In particular, a network may have many abstraction levels and variants in media, which introduce certain physical limitations. Therefore the mentioned factors add complexity to define a metric for this purpose.

In addition, one must have sufficient knowledge in GA in terms of selection and population manipulation. Selection is a component which chooses individuals from the population for mating. Providing the fitness function is well-defined, the method of probability allocation and the method of selecting will largely affect the individuals chosen. During selection, it is crucial to observe the chance of each individual being selected, biasness (if exists), and the spread and diversity of solution [9]. Care must be taken when defining methods to manipulate the population because the style of recombination and mutation of chromosome are also application specific.

It is noticed that in the past, computing power may be insufficient to exploit evolutionary computation. Thus we have the advantage of the faster technology on course of our research, although it is advisable to develop software that requires minimal computation power.

3. Project Requirements

3.1.Important Specifications and Requirements

The important milestones for this project are indicated as follows:

1. Setup the software developing environment. This includes setting up the Java 2 Platform, Standard Edition (J2SE) Version 1.4.x or 1.5.0 software, a text editor and appropriate compiler and runtime environment for scripting languages used to manipulate files. A configuration management tool shall also be setup to help manage changes in the software requirements and design.
2. Maintain a logbook to indicate the thought processes of the authors.
3. Prepare the presentation slides, rehearse and present the project proposal seminar.
4. Do a literature review about evolutionary computation, genetic algorithms, graph data structure, GUI development in Java, telecommunications networks, No Free Lunch theorem, and traditional methods of optimization.
5. Comprehend the existing software architectural design and test it to determine if it works as expected. This includes "walking through" the software code, writing test suites for important classes, keeping an audit of the software verification, validation and testing process.
6. Evaluate, modify and add metrics or functions used to measure the reliability and cost of the system.
7. For a particular set of parameters, compare the values of data obtained from the reliability and cost functions of the GA between simulation runs. These values are expected to change between simulation runs.
8. Repeat the previous step for varying inputs, and for a suitable range of inputs. Analyse the effects of changing the number of agents or tasks required to be performed. The number of agents and tasks shall also be increased independently towards large numbers, in the order of tens or hundreds of thousands. Note that this depends of the computing power available to us.

9. Determine the relationships between reliability, cost, constant and varying failure and repair rates, and maximum and varying number of data links, data transfer rates, degree of cluster and separation, and number of operating servers.
10. Analyse the data of the simulated results and draw conclusions from that.
11. Prepare the presentation slides and rehearse for, and present, the final project seminar.
12. Write a final project report individually and a technical paper together.

The software shall be developed using the Java programming language since it is object oriented and is computing platform independent. This is the language that the authors are most competent in; however, they will be enhancing their knowledge about Java during the course of this project. Since the existing software is developed in Java, the authors will be able to extend the work of their predecessors.

The system requirements for the software are the Java Runtime Environment Version 1.4.x or 1.5.x software for execution of the program, an Intel Pentium IV personal computer operating at 2.0 GHz with 256 MB of Random Access Memory or equivalent. A reasonable fast computer is required for the system since simulation speed is critical. Since Java is platform independent, the operating system on which the program is executed does not matter.

3.2.Project Deliverables

The deliverables of the project are indicated in Table 3-1.

Table 3-Number range Table List of project deliverables, corresponding deadlines and version numbers

Deliverable	Date of delivery	Version number
Project proposal	19/08/04	1.0
Project proposal seminar	27/08/04	1.0
Program to find an optimal balance between pleiotropy and redundancy using evolutionary computation	22/10/04	1.0
	25/03/05	2.0
	22/04/05	3.0
User manual	12/05/05	3.0

Soft copies format of all documents (include non deliverables [internal documentations, reading material, reference lists, meeting minutes and emails, Formal Technical Review audits and test audits]) and program must be kept in a CD in each individual's report.	12/05/05	Not applicable
Logbook	03/06/05	Not applicable
Final report	12/05/05	Not applicable
Technical paper	12/05/05	Not applicable
Final project seminar	To Be Announced	Not applicable

3.3. Software Design Interfaces, Operation and Performance

The user can run the program through a graphical user interface. The user shall be able to determine the optimal balance between pleiotropy and redundancy of telecommunications network. The user is able to enter key parameters and constraints in the pop-up window. Users can select the type of strategies of population selection, allow parameters to remain constant of variable during the simulation. The user can pause, resume or halt the simulation when desired.

The program shall display the representation of the network with the optimal balance between pleiotropy and redundancy as a graph in the GUI window. It shall simulate for 5000 generations in approximately 2 hours for 200 servers, 20,000 nodes, and 10,000 data links. The maximum numbers of servers, clients, and generations that can be simulated are 500, 50000, and 100000, respectively.

4. Proposed Approach

4.1.Software Architecture

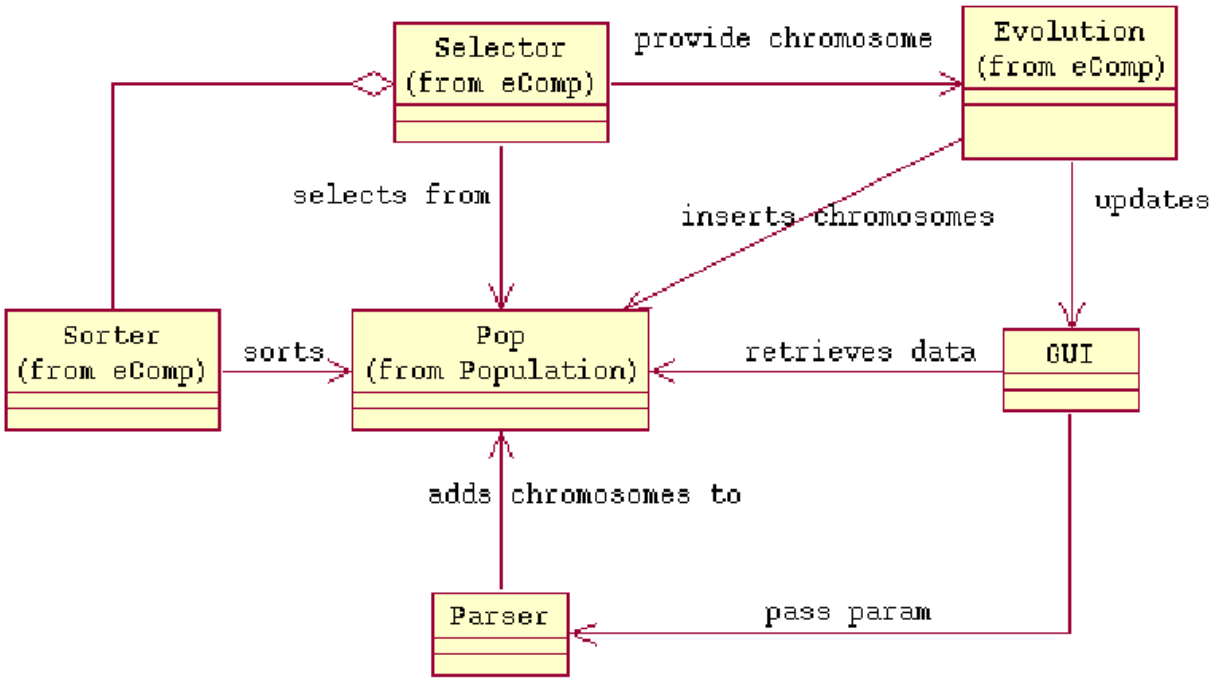

Figure 4-Number range Figure Software architectural design using the Unified Modelling Language (UML) notation

The above diagram displays the relationship amongst each class (box). The software is divided into three packages. The main package consists of a graphical user interface (*GUI*), which the user operates with the program as outlined in Section 3.3, and a *Parser* that reads in a set of data (from the *GUI*) to generate a *Population (indicated as Pop)* of networks. The next important component is the *eComp* package, where the main computation runs. Once a population exists, the selector simulates natural selection of individuals (chromosomes) from the sorted population, which is maintained in order by the Sorter. After each mating or mutation action, the *EvolutionaryComputation (indicated as Evolution)* class shall reinsert the chromosomes back into the population. The third package is called *Population*, which is the portion containing all of the

solutions, is represented as chromosomes. These chromosomes are flattened versions of the adjacent matrix in graph theory. The class *Graph* is made up of *Nodes* and *Edges*.

4.2.Description of the Genetic Algorithms

The pseudo code of the genetic algorithm can be found in Appendix A Pseudo code for Genetic Algorithms. This describes how the proposed genetic algorithm selects genes from a chromosome for mutation and the method of offspring generation via chromosome mating through crossover and mutation.

4.3.Proposed Fitness Functions

The first of the two proposed fitness functions for the genetic algorithms is:

$$F = m*R - n*C;$$

where *R* and *C* are the Reliability and Cost Functions of the telecommunications networks. R is a measure of how reliable is the system whilst C is a measure of the costs of the network. The constants *m* and *n* are weights to indicate the contribution of the reliability and cost of the network to the fitness function.

Note that for these functions, the clients or genetic agents/devices of the telecommunications network are telephones, mobile phones, fax machines, computers, and walker-talkies. The servers of the telecommunications network are satellites, routers, switches for telecommunications exchange, antennae or access points for wireless networks, and computers.

The second proposed fitness function is:

$$F = n*[a*R + b*C - *c*R*C];$$

where *R* and *C* are the Reliability and Cost functions of the telecommunications network as before. The term R*C is used to measure the amount the interaction between the reliability and cost factors. The constants *a*, *b* and *c* are used to indicate the contribution of each factor to the fitness function. The term *n* is used to indicate the type of network. See Appendix B Definition of cost and reliability functions for more information about defining reliability and cost functions.

5. Milestones, Timeline and Division of Work

5.1.Key Milestones and resource allocation

The milestones to be met and the project personnel responsible for the completion of those milestones on time are listed as follows:

1. Zhiyang, with the help of Andy, shall prepare the project proposal. Andy will hold the formal technical review for the proposal.
2. The authors shall present the project proposal at the proposal seminar on August 27, 2004 in EM 316 at 1500 hrs. Andy shall oversee the preparation of the presentation slides and the rehearsal for the seminar.
3. The authors shall begin developing version 1.0 of the "Evo Comp" software on August 30, 2004. Version 1.0 shall include the addition of a cost and reliability function, testing and refactoring of the software architecture. They shall complete the development of version 1.0 by October 22, 2004.
4. Version 2.0 is scheduled to begin on February 28, 2005 and to finish on March 25, 2005. This version shall include an additional cost and reliability function, and allowing the program to deal with varying repair and failure rates. It shall also allow the network to function as a non-repairable or repairable system.
5. Version 3.0 of the software shall commence on the 26th of March, 2005 and be completed by April 22. The network will be designed to handle varying amounts of each server's data links in this version, including networks with unlimited resources. It shall also include limits on the number of servers that can be operating at any given time and the varying number of clients that are connected to the servers. This version shall enable simulation of unlimited clients in a telecommunications network.
6. Each individual shall produce the final project report on the 12th of May, 2005.
7. The authors will collaborate on a joint technical paper that is also due on May 12.

For all three planned versions of the software, Andy shall be the software architect of the software design whilst Zhiyang will play the role of a software quality analyst. He shall ensure that Formal

Technical Reviews are carried out and errors are promptly corrected. [See Appendix C Monitor and Controlling Mechanisms for information about Software Quality Assurance.]

5.2. Software Lifecycle Development Model

Extreme Programming (XP) shall be adopted as the author's agile software development methodology. Whilst XP's values are and will be embraced, some of XP's practices will not be followed as they are not feasible for this project. The good practices of XP result in incremental and iterative development (IID) of the software [10][11][12][13]. This enables the authors to avoid the risks involved in a single-pass linear project development process, whereby an incomplete and over-budget software is available by the project deadline.

Whilst extensive documentation is necessary in the development of large complex systems, the authors chose XP instead of other IID software development lifecycle models so that documentation, which is not required for academic assessment, is kept to a minimum. This allows the authors to make modifications to the software without worrying about updating all the relevant documentation; thus, production and maintenance labour costs will be reduced [14][15].

By enabling the authors to respond to changes quickly and easily using XP, the authors can aim for and achieve three iterations of the software development lifecycle. In each version of the software, software verification, validation and testing will ensure that the version has a simple and easily comprehensible design to enable further modifications in design and requirements by the authors or their successors [See Appendix G Extreme Programming Practices for more information] [16].

5.3. Project Scheduling and Organisation

A Gantt chart, shown in Figure C-1 in Appendix C, has been drawn to indicate the expected schedule of the tasks and milestones that are expected to be completed during the duration of the project. Whilst planning the timeline, the authors have endeavoured to allow tasks to run

concurrently where possible. They have also taken into account the importance and urgency of the tasks whilst prioritising them.

The tasks literature review and logbook maintenance are continuous tasks that will be carried out concurrently with other tasks throughout the project. The former task allows the authors to familiarise themselves with the specialised and extensive fields covered in the scope of this project. The fields are optimization, evolutionary computation using genetic algorithms, telecommunication networks, concurrent programming, design patterns, graph theory in Computer Science and GUI development. The latter task reflects the ongoing documentation of our ideas and thought process.

The initial two milestones of the project proposal and the proposal seminar presentation shall be concurrently developed and prepared for, respectively. This sets out aims and requirements of this project so that the authors will be able to structure their solutions incrementally to meet these goals. The desired outcomes for each iteration are distributed amongst the versions, milestones 3, 4 and 5, subject to their difficulty. The easier requirements are handled in the first iteration so that the authors can develop confidence to handle them by the third iteration.

The last two milestones shall be written towards the end of milestone 5. The sections of the final report on literature review, simulation procedure, test results and data analyses shall be updated at the end of each version. That is, the final report shall be kept as an electronic workbook throughout the duration of this project. However, the authors will officially commence on writing the final reports at the aforementioned period. The task of refining the technical paper is optional. This task shall be carried out if the technical paper needs to be refined for submission to a journal or conference

[See Appendix C Monitor and Controlling Mechanisms for details on how the authors will manage changes and risk, and develop good quality software.]

6. Proposed Budget

A budget ofAUD$320 has been given by the School of Electrical and Electronic Engineering, Adelaide University, to cover project expenses . Table 6-1 shows the proposed budget using this allocated resource. The expenses that could be possibly claimed from the budget are for the printing and binding of the project documents that will be produced. The authors have endeavoured to use free software to minimise costs. The software are available from the internet or are currently provided by the faculty. Purchase of journal articles that the authors deemed to be informative and relevant to the development of the project are tentatively limited to ten. This amounts to AUD$132.00. The provisional budget for the purchase of books is AUD$100.00.

Table 6-Number range Table Proposed budget using the allocated resources for this project

Task	Cost in Australian dollars (AUD$)
Printing of project proposal	3
Printing of final report	5
Printing of technical paper	2
Binding of project proposal and reports, and technical paper.	4
Printing of Seminar handouts	6
Printing of overheads slides	25
Purchase of journal articles	132
Purchase of relevant books	100
Total	277

7. References

[1] A Marczyk, *Genetic Algorithms and Evolutionary Computation.* Viewed 28 July, 2004; available from the Internet at <http://www.talkorigins.org/fgaqs/genalg/genalg.html>, 2004

[2] *Wikipedia* [Online] 2004. Available: Genetic Algorithm. Viewed 12 Aug, 2004, available from the Internet at <http://en.wikipedia.org/wiki/Genetic_algorithm>

[3] *NIBS Inc* [Online] 1996, Available: Genetic Algorithms and Genetically Evolved Neural Networks. Viewed 12 Aug, 2004, available from the internet at <http://www.singaporegateway.com/products/nfga/>

[4] PH Winston, *Artificial Intelligence*, 3rd Edition, Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1992

[5] MJ Berryman, A Allison & D Abbott, *Optimizing genetic algorithms strategies for evolving networks*. Viewed 8 August, 2004; available from the Internet at <http://arxiv.org/PS_cache/cs/pdf/0404/0404019.pdf>, 2004

[6] MT Goodrich & R Tamassia, *Data Structures and Algorithms in Java*, 3rd Edition, John Wiley & Sons, Inc, New Jersey, 2003

[7] *Caslon Analytics* [Online] 2004. Available: Net metrics and statistics guide. Viewed 16 August, 2004; available from the Internet at <http://www.caslon.com.au/metricsguide.htm>

[8] JJ Li, E Wong, D Zage, & W Zage, *Validation of Design Metrics on a Telecommunication Application.* Viewed 15 August, 2004; available from the Internet at <http://www.serc.net/report/tr171p.pdf>, n.d.

[9] *GEATbx* [Online] 2000. Available: Selection - Evolutionary Algorithms. Viewed 12 Aug, 2004, available from the Internet at <http://www.geatbx.com/docu/algindex-02.html>

[10] B Boehm & R Turner, "Using Risk to Balance Agile and Plan-Driven Methods", *IEEE Computer*, vol. 36, no. 6, pp 57-66, 2003

[11] M Cohn & D Ford, "Introducing an Agile Process to an Organization", *IEEE Computer*, vol. 36, no. 6, pp 74-78, 2003

[12] C Larman & VR Basili, "Iterative and Incremental Development: A Brief History", *IEEE Computer*, vol. 36, no. 6, pp 47-56, 2003

[13] M Lycett, RD Macredie, C Patel & RJ Paul, "Migrating Agile Methods to Standardized Development Practice", *IEEE Computer*, vol. 36, no. 6, pp 79-85, 2003

[14] L Williams, "The XP programmer: The few-minutes programmer", *IEEE Software*, vol. 20, no. 3, pp 16-20, 2003

[15] O Murru, R Deias & G Mugheddue, "Assessing XP at a European Internet Company", *IEEE Software*, vol. 20, no. 3, pp 37-43, 2003

[16] B Pyritz, “Extreme programming in the Telecommunications Domain”, *Bell Labs Technical Journal – Software Technologies for Telecommunications Competitiveness*, vol. 8, no. 3, 2003

[17] A van den Hengel, *Artificial Intelligence Lecture notes: Genetic Algorithms*, The University of Adelaide, Adelaide, 2004

[18] I Sommerville, *Software Engineering*, 5th Edition, Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1996

[19] RS Pressman, *Software Engineering: A Practitioner's Approach*, 5th Edition, McGraw-Hill, Singapore, 2001

[20] WS Humphrey, *A Discipline for Software Engineering*, Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1995

[21] *iSixSigma* [Online] 2004. Available: New to Six Sigma. Viewed 18 August, 2004; available from the Internet at <http://www.isixsigma.com/library/content/six-sigma-newbie.asp>

[22] RS Kenett & S Zacks, *Modern Industrial Statistics: Design and Control of Quality and Reliability*, Duxbury Press, Pacific Groove, California 1998

[23] R Jeffries, *Are we doing XP?*, XProgramming.com, viewed 11 August, 2004; available from the Internet at <http://www.xprogramming.com/xpmag/are_we_doing_xp.htm>, 2000

[24] J Rasmussen, "Introducing XP into Greenfield Projects: lessons learned", IEEE Software, vol. 20, no. 3, pp 21-28, 2003

[25] R Jeffries, *What is Extreme Programming?*, XProgramming.com, viewed 11 August, 2004; available from the Internet at <http://www.xprogramming.com/xpmag/whatisxp.htm>, 2001a

[26] D Wells, *Refactor Mercilessly*, Extreme Programming.com, viewed 11 August, 2004; available from the Internet at <http://www.extremeprogramming.org/rules/refactor.html>, 1999

[27] R Jeffries, *Essential XP: Emergent Design*, XProgramming.com, viewed 11 August, 2004; available from the Internet at <http://www.xprogramming.com/xpmag/expEmergentDesign.htm>, 2001b

[28] WA Wood & WL Kleb, "Exploring XP for scientific research", IEEE Software, vol. 20, no. 3, pp 30-36, 2003

[29] JK Ousterhout, "Scripting: Higher-Level Programming for the 21st Century", IEEE Computer, vol. 31, no. 1, pp 23-30, 1998

[30] *Sun Microsystems, Inc* [Online] 2004. Available: Developers Home: Product Technologies: Java Technology: Java 2 Standard Edition: Core Java: Javadoc Tool: Reference, "How to Write Doc Comments for the Javadoc Tool". Viewed 12 August, 2004; available from the Internet at <http://java.sun.com/j2se/javadoc/writingdoccomments/>

[31] M Kamat, *In Quest of tele-quality*. Viewed 16 August, 2004; available on the Internet <http://www.networkmagazineindia.com/200103/comm1.htm>, 2001

9. Appendix A Pseudo code for Genetic Algorithms

The pseudo code for the genetic algorithm is described as follows [17]:

```
To evolve M generations of a population of N chromosomes…
Generate a random initial population of N chromosomes.
while ( (M – 1) > 0 )
{
    while ( (N/2) > 0 )
    {
        Select a pair of chromosomes for breeding;
        Generate two offspring via chromosome mating using crossover or
        mutation;
        Add offspring to new generation;
        (N/2) – 1;
    }
    replace current population with new population;
    (M – 1) – 1;
}
```

10. Appendix B Definition of cost and reliability functions

The proposed reliability function of a telecommunications network is:

$$R = W / T + numf;$$

where *W* is the total number of data links that are working, *T* is the total number of available links and *numf* is the number of failures per month.

The proposed cost function of a telecommunications network is:

$$C = \alpha*numf + \varepsilon*(\beta*dist + \gamma*amp + \delta*sat);$$

where *numf* is used to indicate the number of failed links, *dist* measures the cost of transmitting a signal over each failed link, *amp* indicates the amount of amplification required in the transmitted signal, and *sat* is the cost of a satellite orbiting around earth. The constants α, β, γ, δ, and ε are constants used to indicate the weight of the factors.

11. Appendix C Monitor and Controlling Mechanisms

The project shall be monitored informally, through regular discussions about problems encountered, and formally, through document and code reviews. This activity shall be a continuous activity for the project's duration. To enhance this process, documents such as test reports for the program, document and code review audits, and data files for software metrics will be kept. The software metrics shall be used to indicate the project's progress; for more information on the software metrics that are used, see Appendix D Software Metrics used in Software Quality Assurance [18].

The authors shall perform unit, module, system and acceptance tests to verify and validate the software. Different types of software strategies such as top-down, bottom-up, thread and stress testing, and defect testing approaches, like interface, black- and glass- box testing, will be employed in those tests. A guideline on the coding style to be used is found in Appendix H Coding Standard. This ensures that the code written by either author is similar in style and easily understandable to the other author [18][19].

The authors shall also review documents and code regularly to verify and validate the product; the guidelines for such reviews are found in Appendix E Checklist for Formal Technical Review of Documents and Appendix F Checklist for Code Review. Regarding these reviews, the authors will prepare for them by going through the documents or code. They will endeavour to find errors, omissions and inconsistencies in the program and documents. The authors shall fix defects discovered in this inspection process. These activities shall ensure that the developed software shall be reliable, efficient, and easily maintainable and usable [18][19].

Written by Zhiyang Ong (1085011)
And Andy Hao Wei Lo (1077531)

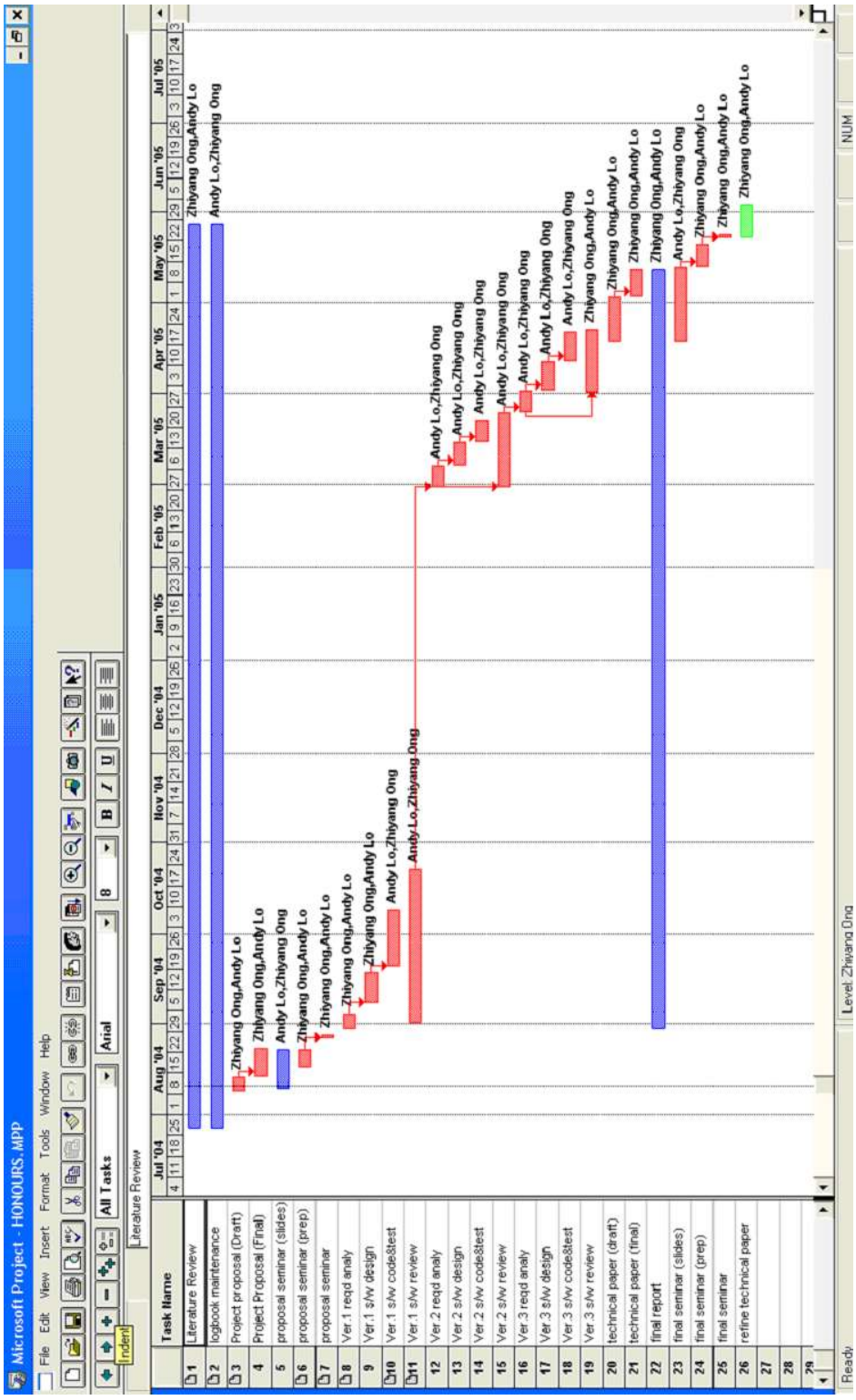

Figure C-Number range Figure Gantt chart indicating the timeline for the completion of the milestones

12. Appendix D Software Metrics used in Software Quality Assurance

Whilst the authors do not wish to manage the project that will allow them to obtain International Organisation for Standardisation (ISO) 9002 and Capability Maturity Model Integration (CMMI) Level 5 certifications, they will not compromise the quality of the software in development. Practices from the Six Sigma methodology of improving operational performance will adopted to help the authors improve the quality of the software [20][21]. The metrics that shall be employed to improve the quality of the software development process using simple statistical process control are [19]:

- Number of errors per 100 lines of code (errors/100LOC). This will help the authors to determine if their process mean has shifted [22]. An example of how a control chart can be plotted is indicated in Figure D-1.
- Number of lines of code written by the authors per month (LOC/mth). This is used as a rough indicator of the authors' progress.
- Defect Efficiency Removal (DRE).
  DRE = E/(E+D), where E is the number of errors found before the code walkthroughs and D is the number of defects found after the code walkthroughs.

Note that scripting languages can be used to plot the control charts automatically.

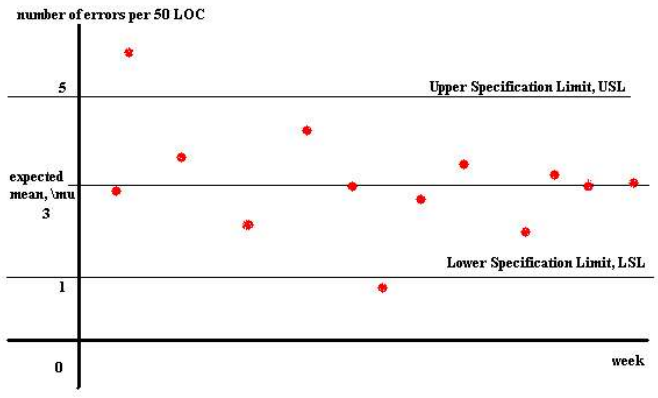

Figure D-Number range Figure A control chart indicating the number of errors per 100 lines of code each week

13. Appendix E Checklist for Formal Technical Review of Documents

The checklist for the review of documents is listed in Table E-1.

Table E-Number range Table Checklist for Formal Technical Review of Documents

Purpose	Questions to guide you in conducting a review of documents
Layout	Check the presentation of the document for: • an appealing layout • proper structure of the documents • consistency of sections with the contents page and other sections
Grammar and Semantics	Check for the grammar of the language used and for the meaning of the sentences
References	Check that references are properly included in the text and are fully listed in the reference list Check for the proper use of quotations and paraphrasing of statements/adaptation of figures and tables
Content	Check that the content of the documents meet the requirements of the project and are consistent with other documents

14. Appendix F Checklist for Code Review

The checklist for code inspection is listed in Table F-1 [18][19][20]. Each step in the checklist shall be completed before proceeding to the next step.

Table F-Number range Table Code Review Guideline and Checklist

Purpose	Questions to guide you in conducting a code inspection
Complete	Verify that the code implements the software design
Initialization and declaration	Check that all variables and parameters are properly initialised and declared • At the initialisation of the program • At the start of every loop • At the entry to functions/procedures/methods Check that there are no duplicate variables, constants and methods
Method/function calls	Check that method/function call formats have appropriate parameters and use of pointers Check that the types of parameters in method calls and declarations match, and they are in the correct order
Names	Check that variable, parameter and method names are spelt correctly, and have a consistent naming style
Output format	Check that the line stepping and spacing of the output format is proper
Parentheses and brackets	Check that parentheses and brackets are properly matched.
Logic operators	Verify the proper use of Boolean operators and each logic function has proper brackets surrounding it
Line-by-line check	Check every line of code for proper syntax, semantics and punctuation
Standards	Ensure that the code conforms to the coding standards
Entry and exit conditions	Verify that all program files are properly declared, entered and exited Check that all input variables are used Check that the output of method/function calls are correct
Tests for pre- and post-conditions, and assertions	Ensure that tests for pre- and post- conditions and assertions are included in the program to determine the correctness of the program during execution.
Exceptions	Check that exceptions are properly declared, thrown and dealt with Check that all possible error conditions have been taken into account
Loops and recursion	Check that all loops and recursive method calls are properly terminated

Control faults	Check that all conditional statements are correct Check that switch/case statements have all possible cases accounted for
Range	Check that the variables and methods handle values within an acceptable range Check if the correct limits are used
Error Messages	Check that appropriate messages are produced to help the user debug the program by indicating the type of error and the cause of the error
Pointers	Check that pointers are used appropriately
Application	Check the correctness of all computation and algorithms
System	Ensure that the system is operating efficiently as specified by the supervisor
Environment	Check that the program is indeed platform independent. Check of it complies and executes on major platforms

15. Appendix G Extreme Programming Practices

The values of Extreme Programming (XP) are communication, feedback, simplicity and courage. Their significance in software development is indicated as follows [16]:

- Personal communication is preferred as it allows a greater and faster flow of information exchange and it provides opportunities for clarification of doubts.
- Feedback is valued as software developers can correct misinterpretation of, or adapt to, changing requirements and software design more easily than they would be by making optimistic estimates and hypothesising. It allows developers to correct design and/or implementation errors before investing a lot of resources into implementing the design, or further developing, the system. Feedback also allows the authors to meet their supervisor for guidance and insight about, and to shed new light on, the project.
- Keep the software simple allows it to be easily modified to meet changing needs. The simply designed software does not require unnecessary development of safeguards to prevent possible errors in the future. Investments in safeguards are a waste of resources, especially if they are redundant and do not prevent the expected errors from occurring due to changing requirements.
- Simplicity in the software design and the availability of automated test suites for verification and validation allow software developers to make modifications to the software confidently and quickly. By using the simplest solution, it also allows easier management of the software development process.

Some practises of XP that shall be implemented are described as follows. Software planning makes use of regular feedbacks and communication to enhance the rapid learning process and increases the confidence of the authors. They have planned to develop the software for small releases to acquire adequate feedback and to allow the assessors to gauge the project's progress. This builds confidence into the authors as they do not have to worry about whether the program will function as expected at the end of a large release [23][24].

Testing of the program by building automated test suites that can be executed before and after modifications ensures rapid development of the software. Hence, the developers do not have to waste unnecessary time manually testing the program to determine the integrity of the software after modifications [16][25]. Whilst XP encourages test-first development by using test suites to reflect adherence to software specifications, it is up to each individual to practice it [10][24]. Test will be executed early and regularly in the development process to prevent the accumulation of errors that makes the program hard to maintain. This will decrease the risk of the late delivery of the software.

Simple design allows the authors to understand the code better and more easily so that the program can be modified more easily. Software developers will also not have to spend unproductive time understanding an unduly complex software design. Thus, simple design helps users to reduce the number of errors introduced due to miscomprehension of the requirements [23].

Refactoring allows the internal structure of the software to be modified whilst maintaining the same external functions and Application Programming Interface so that the software is simpler, and is easier to understand and modify. It includes the removal repetitive code and rewording the names of programs and operations, and program variables so that they are more descriptive [16] [24][26]. This shall result in a highly modular system, where each module is made up of separate components that are highly cohesive and loosely coupled [27].

Collective code ownership is practised by allowing the authors and their supervisor full access to the source code through the configuration management tool. That is, nobody has sole ownership of the source code so that each author or the supervisor will be able to access and improve necessary parts of the programs that the individual has not written. Defects found to be in existence in the program shall be treated as the concern of everybody, rather than the person assigned to write that part of the program, where the errors exist [23][25][28].

Continuous integration of the program shall be facilitated through the use of automated testing and development through the use of test suites and scripting languages. That is, automated software tools shall be employed to maintain a fully integrated system regularly [25][29]. The software shall be developed at a sustainable pace so that the authors will not be under undue pressure during the examination preparation period. This can happen if they had spent too much time on the final year project and had neglected their physical, mental and emotional well-being, and other subjects [28].

A coding standard, found in Appendix H Coding Standard, has been agreed upon by the authors for the development of the software. This will facilitate the development of code that is mutually understandable by the authors and emphasizes clarity and consistency. Lastly, metaphors and analogies shall be used, when necessary, so that the authors and supervisors can communicate using the same jargon [24][28].

The following two XP practices shall or may not be adopted. Paired programming takes place when a junior programmer works in front of the computer whilst the senior programmer "watches over" the junior programmer. The experienced observer strategically thinks about whether the code in development fulfils the requirements of the project. Hence, it is also known as real time code review. This practice is unlikely to be adopted as the observer may be easily distracted and would do not determine if the programmer has written code efficiently, effectively and correctly. This would result in a waste of effort and time, and an insidious belief that the error ridden program has been verified and validated [16][24].

Consequently, code inspection shall be adopted even though it may take more time and effort to understand the program being reviewed. When the authors are prepared for the code reviews they conduct, they can avoid the pitfalls that may occur in paired programming that may exacerbate the error ridden programs

The XP practice of having an on-site customer is not feasible since the authors and their supervisor have a myriad of academic, research, family and personal commitments. It is hard to

find adequate timeslots each week for them to meet and develop the software. However, this allows the authors to be independent and hone their problem solving skills [24].

16. Appendix H Coding Standard

The software in the project shall be written by both of the authors in several languages; Java is used for the main program whilst scripts shall be written in a language of the author's choice. The lifetime of the software shall extend beyond the date of delivery so that further research may be carried out in this area. Successors shall be able to modify and extend the delivered software. Hence, to enable the ease of software development and maintenance, all code written must conform to the coding standards specified in this section [19].

H.1 Java Coding Standards

The coding standard for Java is defined as [30]:

- The structure of Java code shall be organised in the following order: imported packages, constants, instance variables, constructors, and methods.
- Indents included in the code shall be four spaces.
- Names of constants shall be capitalised, with underscores indicating the start of a new word. For example, MAX_SIZE is the name of a constant.
- Variable, method, Class, and package names shall be labelled using the Camel/Hump notation. Class names shall begin with a capital letter. For example, GuiManager, Gui, numOfGenerations, evolve are names of a Class, package, instance variable and a method.
- All Java code must be written with the following style of bracing:

```
methodName(arguments) {
        method body…
}
or
while(condition) {
        loop body…
```

}

That is, parenthesis start one character space after the name of the method or programming construct, such as if-else statements, for or while loops and switch statements.

- Comments shall use the Javadoc conventions as a clear and convenient standard. This allows html Application Programming Interface specifications for the software to be generated.

For Java classes, the Javadoc convention is implemented as follows:

- Each class must begin with a brief description of the Class. The name of the Class's author is preceded by the tag @ author.
- If the Class is revised at a later date, the editor's name, date of revision and brief description of the revision must be included. These are preceded by the tag @revision. This list shall be truncated to its baseline entry once this Class has been entered into a new baseline.
- The Javadoc style tag @version indicates the version of the Class.
- The Javadoc style tag @approved indicates the date that this Class has successfully passed a code review. It also indicates the name of the individual whom did the code review with the author.
- The Javadoc style tag @errorsHandled shall indicate the types of Exceptions that are thrown by this Class and the circumstances that they are thrown in.
- The Javadoc style tag @bugsList indicates the list of bugs that have been found in this Class as well as the actions taken, or yet to be taken, to debug them.
- There shall be no Javadoc style tags for packages that the Class belongs to since the name of the package shall be explicitly stated in the package's import statement.
- Two rows of white space shall be used to separate methods and constructors whilst one row of white space shall be used to separate constants and instance variables.
- A line of dashes in between two rows of white space is used to separate constants and variables, constructors and methods from each other.

For Java classes, the Javadoc convention is implemented as follows:

- Each method in a Java Class must be preceded by a comment describing the function of the method, preconditions and post conditions.
- The comment block must begin with '/**' on a line by itself and each subsequent comment line shall begin with '* '. The final comment line shall contain '*/' by itself.
- The comments shall begin with a succinct description of the method.
- The description shall be followed by the pre- and post- conditions of the method.
- Each precondition shall be preceded by Javadoc style tag @pre.
- Each postcondition shall be preceded by Javadoc style tag @post.
- The Javadoc style tag @param indicates the list of formal parameters required by this method and what are the formal parameters used for.
- The Javadoc style tag @throws indicates the Exceptions that can be thrown by this method and the circumstances that they were thrown in.
- The Javadoc style tag @bugs includes the list of bugs known about this method and the actions taken to remedy this problem.
- If glass-box testing is used with the method, comments preceded with the tag @tested must include a description of the name of the tester, date of the test and how it was tested.

To prevent the occurrence of wrapped lines, limit any Javadoc-commented lines to 80 characters.

H.2 Coding Standards for Scripting Languages

All scripts shall begin with a comment describing the program, the date that the script was written, the name of the author, version number. Each tested script shall indicate the test date and name of the tester. If pre- and post-conditions are required, they must be clearly commented.