FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

UNIVERSITY OF MALAYA



# SIMULATION OF QUANTUM GROWING NETWORK

**ONG BOON TEONG**
**WEK010223**

**Department of Artificial Intelligence**

**WXES3182**

Session 2003/2004

*Under the supervision of*
Prof. Ir. Dr. Selvanathan Narainasamy

*Moderator*
Mr. Mohd. Nor Ridzuan bin Daud

---

A system development in partial fulfillment of the requirements of the degree of
Bachelor of Computer Science
University of Malaya

# Abstract

Over the past 40 years there has been a dramatic miniaturization in computer technology. If current trends continue, by the year 2020 the basic memory components of a computer will be the size of individual atoms. At such scales, the current model of computation, based on a mathematical idealization known as the Universal Turing Machine, is simply invalid. A new field, called "quantum computing" is emerging that is re-inventing the foundations of computer science and information theory in a way that is consistent with quantum physics – the most accurate model of reality that is currently known.

Remarkably this new theory predicts that quantum computers can perform tasks exponentially faster than any conventional computer. Moreover, quantum effects allow unprecedented tasks to be performed such as teleporting information, breaking supposedly "unbreakable" codes, generating true random numbers, and communicating with messages that betray the presence of eavesdropping. These capabilities are of significant practical importance to banks and government agencies. Indeed, a quantum scheme for sending and receiving ultra-secure messages has already been implemented over a distance of 30km – far enough to wire the financial district of any major city.

Modern microprocessors have to be designed with quantum mechanics in mind in order to ensure that they function correctly. In essence they use quantum effects to ensure that their classical computations are upheld. What no current processor does is to fully exploit quantum effects. Recently, it took 1,600 computers communicating over the Internet 8 months to solve the factorisation of a 129-digit number. Should it be possible to build a microprocessor which fully exploits quantum mechanics then it may be

possible to carry out such factorisations in remarkably less time. The RSA algorithm, a widely used encryption system, is safe only if such factorisations cannot be performed quickly.

Although modern computers already exploit some quantum phenomena they do not make use of the full repertoire of quantum phenomena that Nature provides. Harnessing these phenomena will take computing technology to the very brink of what is possible in this Universe.

# Acknowledgement

I would like to thank Prof. Ir. Dr. Selvanathan for introducing me to the field of quantum computation in the first place, suggesting an excellent project idea and then explaining the basic and fundamental of the field in such clear terms. I would also like to thanks him for his help and guidance throughout the course of the project.

The works of Paul Benioff, Richard Feynman, P W Shor, Lov K Grover, David Deutsch, P A Zizzi, S Hameroff, R Penrose, Jacob West of Caltech, Michael A Nielsen, Isaac L Chuang, and many others have provided the central research material for this project. Without their constant research into this exciting field, projects such as this would not be possible.

# Table of Contents

# Table of Figures

# Chapter 1: Introduction

## 1.1 Overview

Quantum computing is a field which falls under one the studies of natural computing. Natural Computing is a general term referring to computing going on in nature and computing inspired by nature. When complex phenomena are going on in nature are viewed as computational processes, our understanding of these phenomena and of the essence of computation is enhanced. In this way one gains valuable insights into both natural sciences and computer science. Characteristic for man-designed computing inspired by nature is the metaphorical use of concepts, principles and mechanisms underlying natural systems.

Quantum computing is the area of study focused on developing computer technology based on the principles of quantum theory, which explains the nature and behavior of energy and matter on the quantum (atomic and subatomic) level. Development of a quantum computer, if practical, would mark a leap forward in computing capability far greater than that from the abacus to a modern supercomputer, which performance gains in the billion-fold realm and beyond. The quantum computer, following the laws of quantum physics, would gain enormous processing power through the ability to be in multiple states, and to perform tasks using all permutations simultaneously.

Current centers of research, which are doing extensive researches in quantum computing, include International Business Machine (IBM), California Institute of Technology (Caltech), Oxford University – Qubit, Stanford University, Massachusetts Institute of Technology (MIT), Los Alamos National Library, and many more.

## 1.2 Project Motivation

In the twentieth century, as civilization advanced, information was added to the list of resources such as materials, forces, and energies. Invention of computers meant that information processing was possible outside the realm of the human brain. Computer transformed from simple mechanical devices using gears and relays to complex electronic circuits involving transistors and ICs (integrated circuits). With the creation and advancement of Lithographic Techniques, fraction of a micron wide logic gates and wires on silicon chips are invented and manufactured. Soon, logic gates and parts will be shrunk so that they are only made of a handful of atoms. At this level, quantum mechanics and theory have to be used and applied to supplement and replace what we have now.

Basically, there are two main factors that motivate the study and research of quantum computing. Firstly, some important computational problems may seem permanently intractable. Their complexity grows exponentially with problem size. For an example, factoring a large number (large number factorization is the main concept of internet unbreakable codes). Secondly, to do all these heavy and complex computational problems, supercomputers are needed but as described by Moore's Law, performance improvements in classical computer circuits may be a approaching a limit. This is when transistors will reach fundamental physical limits when they begin to approach the size of atom, where quantum mechanics comes into picture.

The essential elements of quantum computing originated with Paul Benioff, working at Argonne National Labs, in 1981. He theorized a classical computer operating with some quantum mechanical principles. But it is generally accepted that David

Deutsch of Oxford University provided the critical impetus for quantum computing research. In 1984, he was at a computation theory conference and began to wonder about the possibility of designing a computer that was based exclusively on quantum rules, then published his breakthrough paper a few months later. With this, the race began to exploit his ideas.

This final year project is fundamentally motivated by the field of quantum computing, and is specifically motivated by the interesting findings of quantum coherent superposition which can lead to a massive parallel processing as described as a quantum growing network. This study can be regarded as the first attempt toward a future model for the quantum World Wide Web or rather, the first real Intelligent Web, which mimics the process of thoughts in human brain.

Apart from that, this project is also motivated by the study and research in the area of quantum error correction, particularly the effect of decoherence on the quantum states of an energy or matter.

## 1.3 Project Objective

The surface objectives of this project are as below:

- To study the emergence and impact of Quantum Computation and Quantum Information Processing.
- To understand the concepts of Quantum Theory and Quantum Mechanics.
- To study the differences between classical computing with quantum computing.
- To study the functions and operations of important Quantum Gates and Circuits.

- To understand some of the major Quantum Algorithms.

Besides the surface objectives as mentioned above, this project is to meet a main and specific objective. The objective is to conduct a deep study on quantum coherent superposition which lead to the findings of Penrose-Hameroff Orch Model of Consciousness and its relation with the quantum growing network. The implementation of this project will meet the objective of simulating and visualizing the effect of virtual quantum states on the speed of growth (related to massive parallel processing) of a quantum network, how they are operated by quantum logic gates (using Hadamard gates) and transformed into qubits (information).

## 1.4 Project Scope and Expected Outcome

The scope of the project is basically revolving around understanding the quantum growing network algorithm and code the algorithm using a chosen programming language, which is the Java programming language with an object oriented approach.

The outcome of the project should be a standalone Java application which can also be executed as a Java applet that graphically visualizes the quantum growing network with nodes and links.

# Chapter 2: Literature Review

## 2.1 Background to Quantum Computation

### 2.1.1 Quantum Theory

Quantum theory's development began in 1900 with a presentation by Max Planck to the German Physical Society, in which he introduced the idea that energy exists in individual units (which he called "quanta"), as does matter. Further developments by a number of scientists over the following 30 years lead to the modern understanding of quantum theory.

The essential elements of Quantum Theory are:

- Energy, like matter, consists of discrete units, rather than solely as a continuous wave.
- Elementary particles of both energy and matter, depending on the conditions, may behave like either particles or waves.
- The movement of elementary particles is inherently random, and, thus, unpredictable.
- The simultaneous measurement of two complementary values, such as the position and momentum of an elementary particle, is inescapably flawed; the more precisely one value is measured, the more flawed will be the measurement of the other value.

The development of Quantum Theory does not stop here. In further developments, Neil Bohr proposed the Copenhagen interpretation of quantum theory, which asserts that a particle is whatever it is measured to be (for example, a wave or a

particle) but that it cannot be assumed to have specific properties, or even to exist, until it is measured. In short, Bohr was saying that objective reality does not exist. This translates to a principle called superposition that claims that while we do not know what the state of any object is, it is actually in all possible states simultaneously, as long as we do not look to check.

To illustrate this theory, the famous and somewhat cruel analogy of Schrodinger's Cat. First, have a living cat and place it in a thick lead box. At this stage, there is no question that the cat is alive. We then throw in a vial of cyanide and seal the box. We do not know if the cat is alive or if it has broken the cyanide capsule and died. Since we do not know, the cat is both dead and alive, according to quantum law – in a superposition of states. It is only when we break open the box and see what condition the cat is in that the superposition is lost, and the cat must be either alive or dead.

The second interpretation of quantum theory is the mutiverse or many-worlds theory. It holds that as soon as a potential exists for any object to be in any state, the universe of that object transmutes into a series of parallel universes equal to the number of possible states in which that the object can exist, with each universe containing a unique single possible state of that object. Furthermore, there is a mechanism for interaction between these universes that somehow permits all states to be accessible in some way and for all possible states to be affected in some manner. Stephen Hawking and late Richard Feynman are among the scientists who have expressed a preference for the many-worlds theory.

Whichever argument one chooses, the principle that, in some way, one particle can exist in numerous states opens up profound implications for computing.

### 2.1.1 Introduction to Quantum Effects

In Young's double slit experiment below (Figure 2.1) light coming from the light source produces interference patterns on the screen S. If we were to consider that light is a wave then the patterns can be explained by the interference of light travelling through slits X and Y.

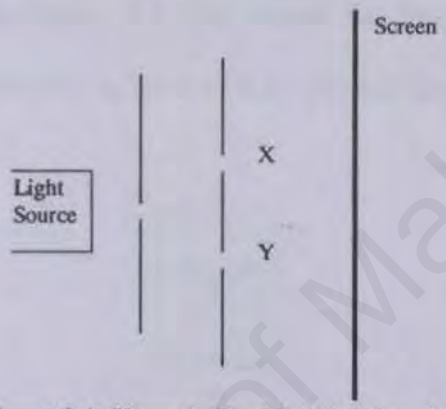


**Figure 2.1: Young's Double Slits Experiment.**

Since Young first performed his experiment in 1801 we have gained a far better control over our light sources. We also now know that light is made of particles called photons. We can arrange for a single photon to leave the emitter, pass through the slits and hit the screen. What we see is that the photons strike certain areas of the screen more often than other areas, and that that this is the reason for the light and dark fringes.

This raises some interesting questions. For instance, just why does a single photon hit the screen more in some areas than in others? If light is made up of particles then how can a single particle travelling through one of the slits X or Y produce an interference pattern? As only one particle has been emitted from the light source surely there is nothing for the particle to interfere with.

One theory is that the photon is more likely to strike certain areas of the screen because it has effectively travelled to the screen via every possible path, with some paths

interfering with others. This interference may be constructive or destructive and accounts for why the photons never hit certain areas of the screen. The effects of quantum interference are even more apparent in the results of a second experiment, shown below.

Consider a beam of light being emitted from a laser and hitting a partially reflective mirror, as shown in Figure 2.2. The mirror has been designed to reflect or transmit light with equal probability, so 50% of the light will hit detector 1 and 50% will hit detector 2.

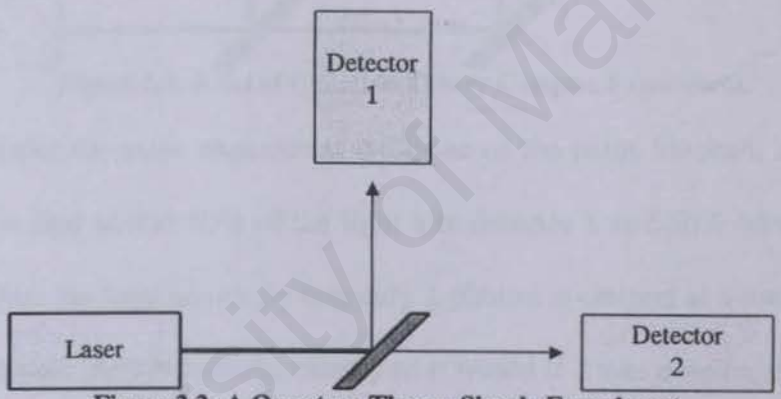


**Figure 2.2: A Quantum Theory Simple Experiment.**

Now consider an attempt to recombine the beam, as shown in Figure 2.3. The paths are set up to be exactly equal in length. What we find is that 100% of the light hits detector 1.

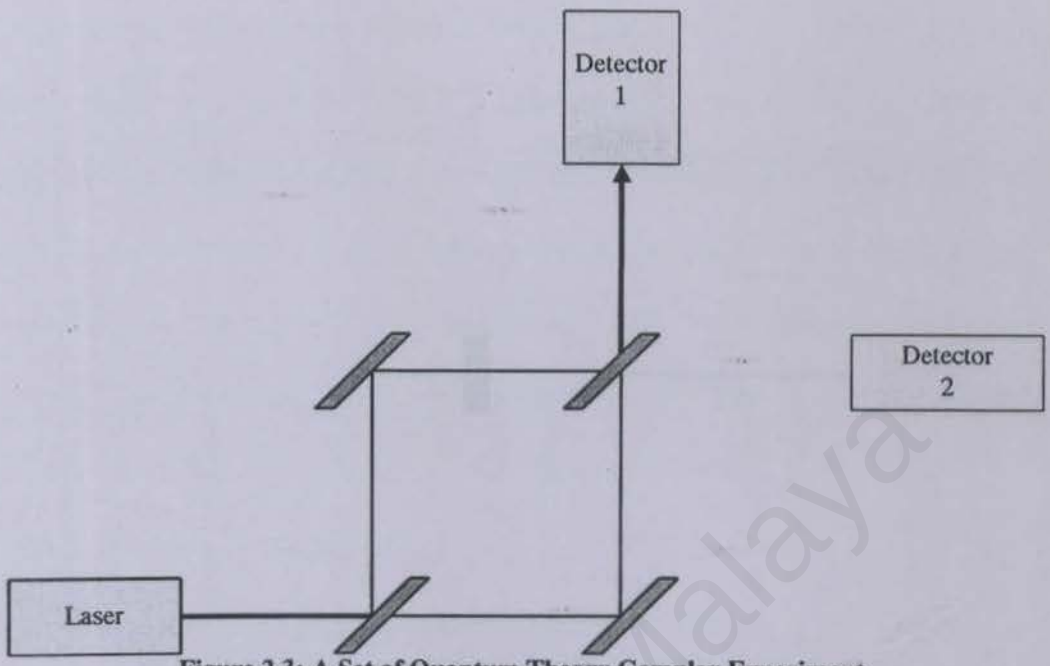


**Figure 2.3: A Set of Quantum Theory Complex Experiment.**

Now consider the same experiment with one of the paths blocked, as in Figure 2.4. What we now find is that 50% of the light hits detector 1 and 50% hits detector 2. Again we can adjust the light source so that only 1 photon is emitted at a time and again we find that this single photon behaves exactly as it would if it was a wave. In Figure 2.4 we know which path the electron has travelled as one of the paths is blocked. Yet how does this single photon know that this blockage on a remote path exists and that, with such a blockage, it must act differently when it reaches the second partially mirrored surface?

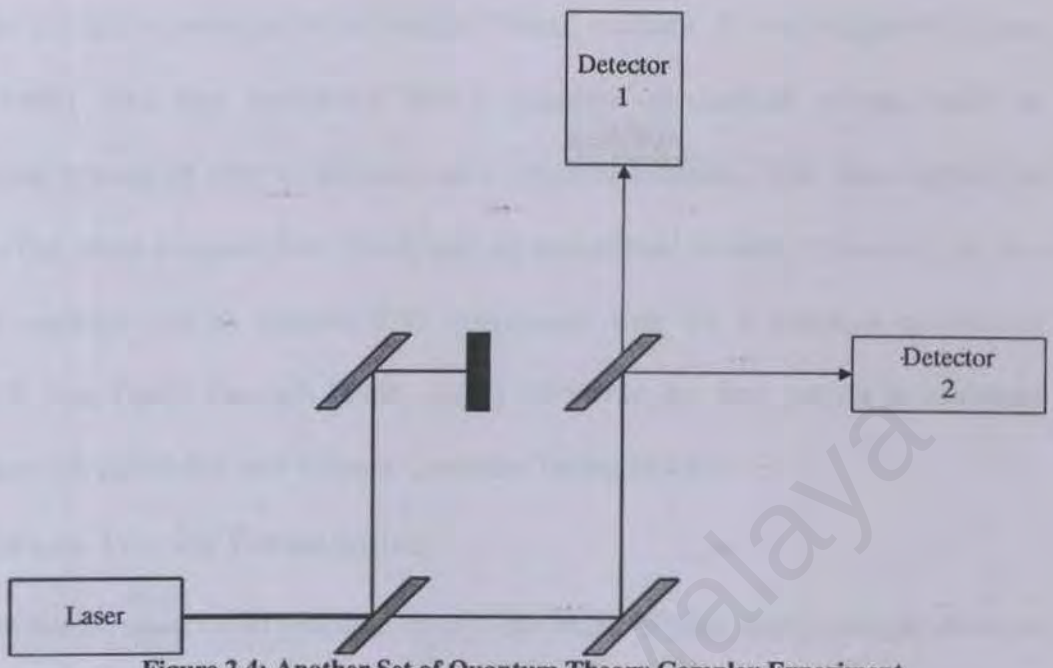


**Figure 2.4: Another Set of Quantum Theory Complex Experiment.**

Again it seems that we must accept the result that at quanta level, particles do not travel single paths, instead they travel to their destination by every possible path. Although this seems counterintuitive this area of quantum physics predicts measurements that agree with our observations to an astounding degree. For a more comprehensive explanation of the basic theory of quantum mechanics read Richard Feynman's excellent book, QED: The strange theory of light and matter.

If a photon travels to a destination via every possible path then can we use this quantum behaviour to our advantage in computing? It is clear that the universe is performing far more work in calculating the destination of a photon than one would expect from a classical point of view. As our machines are based on classical ideas of mathematics can this extra work by the quantum universe be converted in to extra computational power for our machines? The answer would appear to be a yes.

It was Benioff [1980, 1982a, 1982b] who showed that a machine whose computations were performed according to the laws of quantum mechanics physics

would be at least as powerful as a classical Turing machine. It was Richard Feynman [1982, 1986] who first postulated that a quantum mechanical system takes an exponential amount of time to simulate on a classical machine. This then implies the reverse - that some computations which take an exponential amount of time to run on a classical machine can be computed in polynomial time by a quantum mechanical system. It was David Deutsch [1985, 1989] who was the first person to seriously investigate this possibility and define a Quantum Turing Machine.

### 2.1.2 Many-Worlds Formulation

In one formulation of quantum theory, the Many Worlds interpretation, there are actually many copies of the universe which have certain probabilities of existing. In each universe the photon travels to its destination along one path. The interference that we see on the screen is due to the universes constructively and destructively interfering with one another. The universes where the photon strikes a dark patch of the screen have little possibility of existing while the universes with light patches have a reasonable chance of existing.

The Many Worlds theory originated with Dr Hugh Everett, III, is supported by some of the leading investigators in the field of quantum computation. The following in an extract from the "Many Worlds" faq by Michael Clive Price, which is available online.

*"Political scientist" L David Raub reports a poll of 72 of the "leading cosmologists and other quantum field theorists" about the "Many-Worlds Interpretation" and gives the following response breakdown.*

*1) "Yes, I think MWI is true"* 58%

*2) "No, I don't accept MWI"* 18%

*3) "Maybe it's true but I'm not yet convinced"* 13%

*4) "I have no opinion one way or the other"* 11%

*Amongst the "Yes, I think MWI is true" crowd listed are Stephen Hawking and Nobel Laureates Murray Gell-Mann and Richard Feynman. Gell-Mann and Hawking recorded reservations with the name "many-worlds", but not with the theory's content. Nobel Laureate Steven Weinberg is also mentioned as a many-worlder, although the suggestion is not when the poll was conducted, presumably before 1988 (when Feynman died). The only "No, I don't accept MWI" named is Penrose.*

The Many Worlds theory allows us a view of the behaviour of quantum computation which some find easier to visualise. The theory does differ from other quantum theories in its predicted results in certain areas. This opens up the possibility of being able to disprove the theory one day. Deutsch [1985] describes one possible experiment using an artificial intelligence computer built using quantum circuits. This experiment is currently far beyond our technical expertise.

The theory implies that there are many copies of you in many different universes. We are not aware of other copies because there can be no communication between the universes. This is disconcerting to a number of people who do not like their individuality to be impeached upon.

### 2.1.3 Superposition Particles

Imagine a hydrogen atom in its ground state. If we supply an amount of energy at the correct frequency for a certain period of time the atom will become excited. If we supply the energy for only half of this period then the atom will be in a superpositioned state - that is in some universes the atom will still be in a ground state and in other universes it will be in an excited state. Note that the particle is not in some intermediate state - it is definitely in one state or the other and measuring it will tell you which state the particle occupies in your universe. An otherwise identical copy of you in a different universe will have performed exactly the same measurement and will have seen the opposite result.

Again consider a particle in its ground state in universe X. Again we supply the correct amount of energy so that the particle enters a superpositioned state. The universe X will then split in to two universes: in universe U1 the particle is excited and in universe U2 the particle is still in its ground state. In all other aspects these two universes are absolutely identical. We can see this in Figure 2.5 where time increases along the x-axis.


**Figure 2.5: Split into Two Universes.**

Each of these universes will have an amplitude attached to it. The amplitude is a complex number that corresponds to the likelihood of that universe existing. What we would term as being the probability of the universe existing is the magnitude squared of this complex number, which obviously must lie on or between 0 and 1.

Now if we consider the above example again, what would happen if there was another route for the universe U1 to be created, as in Figure 2.6. Here universe Y can also split in to two universes, one of which is identical to U1. It is obvious that the probability of U1 occurring must now be affected as there are two paths leading to this universe. This is not to say that the probability of U1 increases. The amplitude of U1 is determined by the mathematical combination of the amplitudes of X and Y and of the amplitudes of X leading to U1 and Y leading U1. As amplitudes attached to these universes and actions are complex and may well involve negative numbers the probability of U1 existing may well be less in this example than it was in the previous example. Hence the universe may well destructively interfere as well as constructively so, and this is why we see dark fringes in Young's double slit experiment.

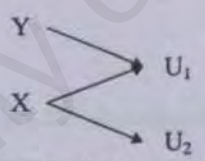


**Figure 2.6: Interference of Universes.**

It is important to note that the universes must be absolutely identical for interference to occur. Again consider a particle in a universe X which is then transferred in to a superpositioned state in universes U1 and U2. Now we measure the particle. In universe U1 you would see that the particle is excited and you and the particle would enter universe U3, and in U2 you would see that it is in the ground state and enter universe U4. By this process of measuring the state of billions of particles in your brain have been affected. The difference between U3 and U4 would not be one particle, as with the differences of U1 and U2, but billions of particles. U3 and U4 would be so far apart that there would never be any hope of the two universes becoming identical at some point in the future. Thus they will never interfere again. We call this process

decoherence, and it can be seen in Figure 2.7 with the y-axis representing a qualitative measure of the distance between universes (that is to say how different they are).

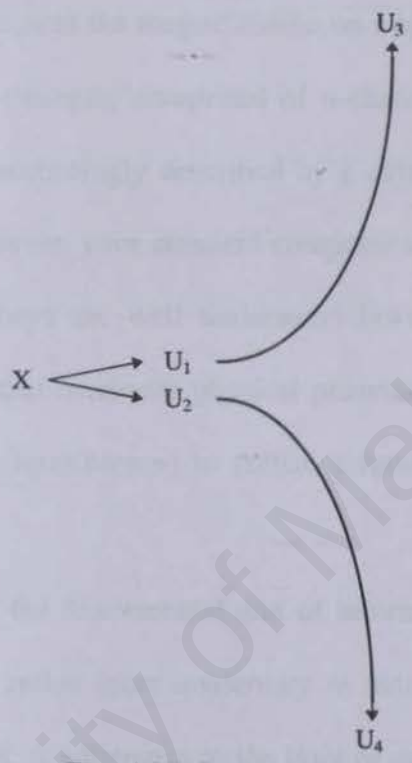


**Figure 2.7: Decoherence.**

## 2.2 Quantum Computing

### 2.2.1 What is a Quantum Computer?

Current modern computer represents the culmination of years of technological advancements beginning with the early ideas of Charles Babbage (1791-1871) and eventual creation of the first computer by German engineer Konrad Zuse in 1941. Surprisingly however, the high speed modern computer sitting in front of you is fundamentally no different from its gargantuan 30 ton ancestors, which were equipped with some 18000 vacuum tubes and 500 miles of wiring! Although computers have become more compact and considerably faster in performing their task, the task remains the same: to manipulate and interpret an encoding of binary bits into a useful

computational result. A bit is a fundamental unit of information, classically represented as a 0 or 1 in your digital computer. Each classical bit is physically realized through a macroscopic physical system, such as the magnetization on a hard disk or the charge on a capacitor. A document, for example, comprised of n-characters stored on the hard drive of a typical computer is accordingly described by a string of 8n zeros and ones. Herein lies a key difference between your classical computer and a quantum computer. Where a classical computer obeys the well understood laws of classical physics, a quantum computer is a device that harnesses physical phenomenon unique to quantum mechanics (especially quantum interference) to realize a fundamentally new mode of information processing.

In a quantum computer, the fundamental unit of information (called a quantum bit or qubit), is not binary but rather more quaternary in nature. This qubit property arises as a direct consequence of its adherence to the laws of quantum mechanics which differ radically from the laws of classical physics. A qubit can exist not only in a state corresponding to the logical state 0 or 1 as in a classical bit, but also in states corresponding to a blend or superposition of these classical states. In other words, a qubit can exist as a zero, a one, or simultaneously as both 0 and 1, with a numerical coefficient representing the probability for each state. This may seem counterintuitive because everyday phenomenon are governed by classical physics, not quantum mechanics -- which takes over at the atomic level.

### 2.2.2 Qubits

We define a quantum bit (Qubit) as being a particle in the superposition of two values, which we will denote $|0\rangle$ and $|1\rangle$. Here we use the ket notation $|\rangle$ to remind us

that we are dealing with quantum systems and not classical ones. For instance, we could denote the excited state of a superpositioned particle as being |1> and the ground state as being |0>.

Let's place a particle in to the superposition of two values, |0> and |1>. Again we get two universes, each with an assigned amplitude. If we were to then to place another particle in a superposition of two values we'd then get four universes, as shown in Figure 2.8. If we were to repeat this process then we would get eight universes, and so on.

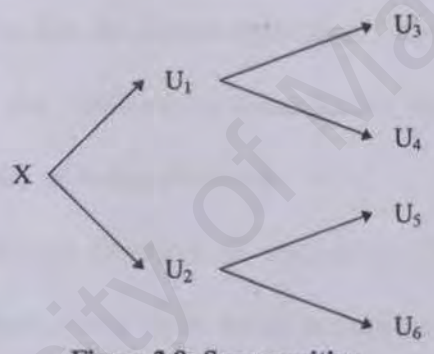


**Figure 2.8: Superposition.**

In fact for n superpositioned particles in two possible states we get 2n different universes with every possible combination of the n particles values being observed. As with classical machines we can call a collection of bits a register. However, the difference is that a classical register can only hold one value. A quantum register of length n bits can hold up to 2n values simultaneously with each value observed in an otherwise identical universe. This quantum register could be used as the input to some circuit. The circuit will then act simultaneously on these 2n different inputs, perform 2n different calculations and output 2n superpositioned results. This is the source of the exponential speed-up of quantum computers, and has also been dubbed parallel

processing on a serial machine. In essence, for the cost of building only one circuit we can have the circuit perform an exponential number of calculations simultaneously.

The trick is to get all of these universes to interfere with each other in such a way as to produce an output that is of some use to us. Consider the situation where we have the functions in the 2n universes outputting a different value with equal probability. If we were to perform a measurement on the output value the systems would decohere and the value read would be a random value from the 2n outputs, which wouldn't really tell us very much. What's required is to arrange for the universes to interfere with each other with each other in such a way so that the output value(s) of interest have a much higher probability of being observed and, conversely, those values which are not of interest having a much smaller probability of being observed.

This leads to an interesting question. If we require that the output from a quantum circuit interferes in such a way as to make certain outputs being made more probable than others, then does this lead to a restriction on the type of class of problems which quantum circuits could perform more efficiently than their classical counterparts? Would it lead to a more efficient algorithm but with a less than exponential speed up? This question is analogous to the question of whether parallel processing machines can effectively speed up all problems or whether there are some inherently sequential problems that refuse to yield to parallel techniques. The answer to this question on quantum computation would appear to be yes, there is a limit to what quantum computation can speed up.

### 2.2.3 Superposition and Entanglement

Think of a qubit as an electron in a magnetic field. The electron's spin may be either in alignment with the field, which is known as a spin-up state, or opposite to the field, which is known as a spin-down state. Changing the electron's spin from one state to another is achieved by using a pulse of energy, such as from a laser - let's say that we use 1 unit of laser energy. But what if we only use half a unit of laser energy and completely isolate the particle from all external influences? According to quantum law, the particle then enters a superposition of states, in which it behaves as if it were in both states simultaneously. Each qubit utilized could take a superposition of both 0 and 1. Thus, the number of computations that a quantum computer could undertake is 2n, where n is the number of qubits used. A quantum computer comprised of 500 qubits would have a potential to do $2^{500}$ calculations in a single step. This is an awesome number - $2^{500}$ is infinitely more atoms than there are in the known universe (this is true parallel processing - classical computers today, even so called parallel processors, still only truly do one thing at a time: there are just two or more of them doing it). But how will these particles interact with each other? They would do so via quantum entanglement.

Particles (such as photons, electrons, or qubits) that have interacted at some point retain a type of connection and can be entangled with each other in pairs, in a process known as correlation. Knowing the spin state of one entangled particle - up or down - allows one to know that the spin of its mate is in the opposite direction. Even more amazing is the knowledge that, due to the phenomenon of superpostition, the measured particle has no single spin direction before being measured, but is simultaneously in both a spin-up and spin-down state. The spin state of the particle being measured is decided at the time of measurement and communicated to the correlated particle, which

simultaneously assumes the opposite spin direction to that of the measured particle. This is a real phenomenon (Einstein called it "spooky action at a distance"), the mechanism of which cannot, as yet, be explained by any theory - it simply must be taken as given. Quantum entanglement allows qubits that are separated by incredible distances to interact with each other instantaneously (not limited to the speed of light). No matter how great the distance between the correlated particles, they will remain entangled as long as they are isolated.

Taken together, quantum superposition and entanglement create an enormously enhanced computing power. Where a 2-bit register in an ordinary computer can store only one of four binary configurations (00, 01, 10, or 11) at any given time, a 2-qubit register in a quantum computer can store all four numbers simultaneously, because each qubit represents two values. If more qubits are added, the increased capacity is expanded exponentially.

### 2.2.4 Quantum Gates

Changes occurring to a quantum state can be described using the language of quantum computation. Analogous to the way a classical computer is built from an electrical circuit containing wires and logic gates, a quantum computer is built from a quantum circuit containing wires and elementary quantum gates to carry around and manipulate the quantum information.

Quantum NOT gate is a single qubit gate. It is analogous to classical NOT gate. The NOT gate is represented with X in a matrix form as $X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Operation on NOT gate is as:

For any $\alpha|0\rangle + \beta|1\rangle \equiv \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$, we get $X\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\beta\\ \alpha\end{bmatrix}$ or $\beta|0\rangle + \alpha|1\rangle$.

To summarize, quantum NOT gate does the following transformation:

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \beta|1\rangle + \alpha|0\rangle.$$

Z gate is also a single qubit gate and is represnted with Z in a matrix form as $Z \equiv \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$. The Z gate leaves $|0\rangle$ unchanged but flips the sign $|1\rangle$ to $-|1\rangle$. Another important and most useful single qubit gate is Hadamard gate. Hadamard gate is represented with H in a matrix form as $H \equiv \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$. It turns $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle$ into $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Figure below shows the summary of the functions of X, Z, and Hadamard gates:

$$\alpha|0\rangle + \beta|1\rangle ----X---- \beta|0\rangle + \alpha|1\rangle$$

$$\alpha|0\rangle + \beta|1\rangle ----Z---- \alpha|0\rangle - \beta|1\rangle$$

$$\alpha|0\rangle + \beta|1\rangle ----H---- \alpha\frac{|0\rangle + |1\rangle}{\sqrt{2}} + \beta\frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

**Figure 2.9: Summary of the Functions of X, Z and Hadamard Gates.**

Besides the mentioned and illustrated single qubit gates, there are also multiple qubit gates such as controlled-NOT or CNOT gate, controlled-U gate, controlled-Z gate, universal Toffoli gate, and Fredkin (controlled-swap) gate.

### 2.2.5 Quantum Algorithms

For certain problem solving, the quantum computer will never surpass traditional computers. An example of this would be function evaluations such as multiplication.

However, there are many other areas of computation that they would be well suited towards such as:

- Problem solving, for example factorizing.
- Proving that an input is true (though not necessarily providing the proof), for example, proving that a number is composite.
- Providing true randomization as opposed to present day computers which only provide pseudo random numbers.
- Undergoing an evolution that mimics a specified physical system e.g. simulating the behavior of a hurricane.
- Possibly producing intelligent machines that can pass the Turing test.

Many of the special properties of quantum computers are similar to properties that the mind has, e.g. correlating large amounts of different data very rapidly to useful conclusions, without necessarily producing proof for this (intuition). This would seem to suggest that there is a link between quantum computers and how the brain works, possibly leading to an artificially intelligent computer.

**Shor's Algorithm** is an algorithm invented by Peter Shor in 1995. It is the so called 'killer application' of quantum computers, due to its usefulness. It uses a quantum computer to crack public keys, a very popular method of encrypting data. The basic workings of public key encryption needed to be understood first before one can understand how the algorithm works.

Public Key Encryption is the main method for sending encrypted data. It works by using two keys one public and one private. The public key is used to encrypt the data, while the private key is used to decrypt the data. The public key can be easily derived

from the private key but not visa versa. This system relies on the extreme difficulty of factorising large numbers. An eavesdropper who knows your public key can in principle calculate your private key as they are mathematically related but the difficulty of computing the private key is the problem of factorizing large integers. For example, multiplying 1234 by 3433 is easy to work out, but calculating the factors of 4236322 is not so easy. The difficulty of factorizing grows rapidly with the size. It took 8 months and 1600 Internet users to crack RSA 129. Encrypters thought that more digits could be added as conventional computers increased in speed, i.e. it would take longer than the age of the universe to calculate RSA 140. However, using a quantum computer, which is running Shor's algorithm, a key can be cracked in seconds. This is due to the algorithm being able to parallel process on an unprecedented scale.

There are three main stages to Shor's Algorithm as will be detailed below:

- Take a memory register and place it into a quantum superposition of states (if you had a two bit register, it would be in the states 00, 01, 10, 11 at the same time).
- A calculation is made on the register (and hence on each different value in the register). First of all a random number x is chosen (between 0 and n). Raise this number to the power in the register. Divide this number by n, and place the remainder in a second register. The numbers in the second register will start to repeat at a certain frequency (f).
- Plug into the formula (x.f/ 2) - 1. An example of this would be trying to factorise 15 using x = 3. This would give powers of 3, 9, 27, 81, 243, 729 etc.

These have a repeating pattern of 3, 9, 12, 6, 3, 9 etc., giving the frequency as four. Putting it in the formula gives (3 x 4) / 2 - 1 = 5 which is a factor of 15.

Shor worked out that incorrect answers had a tendency to cancel themselves out while correct ones reinforced each other. Thus an answer for the factors could be found. This technique does not get the correct answer all the time, however it is so quick to implement, it can be run over and over again.

Lov Grover has written an algorithm, which is widely know as **Grover's Algorithm** using quantum computers to search an unsorted database faster than a conventional computer could. Normally a database with N entries would take N/2 number of searches to find the data needed, but using a quantum computer it takes root N. For example, with a database holding 1 million entries instead of taking on average 500,000 searches it will only take 1000 searches (in this universe). With databases getting larger and being integrated together more, this would mean a significant saving in time.

Grover's algorithm has another very useful application, in the field of cracking encrypted data. We are interested in the situation where a virtual database is so large that it would not fit in the memories of all the world's computers. This allows a quantum computer to crack another widely used system to protect data. This is the Data Encryption Standard. DES relies on a 56 bit number which both participants must know before hand. If an eavesdropper intercepts clear and ciphered text then his goal is to find the key so that any future text can be decoded. An exhaustive search by conventional means would make it necessary to search 2 to the power 55 keys before hitting the correct one. This would take more than a year even if one billion keys were tried every second. By comparison Grover's algorithm could find the DES enciphering key after

only 185 searches before hitting the correct one. For conventional DES, a method to stop modern computers from cracking the code (i.e. if they got faster) would be simply to add extra digits to the code which would increase the number of searches needed exponentially. This does not happen similarly in a quantum computer crack.

Grover also stated that quantum computers would be talented statisticians, excelling in finding single numbers which depend collectively on lots of data e.g. median age of a population.

### 2.2.6 Brief History of Quantum Computation

The idea of a computational device based on quantum mechanics was first explored in the 1970's and early 1980's by physicists and computer scientists such as Charles H. Bennett of the IBM Thomas J. Watson Research Center, Paul A. Benioff of Argonne National Laboratory in Illinois, David Deutsch of the University of Oxford, and the late Richard P. Feynman of the California Institute of Technology (Caltech).

The idea emerged when scientists were pondering the fundamental limits of computation. They understood that if technology continued to abide by Moore's Law, then the continually shrinking size of circuitry packed onto silicon chips would eventually reach a point where individual elements would be no larger than a few atoms. Here a problem arose because at the atomic scale the physical laws that govern the behavior and properties of the circuit are inherently quantum mechanical in nature, not classical. This then raised the question of whether a new kind of computer could be devised based on the principles of quantum physics.

Feynman was among the first to attempt to provide an answer to this question by producing an abstract model in 1982 that showed how a quantum system could be used

to do computations. He also explained how such a machine would be able to act as a simulator for quantum physics. In other words, a physicist would have the ability to carry out experiments in quantum physics inside a quantum mechanical computer.

Later, in 1985, Deutsch realized that Feynman's assertion could eventually lead to a general purpose quantum computer and published a crucial theoretical paper showing that any physical process, in principle, could be modeled perfectly by a quantum computer. Thus, a quantum computer would have capabilities far beyond those of any traditional classical computer. After Deutsch published this paper, the search began to find interesting applications for such a machine.

Unfortunately, all that could be found were a few rather contrived mathematical problems, until Shor circulated in 1994 a preprint of a paper in which he set out a method for using quantum computers to crack an important problem in number theory, namely factorization. He showed how an ensemble of mathematical operations, designed specifically for a quantum computer, could be organized to enable a such a machine to factor huge numbers extremely rapidly, much faster than is possible on conventional computers. With this breakthrough, quantum computing transformed from a mere academic curiosity directly into a national and world interest.

### 2.2.7 Potential and Power of Quantum Computation

In a traditional computer, information is encoded in a series of bits, and these bits are manipulated via Boolean logic gates arranged in succession to produce an end result. Similarly, a quantum computer manipulates qubits by executing a series of quantum gates, each a unitary transformation acting on a single qubit or pair of qubits. In applying these gates in succession, a quantum computer can perform a complicated

unitary transformation to a set of qubits in some initial state. The qubits can then be measured, with this measurement serving as the final computational result.

This similarity in calculation between a classical and quantum computer affords that in theory, a classical computer can accurately simulate a quantum computer. In other words, a classical computer would be able to do anything a quantum computer can. So why bother with quantum computers? Although a classical computer can theoretically simulate a quantum computer, it is incredibly inefficient, so much so that a classical computer is effectively incapable of performing many tasks that a quantum computer could perform with ease.

The simulation of a quantum computer on a classical one is a computationally hard problem because the correlations among quantum bits are qualitatively different from correlations among classical bits, as first explained by John Bell. Take for example a system of only a few hundred qubits, this exists in a Hilbert space of dimension ~1090 that in simulation would require a classical computer to work with exponentially large matrices (to perform calculations on each individual state, which is also represented as a matrix), meaning it would take an exponentially longer time than even a primitive quantum computer.

Richard Feynman was among the first to recognize the potential in quantum superposition for solving such problems much much faster. For example, a system of 500 qubits, which is impossible to simulate classically, represents a quantum superposition of as many as 2500 states. Each state would be classically equivalent to a single list of 500 1's and 0's. Any quantum operation on that system --a particular pulse of radio waves, for instance, whose action might be to execute a controlled-NOT operation on the 100th and 101st qubits would simultaneously operate on all 2500 states.

Hence with one fell swoop, one tick of the computer clock, a quantum operation could compute not just on one machine state, as serial computers do, but on 2500 machine states at once! Eventually, however, observing the system would cause it to collapse into a single quantum state corresponding to a single answer, a single list of 500 1's and 0's, as dictated by the measurement axiom of quantum mechanics. The reason this is an exciting result is because this answer, derived from the massive quantum parallelism achieved through superposition, is the equivalent of performing the same operation on a classical super computer with ~10150 separate processors (which is of course impossible).

Early investigators in this field were naturally excited by the potential of such immense computing power, and soon after realizing its potential, the hunt was on to find something interesting for a quantum computer to do. Peter Shor, a research and computer scientist at AT&T's Bell Laboratories in New Jersey, provided such an application by devising the first quantum computer algorithm. Shor's algorithm harnesses the power of quantum superposition to rapidly factor very large numbers (on the order ~10200 digits and greater) in a matter of seconds. The premier application of a quantum computer capable of implementing this algorithm lies in the field of encryption, where one common (and best) encryption code, known as RSA, relies heavily on the difficulty of factoring very large composite numbers into their primes. A computer which can do this easily is naturally of great interest to numerous government agencies that use RSA -- previously considered to be "uncrackable" -- and anyone interested in electronic and financial privacy.

Encryption, however, is only one application of a quantum computer. In addition, Shor has put together a toolbox of mathematical operations that can only be

performed on a quantum computer, many of which he used in his factorization algorithm. Furthermore, Feynman asserted that a quantum computer could function as a kind of simulator for quantum physics, potentially opening the doors to many discoveries in the field. Currently the power and capability of a quantum computer is primarily theoretical speculation; the advent of the first fully functional quantum computer will undoubtedly bring many new and exciting applications.

### 2.2.8 Obstacles and Researches

The field of quantum information processing has made numerous promising advancements since its conception, including the building of two- and three-qubit quantum computers capable of some simple arithmetic and data sorting. However, a few potentially large obstacles still remain that prevent us from "just building one," or more precisely, building a quantum computer that can rival today's modern digital computer. Among these difficulties, error correction, decoherence, and hardware architecture are probably the most formidable. Error correction is rather self explanatory, but what errors need correction?

The answer is primarily those errors that arise as a direct result of decoherence, or the tendency of a quantum computer to decay from a given quantum state into an incoherent state as it interacts, or entangles, with the state of the environment. These interactions between the environment and qubits are unavoidable, and induce the breakdown of information stored in the quantum computer, and thus errors in computation. Before any quantum computer will be capable of solving hard problems, research must devise a way to maintain decoherence and other potential sources of error at an acceptable level.

Thanks to the theory (and now reality) of quantum error correction, first proposed in 1995 and continually developed since, small scale quantum computers have been built and the prospects of large quantum computers are looking up. Probably the most important idea in this field is the application of error correction in phase coherence as a means to extract information and reduce error in a quantum system without actually measuring that system. In 1998, researches at Los Alamos National Laboratory and MIT led by Raymond Laflamme managed to spread a single bit of quantum information (qubit) across three nuclear spins in each molecule of a liquid solution of alanine or trichloroethylene molecules. They accomplished this using the techniques of nuclear magnetic resonance (NMR). This experiment is significant because spreading out the information actually made it harder to corrupt.

Quantum mechanics tells us that directly measuring the state of a qubit invariably destroys the superposition of states in which it exists, forcing it to become either a 0 or 1. The technique of spreading out the information allows researchers to utilize the property of entanglement to study the interactions between states as an indirect method for analyzing the quantum information. Rather than a direct measurement, the group compared the spins to see if any new differences arose between them without learning the information itself. This technique gave them the ability to detect and fix errors in a qubit's phase coherence, and thus maintain a higher level of coherence in the quantum system. This milestone has provided argument against skeptics, and hope for believers. Currently, research in quantum error correction continues with groups at Caltech (Preskill, Kimble), Microsoft, Los Alamos, and elsewhere.

At this point, only a few of the benefits of quantum computation and quantum computers are readily obvious, but before more possibilities are uncovered theory must be put to the test. In order to do this, devices capable of quantum computation must be constructed. Quantum computing hardware is, however, still in its infancy. As a result of several significant experiments, nuclear magnetic resonance (NMR) has become the most popular component in quantum hardware architecture. Only within the past year, a group from Los Alamos National Laboratory and MIT constructed the first experimental demonstrations of a quantum computer using nuclear magnetic resonance (NMR) technology.

Currently, research is underway to discover methods for battling the destructive effects of decoherence, to develop an optimal hardware architecture for designing and building a quantum computer, and to further uncover quantum algorithms to utilize the immense computing power available in these devices. Naturally this pursuit is intimately related to quantum error correction codes and quantum algorithms, so a number of groups are doing simultaneous research in a number of these fields. To date, designs have involved ion traps, cavity quantum electrodynamics (QED), and NMR. Though these devices have had mild success in performing interesting experiments, the technologies each have serious limitations. Ion trap computers are limited in speed by the vibration frequency of the modes in the trap. NMR devices have an exponential attenuation of signal to noise as the number of qubits in a system increases. Cavity QED is slightly more promising; however, it still has only been demonstrated with a few qubits.

Seth Lloyd of MIT is currently a prominent researcher in quantum hardware. The future of quantum computer hardware architecture is likely to be very different from

what we know today; however, the current research has helped to provide insight as to what obstacles the future will hold for these devices.

## 2.3 Review of Development Tools and Technologies

### 2.3.1 Java Programming Platform and Language

#### 2.3.1.1 Java 2 Platform Standard Edition (J2SE)v1.4

The Java 2 Platform, Standard Edition is at the core of Java technology, and version 1.4 raises the Java platform to a higher standard. From client to server, from desktop to supercomputer, improvements have been made to J2SE across the board. With version 1.4, enterprises can now use Java technology to develop more demanding business applications with less effort and in less time.

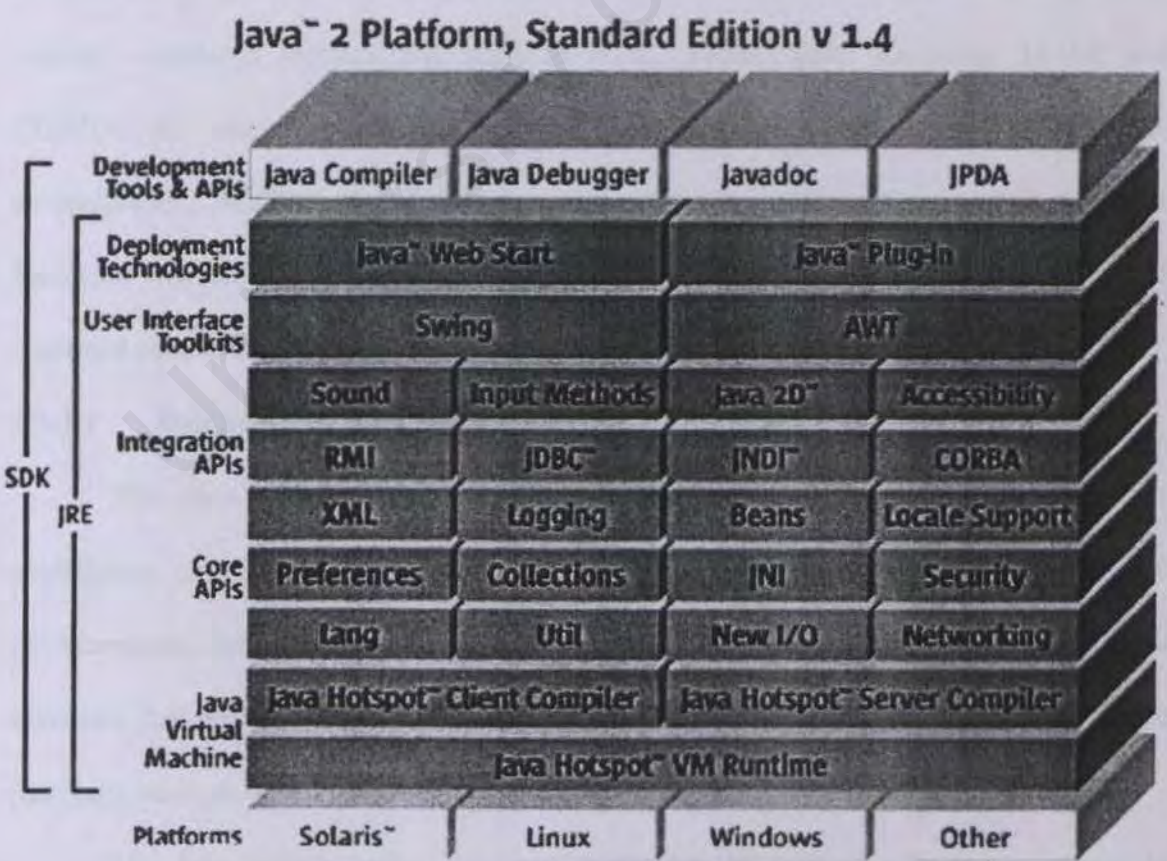


Figure 2.10: Overview of Java 2 Platform v1.4.

Version 1.4 builds upon the current J2SE platform and provides even more features for developers to build into their applications. More functionality in 1.4 means developers can now spend less time writing custom code to accomplish what is now part of the core J2SE platform. The result is faster application programming with more consistency for enterprise development initiatives. New features in J2SE 1.4 also reduce the developer's reliance on other technologies such as C or C++, PERL, or SSL and DOM implementations in browsers. This allows developers to use a single technology to develop, test, and deploy end-to-end enterprise applications and solutions. Most anything you want to do, you can do in J2SE version 1.4.

Version 1.4 provides more ways for developers leverage existing systems without changing their underlying platforms. Version 1.4 provides additional support for industry standards technologies such as XML, DOM, SSL, Kerberos, LDAP and CORBA to ensure operability across heterogeneous platforms, systems, and environments. Additionally, developers and software vendors may now use a new Endorsed Standards Override Mechanism in version 1.4 to provide newer versions of endorsed standards, such as CORBA, as they become available.

#### 2.3.1.2 Beginning of the Java Programming Language

The Java programming language is designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments. Paramount among these challenges is secure delivery of applications that consume the minimum of system resources, can run on any hardware and software platform, and can be extended dynamically.

The Java programming language originated as part of a research project to develop advanced software for a wide variety of network devices and embedded

systems. The goal was to develop a small, reliable, portable, distributed, real-time operating platform. When the project started, C++ was the language of choice. But over time the difficulties encountered with C++ grew to the point where the problems could best be addressed by creating an entirely new language platform. Design and architecture decisions drew from a variety of languages such as Eiffel, SmallTalk, Objective C, and Cedar/Mesa. The result is a language platform that has proven ideal for developing secure, distributed, network-based end-user applications in environments ranging from network-embedded devices to the World-Wide Web and the desktop.

#### 2.3.1.3 Design Goals of the Java Programming Language

The design requirements of the Java programming language are driven by the nature of the computing environments in which software must be deployed. The massive growth of the Internet and the World-Wide Web leads us to a completely new way of looking at development and distribution of software. To live in the world of electronic commerce and distribution, Java technology must enable the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks.

Operating on multiple platforms in heterogeneous networks invalidates the traditional schemes of binary distribution, release, upgrade, patch, and so on. To survive in this jungle, the Java programming language must be architecture neutral, portable, and dynamically adaptable.

The system that emerged to meet these needs is simple, so it can be easily programmed by most developers; familiar, so that current developers can easily learn the Java programming language; object oriented, to take advantage of modern software development methodologies and to fit into distributed client-server applications;

multithreaded, for high performance in applications that need to perform multiple concurrent activities, such as multimedia; and interpreted, for maximum portability and dynamic capabilities.

#### 2.3.1.4 Java Foundation Classes (JFC)

The Java Foundation Classes (JFC) are a set of Java class libraries provided as part of the Java platform to support building graphics user interface (GUI) and graphics functionality for Java technology-based client applications ("Java applications"). JFC includes an extensive set of technologies that enable developers to create a rich, interactive user experience for client applications that can run not only on Microsoft Windows, but also on other increasingly popular platforms, such as Mac OSX and Linux.

Features of the JFC are as listed below:

- Abstract Window Toolkit (AWT): APIs that enable programs to integrate into the native desktop window system, including APIs for Drag and Drop.
- Java 2D: APIs to enable advanced 2D graphics, imaging, text and printing.
- Swing GUI Components: APIs that extend the AWT to provide a rich, extensible GUI component library with a pluggable look and feel.
- Accessibility: APIs and assistive technologies for ensuring an application is accessible to users with disabilities and meets government requirements for accessibility.
- Internationalization: All JFC technologies include support for creating applications that can interact with users around the world using the user's

own language and cultural conventions. This includes the Input Method Framework API.

These five technologies are designed to be used together to enable developers to build fully functional GUI client applications that run and integrate on any client machine that supports J2SE, including Microsoft Windows, Solaris, Linux, and Mac OSX.

Several new enhancements have been made to the Java Foundation Classes/Swing APIs in J2SE 1.4:

- New JFC/Swing features include support for spinners, scrollable tabbed panes, and indeterminate progress bar controls.
- Swing component data can now be transferred between applications using cut, copy, and paste functions in addition to full drag-and-drop support across all Swing components.
- A new API for the long-term persistence of JavaBeans technology will allow developers to create UI designs that are portable between different Integrated Development Environments.
- To help applications adapt to different users and environments, a new Preferences API provides a way to store, retrieve, and modify data from applications.
- The new full-screen exclusive mode API supports high performance graphics by suspending the windowing system so that drawing can be done directly to the screen; a benefit to applications such as games.

- A redesigned focus architecture addresses many focus-related bugs caused by platform inconsistencies and incompatibilities between AWT and Swing components.
- Undecorated frames allow a Java application to turn off the creation of frame decorations such as native title bars, system menus, borders, or other native operating system dependent screen components.
- Other JFC improvements include an updated file chooser for Windows, a new Auditory Feedback mechanism, higher quality font rendering, mouse wheel support, Section 508 accessibility compliance, support for Macintosh style menu bars, and a comprehensive new Print Service API.

#### 2.3.1.5 Overview of Java 3D

It used to be that if one wanted to use Java to write a graphics program of any consequence, one had to write a lot of libraries to support his application. Writing those libraries required a good knowledge of computer graphics theory and mathematics. Not only was Java far too slow at the time, but also one had to reinvent an uncomfortably large amount of functionality before he could start writing an application.

Although it seems like yesterday, those days are long gone, and Java now provides a wide variety of graphics programming APIs in addition to acceptable performance. Many graphics programming tasks are greatly simplified, but it is still not very easy to do some things. In particular, working with 3-D graphics is no easy task. The Java 3D API provides all the raw functionality one could hope for, but assembling that functionality into a coherent program takes a good amount of work.

The Java 3D API is an optional package belonging to a broader set of APIs called the Java Media APIs. As is the case with other Java API collections, all of the individual APIs were not designed together and appear to have been lumped together after the fact. In particular, the Java 3D API appears to bear little relation to the Java 2D API. In addition to the Java Media APIs, Java 2D is also billed as belonging to the Java Foundation Classes (JFC) with which it integrates more cleanly. Therefore, do not expect that an understanding of Java 2D will help one much with Java 3D.

All 3-D graphics programs do not have the same requirements. Some require real-time interactive navigation of a virtual world. Others require only object modeling and rendering capabilities to generate static scenes. Yet others require specialized input devices or stereoscopic vision. Java 3D tries to meet all of the various requirements a 3-D program may impose. Therefore, it defines many classes and methods, many of which one may not be interested in. Zeroing in on the essentials can be difficult, even with a sample program or two and the Java 3D specification to guide.

Java 3D breaks down into two packages: javax.vecmath and javax.media.j3d. As its name implies, javax.vecmath contains all of the classes concerned with manipulating vectors and matrices. The javax.media.j3d package contains a big grab bag of everything else and could have benefited from being subdivided into several subpackages. The Java 3D reference implementation also includes about 20 com.sun packages that implement an assortment of utility and support functions. Many of the Java 3D demos use classes from the com.sun packages, and they implement commonly required high-level functionality, but they are not a formal part of the Java 3D API.

### 2.3.2 MATLAB (MATrix LABoratory)

#### 2.3.2.1 Beginning of MATLAB

The founders of The MathWorks recognized the need among engineers and scientists for more powerful and productive computation environments beyond those provided by languages such as Fortran and C. In response to that need, the founders combined their expertise in mathematics, engineering, and computer science to develop MATLAB, a high-performance technical computing environment. MATLAB combines comprehensive math and graphics functions with a powerful high-level language.

#### 2.3.2.2 What Is MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation.
- Algorithm development.
- Data acquisition.
- Modeling, simulation, and prototyping.
- Data analysis, exploration, and visualization.
- Scientific and engineering graphics.
- Application development, including graphical user interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

#### 2.3.2.3 The MATLAB System

The MATLAB system consists of five main parts:

- Development Environment – This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.
- The MATLAB Mathematical Function Library – This is a vast collection of computational algorithms ranging from elementary functions like sum, sine,

cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

- The MATLAB Language – This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.
- Graphics – MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.
- The MATLAB Application Program Interface (API) – This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

#### 2.3.2.4 MATLAB GUIDE (Graphical User Interface Development Environment)

GUIDE, the MATLAB Graphical User Interface development environment, provides a set of tools for creating GUIs. These tools greatly simplify the process of laying-out and programming a GUI.

When one opens a GUI in GUIDE, it is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools. The Layout Editor enables one to lay out a GUI quickly and easily by dragging components, such as push buttons, pop-up menus, or axes, from the component palette into the layout area. The following picture shows the Layout Editor:

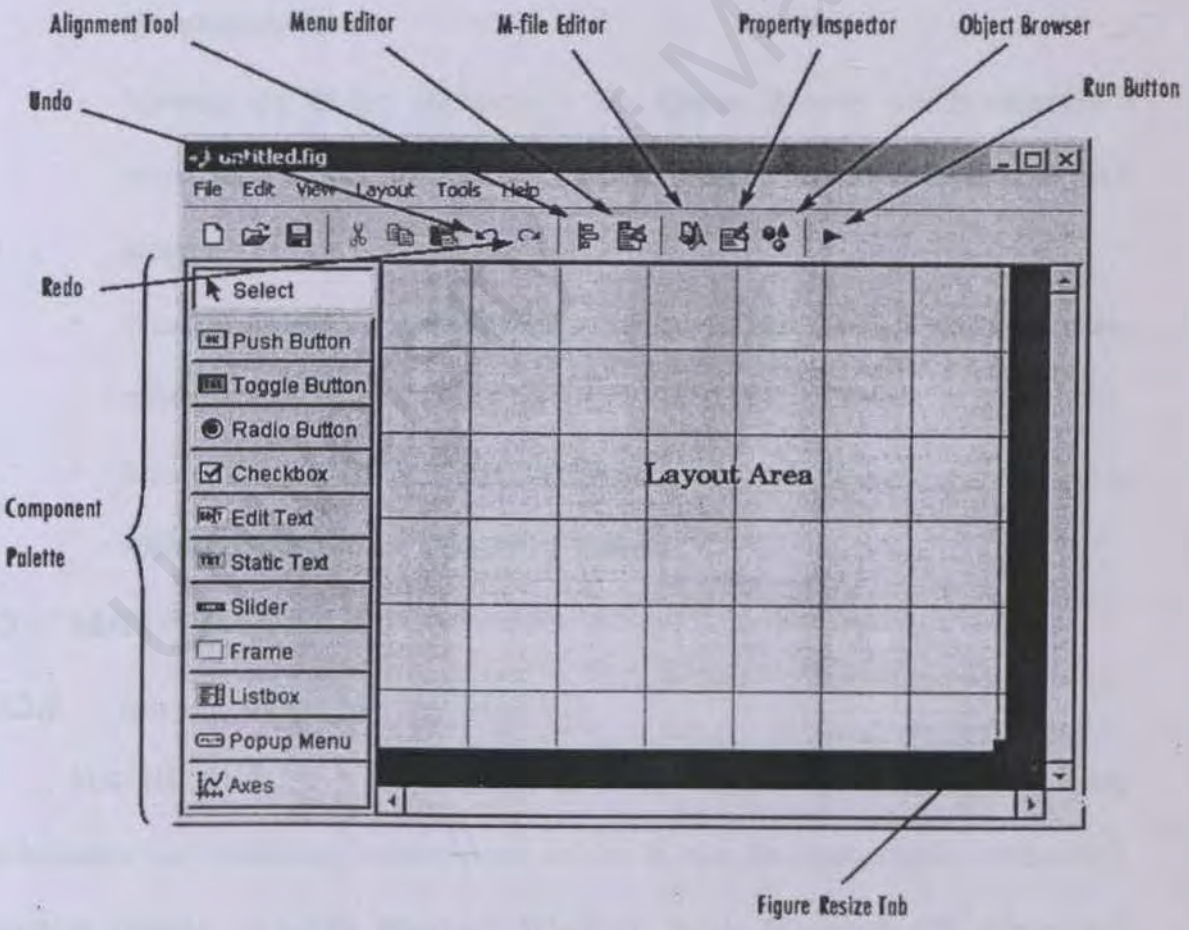


**Figure 2.11: MATLAB GUI Layout Editor.**

Once one lays out his GUI and set each component's properties, using the tools in the Layout Editor, he can program the GUI with the M-file Editor. Finally, when one

press the Run button on the toolbar, the functioning GUI appears outside the Layout Editor window.

The list of GUIDE toolsets is as below:

- Laying Out GUIs - The Layout Editor which adds and arranges objects in the figure window.
- Aligning Components in the Layout Editor - align objects with respect to each other.
- Setting Component Properties - The Property Inspector which inspects and sets property values.
- Viewing the Object Hierarchy - The Object Browser which observes a hierarchical list of the Handle Graphics objects in the current MATLAB session.
- Creating Menus - The Menu Editor which creates a menu bar or a context menu for any component in a layout.
- Setting the Tab Order - The Tab Order Editor which changes the order in which components are selected by tabbing.

## 2.3.3 MATHEMATICA

### 2.3.3.1 Introduction to MATHEMATICA

MATHEMATICA is a versatile, powerful application package for doing mathematics and publishing mathematical results. It runs on most popular workstation operating systems, including Microsoft Windows, Apple Macintosh OS, Linux, and other Unix-based systems.

MATHEMATICA is used by scientists and engineers in disciplines ranging from astronomy to zoology; typical applications include computational number theory, ecosystem modeling, financial derivatives pricing, quantum computation, statistical analysis, and hundreds more.

The best way to understand MATHEMATICA is to see it in action. The sections below describe three major categories of usage:

- User Tool: MATHEMATICA can be used to perform computations, either numeric or symbolic. Results can be visualized using 2-D and 3-D graphics.
- Programming Tool: MATHEMATICA provides a rich set of programming extensions to its end-user language. Programming can be done in procedural, functional, or logic (rule-based) style, or a mixture of all three. For tasks requiring interfaces to the external environment (such as extraction from a relational database) MATHEMATICA provides MathLink, which allows MATHEMATICA programs to communicate with external programs written in C, Java, or other languages.
- Publishing Tool: MATHEMATICA has extensive capabilities for formatting graphics, text, and equations. Documents, called notebooks, can be exported as PostScript, TeX, HTML, or a combination of HTML and MathML (Mathematical Markup Language).

# Chapter 3: Methodology and Technique

## 3.1 Methodology

The methodology adopted for the development process is important and crucial. Improper or inappropriate choice of methodology for Information Technology and software projects can lead to failure. There are two major factors of a software engineering project failure. The first problem is that too many design flaws are discovered during engineering or development where it is all difficult, expensive, and sometimes impossible to rectify and correct. The second problem is that the scopes of many projects seem to expand rampantly and are out of control as the time progresses.

Every system development process model includes system requirements such as users, constraints (limitations), and resources as inputs and a fully developed system or software as the outputs. There are many popular software process models such as:

- Waterfall Model.
- Waterfall Model with Prototyping.
- V Model.
- Prototyping.
- Operational Specification Model.
- Transformational Model.
- Phased Development Model.
- Incremental and Iterative Model.
- Spiral Model.
- Extreme Programming (XP) Model.

## 3.2 Extreme Programming (XP) Model

### 3.2.1 XP Model as the Project Methodology

The methodology employed to facilitate the design of the system for this project is based upon the Extreme Programming (XP) metaphor, developed by Kent Beck (Beck, 1999). Extreme Programming is a lightweight methodology that is test centric, and is based on an evolving design strategy. The methodology prescribes that functional tests should be defined before any work on the program begins.

A functional test is one that tests whether the program fulfills the specifications. In the case of a quantum computer simulator, the functional tests would be quantum algorithms, and the test would be successful if the output of the program is correct. Unit, or "programmer", tests would ensure that methods function as they should. All unit tests should run successfully at all times, and the aim of the development process is to reach a stage where the functional tests work at 100% correctness. This is done using an iterative test, program, design cycle that continues until the project is complete.

Extreme Programming also defines and specifies documentation is developed with the programming, but kept to the minimum until the end result is achieved. No formal documentation is produced until the end of the project, and during the project the only documentation is usually UML diagrams of the existing system design, and the code comments.

Some aspects of the methodology have to be either ignored or adapted, as Extreme Programming is designed to work with development teams of a size usually greater than two people, working in conjunction with end users. In this instance the project team consists of one individual and there is no immediate end user. Therefore it

is impossible to employ some portions of the methodology, such as pair programming, and user involvement at all stages.

### 3.2.2 XP Model in Greater Details

Extreme Programming (XP) is actually a deliberate and disciplined approach to software development. About six years old, it has already been proven at many companies of all different sizes and industries world wide.

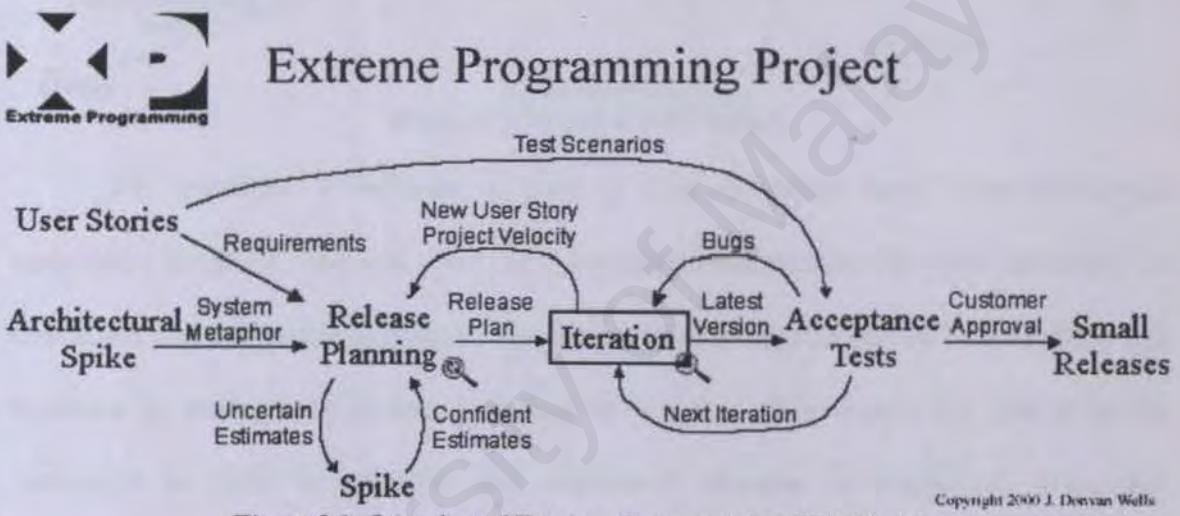


**Figure 3.1: Overview of Extreme Programming (XP) Model.**

XP is successful because it stresses end-user or customer satisfaction. The methodology is designed to deliver the software user needs when it is needed. XP empowers developers to confidently respond to changing user requirements, even late in the life cycle.

This methodology also emphasizes team work. Managers, customers, and developers are all part of a team dedicated to delivering quality software (for instance in this final year project, Dr. Selva the supervisor and manager as and me as the student and developer). XP implements a simple, yet effective way to enable groupware style development.

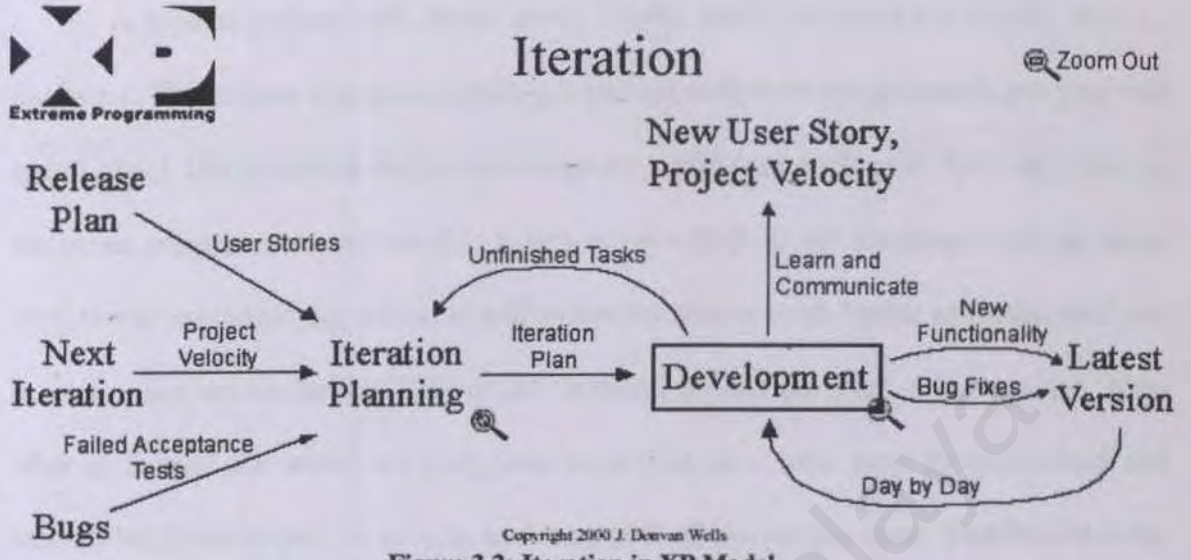


**Figure 3.2: Iteration in XP Model.**

XP improves a software project in four essential ways; communication, simplicity, feedback, and courage. XP programmers communicate with end-users or customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. With this foundation XP programmers are able to courageously respond to changing requirements and technology.

XP is different. It is a lot like a jig saw puzzle. There are many small pieces. Individually the pieces make no sense, but when combined together a complete picture can be seen. This is a significant departure from traditional software development methods and ushers in a change in the way we program.

Software which is engineered to be simple and elegant is no more valuable than software that is complex and hard to maintain. Can this really be true? Extreme Programming (XP) is based on the idea that this is not in fact true.

A typical project will spend about twenty times as much on people than on hardware. That means a project spending 2 million dollars on programmers per year will spend about 100 thousand dollars on computer equipment each year. Let's say that we are smart programmers and we find a way to save 20% of the hardware costs by some very clever programming tricks. It will make the source code harder to understand and maintain, but we are saving 20% or 20 thousand dollars per year, a big savings. Now what if instead we wrote our programs such that they were easy to understand and extend. We could expect to save no less than 10% of our people costs. That would come to 200 thousand dollars, a much bigger savings.

Another important issue to users or customers is concerning bugs. XP emphasizes not just testing, but testing well. Tests are automated and provide a safety net for programmers and customers alike. Tests are created before the code is written, while the code is written, and after the code is written. As bugs are found new tests are added. A safety net of tight mesh is created. Bugs don't get through twice, and this is something the customers will notice.

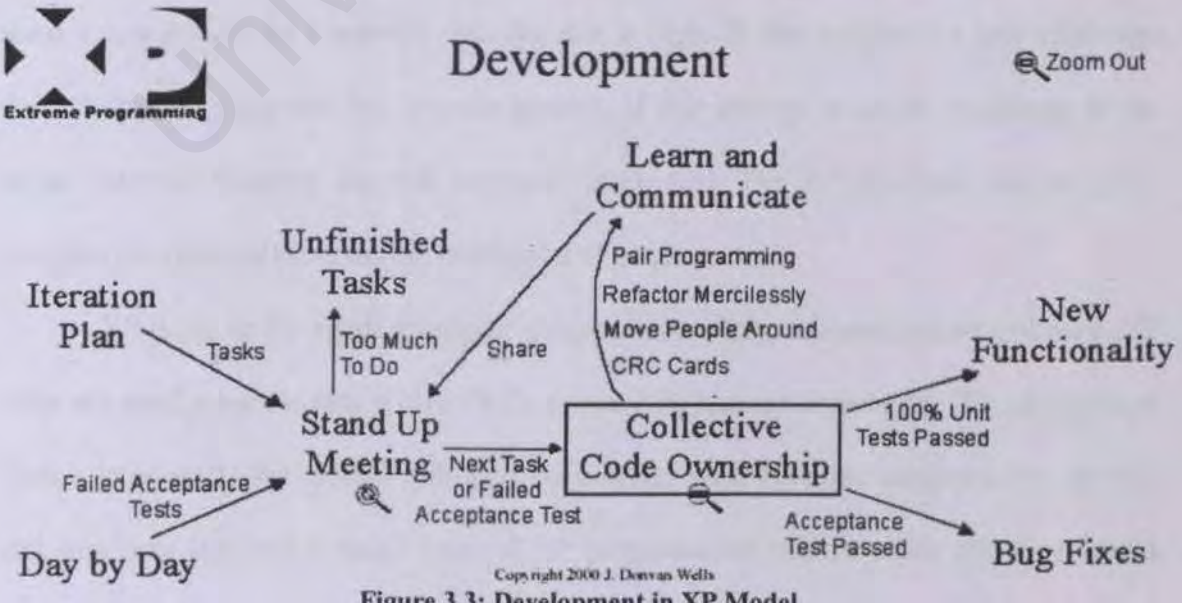


**Figure 3.3: Development in XP Model.**

Another thing that is distinguishable with other methodologies is the attitude XP programmers have towards changing requirements. XP enables programmers to embrace change. Too often a user or customer will see a real opportunity for making a system useful after it has been delivered. XP short cuts this by getting user or customer feed back early while there is still time to change functionality or improve user acceptance.

Much of what went into XP was a re-evaluation of the way software was created. The quality of the source code is much more important than one might realize. Just because users can't see the source code doesn't mean programmers shouldn't put the effort into creating something they can be proud of.

Extreme Programming (XP) was created in response to problem domains whose requirements change. Users or customers may not have a firm idea of what the system should do. Developers may have a system whose functionality is expected to change every few months. In many software environments dynamically changing requirements is the only constant. This is when XP will succeed while other methodologies do not.

XP was also set up to address the problems of project risk. If users or customers need a new system by a specific date the risk is high. If that system is a new challenge for a software group the risk is even greater. If that system is a new challenge to the entire software industry the risk is greater even still. The XP practices are set up to mitigate the risk and increase the likelihood of success.

XP is set up for small groups of programmers. Programmers can be ordinary, XP does not need programmers with a Ph.D. to use XP. But one cannot use XP on a project with a huge staff. We should note that on projects with dynamic requirements or high risk you may find that a small team of XP programmers will be more effective than a large team anyway.

XP requires an extended development team. The XP team includes not only the developers, but the managers and customers as well, all working together elbow to elbow. Asking questions, negotiating scope and schedules, and creating functional tests require more than just the developers be involved in producing the software.

Another requirement is testability. You must be able to create automated unit and functional tests. While some domains will be disqualified by this requirement, you may be surprised how many are not. You do need to apply a little testing ingenuity in some domains. You may need to change your system design to be easier to test. XP adopts the principle where there is a will there is a way to test.

The last thing on the list is productivity. XP projects unanimously report greater programmer productivity when compared to other projects within the same corporate environment. But this was never a goal of the XP methodology. The real goal has always been to deliver the software that is needed when it is needed. If this is what is important to your project it may be time to try XP.

## 3.3 Techniques Used to Gather Information

Before the system is designed, just like any other system, big or small, simple or complex, appropriate techniques must be used to seek, discover, and determine all the requirements. Among the major techniques applied and employed to gather information regarding this project are as below:

1. **Internet**

   Internet and the World Wide Web (WWW) are the main sources of information regardless of field of studies. For this project, surfing and research of related information on the internet about this project is a must

and most important as there is not much information resource or reference about quantum computation (computing) in the library or bookshops. Mining of data, information, and knowledge on the internet, is inevitably most crucial and productive.

2. **Presentation and Discussion with Supervisor**

Dr. Selvanathan is indeed a helpful and contributive supervisor. He has been such an instrumental force in explaining and guiding me through all the basics, fundamentals, complexities, and mathematics of quantum computing. Besides that, he has been kind and benevolent; to understand and solve my difficulty and incapability in certain complicated matters.

3. **Studies on Existing Systems**

Surveying and studying on existing systems have helped me to improve my system design and have better insights and understanding of my system requirements. In attempting to know how other systems work, their functionalities, and their processes, I have a clearer picture how actually and exactly my system is going to perform and work out. As such, I am enlightened of the limitations and scope of my system and also, how and where can I improve my system to make it a better and proper one.

4. **Written Material**

Written materials could refer to related books, magazines, journals, previous theses, research papers or abstracts and articles available in electronic form (online) or in the form of hardcopy. All these materials, especially papers in the LANL and Oxford Quantum Computing Center

archives, are extremely useful and insightful, which have helped me a lot in gaining knowledge and understanding regarding quantum computing and the ways of simulating aspects of quantum computing. All these references have also revealed to me of the researches either done or still in work in all quantum computing research centers throughout the world.

# Chapter 4: System Analysis

A requirement is a feature of a system, or a description of something a system is capable of doing in order to fulfill its purposes. Requirements give a detailed explanation not only the flow of information to and from a system and, the transformation and processing of data by the system, but also the constraints on the system's performance and capabilities. Specifying requirements serves three main objectives:

- Allow developers or programmers to explain their understanding of how users want a system to work and function.
- Tell and instruct designers what functionalities and characteristics a resultant system is to have.
- Tell the developers what to demonstrate to convince the users that a particular system being delivered or developed is indeed as what was needed or ordered.

## 4.1 Functional Requirement

Functional requirement specifies a function that a system or a system's component must be able to perform. These are software requirements that define behaviors of a system, that is, the fundamental process or transformation that software and hardware components of the system perform on inputs to produce expected outputs.

As for this simulation system, there main subsystem functional requirement that will contribute and form the whole simulation system is the Quantum Growing Network Simulation.

There is another main functional module, which is the Graphical User Interface (GUI) system. The GUI must provide users an easy mean or platform of viewing, editing, and inputting data besides editing the system's configuration, inputting the user's details and changing the settings for simulation and visualization.

### 4.1.1 Quantum Growing Network Simulation

The features and requirements of the Quantum Growing Network Simulation System are as:

- Simulating and visualizing the growing tree.
- Simulating and modeling the relations between quantum vacuum states and virtual states.
- Simulating how quantum virtual states affect the growth and its growing speed.

## 4.2 Non-Functional Requirement

Non-functional requirement does not describe what a system or software will do or process, but HOW it does. For example, software performance requirements, some external interface requirements, software design constraints, and software quality attributes. Non-functional requirements are difficult to test; they are usually, or most of the time, evaluated subjectively. Non-functional requirements have been recognized and acknowledged as a vital and crucial determinant to the success of many software projects.

For this simulation project, the following requirements have been set:

1. Clarity and Simplicity

The simulation, 3D visualization and modeling must be clear and simple. Though it is to simulate rather complicated numeric and mathematical processes or equations, the simulation should be clear and sharp, and simple in order for the users to understand what is being simulated when the system is functioning. This is essential to making sure that users can see the ideas and effects of quantum states in a pictorial or graphical way, and not in complex and complicated text or mathematical way.

2. Maintainability

System maintenance always requires more effort and time if the system is not well planned and designed at the beginning. System maintenance is a must for this simulation system, just like any other systems, as it allows certain changes or modifications to be made over the system. Some changes may be like adding the effect of visualization and showing in more details the simulation process.

3. Efficiency

Efficiency is the ability of a process procedure to be called or accessed unlimitedly to produce similar performance outcomes at an acceptable or credible speed. In this simulation system, efficiency comes into picture as how fast the system can process the mathematical notations and equations, and then start to simulate what is require. It is also regarding how well and fast the system can handle and load the 3D visualization.

4. Flexibility

The simulation is a flexible system as it is a standalone system and can be actually executed over many Microsoft and Unix/Linux operating systems.

5. User Friendliness

In this simulation system, the user interface design aims to fulfill three golden rules of user friendliness, which are:

- Place User in Control

  This will define interaction modes in a way that does not force a user into unnecessary or undesired actions or situations. Besides, it also provides flexible interaction, for instance, via mouse movement and keyboard commands.

- Reduce the user's memory load

  One of the principles that enable an interface to reduce the user's memory load is by reducing demand on short term memory. The interface should be designed to reduce and minimize the memory needed to load and execute the system.

- Make the interface consistent

  The interface design should conform to consistent fashion where all visual information must be organized according to a design standard that is maintained throughout all screen displays. Apart from that, input mechanisms are restricted to limited sets that are used consistently throughout the application.

- Accuracy and Correctness

Accuracy means how close or near an output produced by a system to a desired or perfect output as calculated mathematically or according to theory. Correctness is the degree to which the software performs its required functions. To ensure that this simulation system meets its requirements, it will be reviewed from time to time together with Dr. Selva. This is important to assure the quality and maturity of the system, and to ensure that the users will comprehend and get the knowledge correctly and not misunderstand what the system is simulating.

## 4.3 Hardware and Software Requirements

Hardware includes any physical (that can be touched) device or peripheral that is connected to a computer and is controlled by the computer's microprocessor. In this quantum simulation system, no special or specific high-end hardware is required or needed. It just requires an IBM-compatible personal computer (PC), which is powerful enough to support Microsoft's operating systems, Java Development Toolkit (Java 2 SDK v1.4), and MATLAB R12/R13. Nevertheless, a PC with at least an Intel or AMD 1.0 GHz processor and 256 MB of RAM, is very much desired and recommended.

Software is a general term for the various kinds of programs used to operate computers and related devices. Software is often divided into system software and application software. System software is usually operating systems that support application software. Meanwhile, application software is programs that do work users are directly interested in.

Microsoft Windows XP Professional or Microsoft Windows 2000 has been chosen as the development platform and the choice of operating system for this quantum simulation program. The main reason is that most computers in campus and everywhere are using the Microsoft's operating system. Besides, it is more user friendly, reduces the training required for using the system, and most importantly, Windows-based applications are usually always easier to learn and use compared to other's platform applications in Unix or Linux.

The Application Development Tool needed to build and program the quantum simulation system are MATLAB 6.0 or MATLAB 6.5 (Release 12 or 13), and Java 2 Platform Standard Edition (J2SE) v1.4. Open source Java Integrated Development Environments (IDEs) like Netbeans and Eclipse, and commercial IDEs like Borland JBuilder and JCreator might be used to speed up development and debugging.

# Chapter 5: System Design

## 5.1 Quantum Growing Network Simulation Design

### 5.1.1 Mathematical Equations (Vacuum and Virtual States)

The initial state or input of a quantum register is generally taken to be n string where all qubits are "cooled" in the basis state $|0\rangle$. This is called the vacuum state of the quantum register: $|0000000...0\rangle$.

In a quantum growing network, the vacuum state grows at each time step, by an amount of 2n+3 states $|0\rangle$. Let Had(j) represents the Hadamard gate acting on bit j, and:

(1) $U = \prod_{j=1}^{N=(n+1)^2} Had(j)$. Let indicate with $v_n$ the number of virtual states at time $t_n$: (2) $v_n \equiv 2n+3$. Also, we shall indicate with $|virt\rangle_n$ and $|vac\rangle_n$ the virtual states and the vacuum states respectively. Also, let us define (3) $Un = \prod_{j=1}^{v=2n+3} Had(j)$ with n=0,1,2,...

The N qubits at time $t_n$ are given by the application of the operator in equation (1) to the vacuum: (4) $|\overline{N}\rangle = U|vac\rangle_n$. At each time step $t_n$, the virtual state $|virt\rangle_{n-1}$ occurring at time $t_{n-1}$ is transformed into v=2n+3 qubits by the operator $U_n$ in equation (3): (5) $U_n|virt\rangle_{n-1} = |\bar{1}\rangle^{\otimes v}$. The operator $U_n$ will be interpreted as the nodes "n" of the growing quantum network.

At the "unphysical" time $t_{-1}$ (N=0), it is, by definition: $|vac\rangle_{-1} = 1$, and $U_{-1} = 1$ (the node "-1" is the only inactive node). From equation (2) we get $v_{-1} \equiv 1$, then the virtual state is: $|virt\rangle_{-1} = |0\rangle$.

At time $t_0=t_p$, (N=1), we have: $|vac\rangle_0 = |0\rangle$. That means that the virtual state at time $t_{-1}$, turned into a vacuum state at time $t_0$: $|vac\rangle_0 = |virt\rangle_{-1} = |0\rangle$. At node "0", the virtual state $|virt\rangle_{-1} = |0\rangle$ (which in this case coincides with the vacuum state $|vac\rangle_0 = |0\rangle$) is operated on by the operator $U_0$ and transformed into one qubit: $U_0|0\rangle = Had|0\rangle = |\bar{1}\rangle$.

From equation (2), we get:$v_0=3$, then we have: $|virt\rangle_0 = |000\rangle$. At time t1=2tp, (N=4), we have: $|vac\rangle_1 = |0000\rangle$. From equation (2) we get: $v_1=5$, then we have: $|virt\rangle_1 = |00000\rangle$. $|vac\rangle_1$ can be written as: $|vac\rangle_1 = |virt\rangle_0 \otimes |vac\rangle_0 = |virt\rangle_0 \otimes |virt\rangle_{-1}$.

At node "1", the virtual state $|virt\rangle_0$ is operated on by the operator $U_1$ and transformed into 3 qubits: $U_1|virt\rangle_0 = H(1)H(2)H(3)|000\rangle = |\bar{1}\rangle^{\otimes 3}$. The 4 qubits at time $t_1$ are given by the application of the operator $U$ to the vacuum state: $U|vac\rangle_1 = \prod_{j=1}^{4} H(j)|0000\rangle = |\bar{4}\rangle^{\otimes 4} = |\bar{4}\rangle$, which can also be written as: $U|vac\rangle_1 \equiv \prod_{j=1}^{3} H(j)|000\rangle \otimes H|0\rangle \equiv U_1|virt\rangle_0 \otimes U_0|virt\rangle_{-1} \equiv |\bar{1}\rangle^{\otimes 3} \otimes |\bar{1}\rangle$.

At time $t_2=3t_p$, (N=9), we have: $|vac\rangle_2 = |000000000\rangle$. Also, it is: $v_2=7$, thus we get: $|virt\rangle_2 = |0000000\rangle$.

At node "2" the virtual state $|virt\rangle_1 = |00000\rangle$ is operated on by the operator $U_2$ and transformed into 5 qubits: $U_2|virt\rangle_1 = H(1)H(2)H(3)H(4)H(5)|0000\rangle = |\bar{1}\rangle^{\otimes 5}$. The 9 qubits at time $t_2$ are given by the application of the operator $U$ to the vacuum state: $U|vac\rangle_2 = \prod_{j=1}^{9} H(j)|000000000\rangle = |\bar{1}\rangle^{\otimes 9} = |\bar{9}\rangle$, which can also be written as:

$$U|vac\rangle_2 \equiv \prod_{j=1}^{5} H(j)|00000\rangle \otimes \prod_{j=1}^{3} H(j)|000\rangle \otimes H|0\rangle \equiv U_2|virt\rangle_1 \otimes U_1|virt\rangle_0 \otimes U_0|virt\rangle_{-1}$$

and it can be summarized as: $|\bar{1}\rangle^{\otimes 5} \otimes |\bar{1}\rangle^{\otimes 3} \otimes |\bar{1}\rangle = |\bar{9}\rangle$.

In general, the N-qubits state at time $t_n$ can be written as: (6)

$$|\overline{N}\rangle = U_n|virt\rangle_{n-1} \otimes U_{n-1}|virt\rangle_{n-2} \otimes ...U_1|virt\rangle_0 \otimes U_0|virt\rangle_{-1}.$$

The quantum algorithm is illustrated by the following family of quantum growing networks. The diagram below provides a schematic representation of each quantum network, where H represents the Hadamard gate.

At time $t_0$, we have a quantum network of size one:

(7) $|0\rangle$ ____ $H$ ____ $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$

At time $t_1$, we have a quantum network of size four:

(8)
$$\left.\begin{array}{l} |0\rangle \_\_\_\_ H \_\_\_\_ \frac{|0\rangle+|1\rangle}{\sqrt{2}} \\ |0\rangle \_\_\_\_ H \_\_\_\_ \frac{|0\rangle+|1\rangle}{\sqrt{2}} \\ |0\rangle \_\_\_\_ H \_\_\_\_ \frac{|0\rangle+|1\rangle}{\sqrt{2}} \\ |0\rangle \_\_\_\_ H \_\_\_\_ \frac{|0\rangle+|1\rangle}{\sqrt{2}} \end{array}\right\} \frac{1}{4}(|0000\rangle+|0001\rangle+|0010\rangle+...|1111\rangle)$$

And so on. In general, at time $t_n$, the quantum growing network has size $N = (n+1)^2$.

### 5.1.2 Explanation on Quantum Growing Network

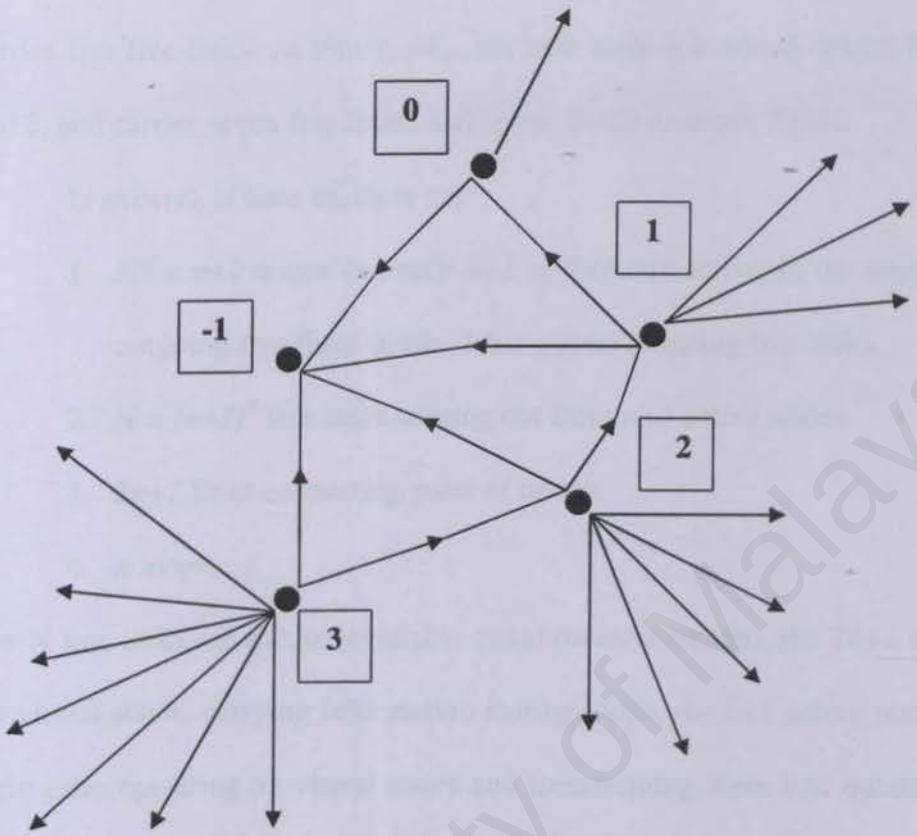


**Figure 5.1: Quantum Growing Network.**

The main function and design of this simulation system is to simulate the quantum growing network as discussed in the section 5.1. The growing network is further illustrated as in figure above. The growing tree will be the main visualization or simulation of this system

The design of the system will take into account how the tree, or in this context the quantum growing network, grows. At the starting time (the unphysical time $t_{-1}=0$). There is one node, call it **-1**. At each time step $t_n$, a new node is added, which links to the youngest and oldest nodes, and also carries 2n+1 free links. Thus, at the Planck time $t_0=t_p$, the new node 0 is added, which links to node **-1** and carries one free link. At time $t_1=2t_p$, the new node **1** is added, which links to node **-1** and node **0**, and carries three free

links. At time $t_2=3t_p$, the new node **2** is added, which links to node **-1** and node **1**, and carries five free links. At time $t_3=4t_p$, the new node **3** is added, which links to nodes **-1** and **2**, and carries seven free links, and so on. Refer to above figure.

In general, at time tn, there are:

1. $N^* = n+2$ nodes, but only $n+1$ of them are active, in the sense that they have outgoing free links (node **-1** has got no outgoing free link).
2. $N = (n+1)^2$ free links coming out from $n+1$ active nodes.
3. $2n+1$ links connecting pairs of nodes.
4. $n$ loops.

The N free links are qubits (available quantum information), the 2n+1 connecting links are virtual states, carrying information among loops, the n+1 active nodes are quantum logic gates operating on virtual states and transforming them into qubits. In fact, notice that the number of free outgoing links at node n is 2n+1, which is also the number of virtual states (connecting links) in the loops from node **-1** to node **n**.

## 5.2 User Interface Design

User interface design concerns how effectively users can use a system and how well they enjoy using it. A good interface makes it easy for users to tell the computer what they want to do, for the computer to request information from the users, and for the computer to present understandable information and visualization. Clear communication between the users and the computer is the working premise or platform of good user interface (UI) design.

For this quantum simulation system, the user interface aims for the following design principles:-

**1. Clear.**

A clear interface helps prevent user errors, make important information obvious, and contributes to ease of learning and use.

**2. Consistent.**

A consistent interface allows users to apply previously learned knowledge to new tasks. Effective applications are both consistent within themselves and consistent with another.

**3. Simple.**

The best interface designs are simple. Simple designs are easy to learn and to use and give the interface a consistent look. A design requires a good balance of maximizing functionality and maintaining simplicity through progressive disclosure of information.

**4. User-Controlled.**

The users, not the computer, initiate and control all actions.

**5. Direct.**

Users must be able to see the visible cause-and-effect relationships between the actions they take and the objects on the screen. This allows users to feel that they are in charge of the computer's activities.

**6. Provide Feedback.**

Keep the users informed and provide immediate feedback. Also, ensure that the feedback is appropriate to the task.

**7. Aesthetic.**

Every visual element that appears on the screen potentially competes for the users' attention. It provide an environment that is pleasant to use and contributes to the users' understanding of the information and simulation presented.

# Chapter 6: System Implementation

System implementation is the material realization phase of the system development. The conceptual and technical designs from the system analysis phase are interpreted as well as modeled to become the physical working system itself.

The following subchapters will explain the development environment as well as the development of the system itself, some system coding and the coding style and approach and object oriented technique applied in the Quantum Growing Network Simulation.

## 6.1 Development Environment

Development environment has a momentous influence on the development of a system. System development can be paced up significantly by utilizing the appropriate software and hardware. The following sections discuss the hardware and software tools used to develop and document the Quantum Growing Network Simulation.

### 6.1.1 Hardware in the Development Environment

The hardware configured for the development environment is the underlying element of the whole system. The hardware used in the system implementation phase plays an important role in realizing the final system architecture.

The hardware configuration of the development environment is as below:

- Intel Pentium IV Processor 1.8GHz.
- Memory – 256MB PC2100 of DDR Ram.
- Storage – 4 GB of Hard disk space is reserved.

- Other standard desktop PC component.

### 6.1.2 Software in the Development Environment

Hardware and software form a tightly coupled cohesion that operates in unison to performed programmed tasks. Without software, the fastest, biggest or the most powerful computer will also be inoperative and idling in the corner.

The software tools utilized in the development environment are listed as follow:

- Operating System – Microsoft Windows XP Professional Service Pack 1.
- Web browsers – Internet Explorer 6.0, Netscape Navigator 6.1.
- Java 2 Development Kit with Java Run Time (JDK 1.4.2).
- Xinox Software's JCreator LE v2.50.
- TextPad
- Documentation – Microsoft Office XP.

## 6.2 Development of the System

Development of the Quantum Growing Network Simulation system began with acquisition of knowledge and experience with the Java programming language. The following subchapters will detail the explanation of object oriented programming (OOP) approach, classes that are defined and created for the simulation system and other related coding parts of the entire simulation program or system.

### 6.2.1 Object Oriented Programming (OOP)

To be able to use Java as the programming language to code or to develop a program or a system, one needs to gain an understanding of the concepts of object oriented. Understanding of what an object is, what a class is, how objects and classes are related, how objects communicate by using messages is much needed.

In definition, an object is a software bundle of related variables and methods. "Objects" is a key to understanding object-oriented technology. One can look around now and see many examples of real-world objects: a car, a bicycle, a desk, a television set, a computer. These real-world objects share two characteristics: They all have state and behavior. For example, cars have state (brand, color, size) and behavior (accelerating, braking). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears).

Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object.

One can represent real-world objects by using software objects. You might want to represent real-world dogs as software objects in an animation program or a real-world bicycle as a software object in the program that controls an electronic exercise bike. You can also use software objects to model abstract concepts. For example, an event is a common object used in GUI window systems to represent the action of a user pressing a mouse button or a key on the keyboard.

Everything that the software object knows (state) and can do (behavior) is expressed by the variables and the methods within that object. A software object that modeled a real-world bicycle would have variables that indicated the bicycle's current state: its speed is 10 mph, its pedal cadence is 90 rpm, and its current gear is the 5th gear. These variables are formally known as instance variables because they contain the

state for a particular bicycle object, and in object-oriented terminology, a particular object is called an instance.

The following figure illustrates a bicycle modeled as a software object:

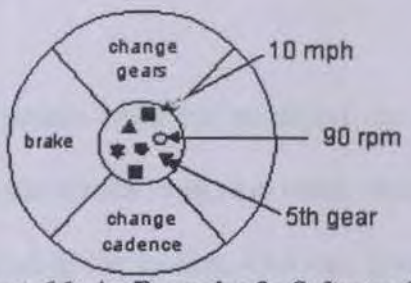


**Figure 6.1: An Example of a Software Object.**

In addition to its variables, the software bicycle would also have methods to brake, change the pedal cadence, and change gears. (The bike would not have a method for changing the speed of the bicycle, as the bike's speed is just a side effect of what gear it's in, how fast the rider is pedaling, whether the brakes are on, and how steep the hill is.) These methods are formally known as instance methods because they inspect or change the state of a particular bicycle instance.

The object diagrams show that the object's variables make up the center, or nucleus, of the object. Methods surround and hide the object's nucleus from other objects in the program. Packaging an object's variables within the protective custody of its methods is called encapsulation This conceptual picture of an object-a nucleus of variables packaged within a protective membrane of methods-is an ideal representation of an object and is the ideal that designers of object-oriented systems strive for. However, it's not the whole story. Often, for practical reasons, an object may wish to expose some of its variables or hide some of its methods. In the Java programming language, an object can specify one of four access levels for each of its variables and methods. The access level determines which other objects and classes can access that

variable or method. Variable and method access in Java is covered in Controlling Access to Members of a Class. Encapsulating related variables and methods into a neat software bundle is a simple yet powerful idea that provides two primary benefits to software developers:

- Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Also, an object can be easily passed around in the system. One can give a bicycle to someone else, and it will still work.
- Information hiding: An object has a public interface that other objects can use to communicate with it. The object can maintain private information and methods that can be changed at any time without affecting the other objects that depend on it. One does not need to understand the gear mechanism on his bike to use it.

A single object alone is generally not very useful. Instead, an object usually appears as a component of a larger program or application that contains many other objects. Through the interaction of these objects, programmers achieve higher-order functionality and more complex behavior. A bicycle hanging from a hook in the garage is just a bunch of titanium alloy and rubber; by itself, the bicycle is incapable of any activity. The bicycle is useful only when another object (cyclist) interacts with it (pedal).

Software objects interact and communicate with each other by sending messages to each other. When object A wants object B to perform one of B's methods, object A sends a message to object B.

Sometimes, the receiving object needs more information so that it knows exactly what to do; for example, when one wants to change gears on a bicycle, he has to indicate which gear he wants. This information is passed along with the message as parameters.

The next figure shows the three components that comprise a message:

- The object to which the message is addressed (Bicycle).
- The name of the method to perform (changeGears).
- Any parameters needed by the method (lowerGear).

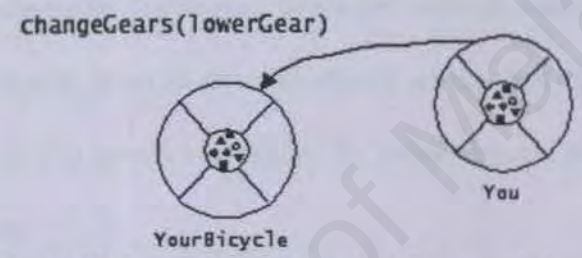


**Figure 6.2: Interaction (Messaging) Between Objects.**

These three components are enough information for the receiving object to perform the desired method. No other information or context is required.

Messages provide two important benefits:

- An object's behavior is expressed through its methods, so (aside from direct variable access) message passing supports all possible interactions between objects.
- Objects don't need to be in the same process or even on the same machine to send and receive messages back and forth to each other.

A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind. In the real world, one often have many objects of the same kind. For example, a bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that a bicycle object is an instance of the class of

objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles.

When building bicycles, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every individual bicycle manufactured.

In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like the bicycle manufacturers, one can take advantage of the fact that objects of the same kind are similar and he can create a blueprint for those objects. A software blueprint for objects is called a class.

The class for bicycle example would declare the instance variables necessary to contain the current gear, the current cadence, and so on, for each bicycle object. The class would also declare and provide implementations for the instance methods that allow the rider to change gears, brake, and change the pedaling cadence, as shown in the next figure:

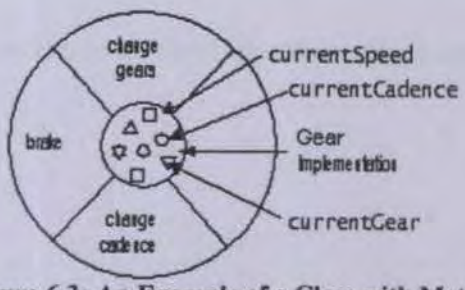


**Figure 6.3: An Example of a Class with Methods.**

After creating the bicycle class, one can create any number of bicycle objects from the class. When one creates an instance of a class, the system allocates enough

memory for the object and all its instance variables. Each instance gets its own copy of all the instance variables defined in the class.

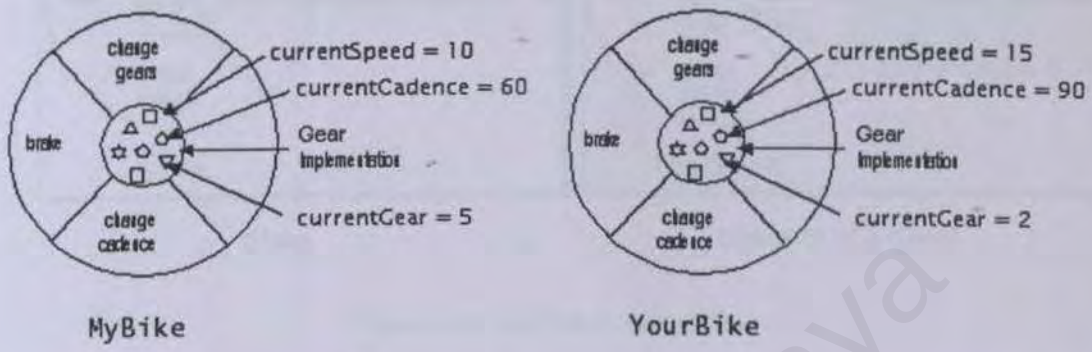


**Figure 6.4: Different Objects from a Same Class with Instances.**

In addition to instance variables, classes can define class variables. A class variable contains information that is shared by all instances of the class. For example, suppose that all bicycles had the same number of gears. In this case, defining an instance variable to hold the number of gears is inefficient; each instance would have its own copy of the variable, but the value would be the same for every instance. In such situations, one can define a class variable that contains the number of gears. All instances share this variable. If one object changes the variable, it changes for all other objects of that type. A class can also declare class methods. One can invoke a class method directly from the class, whereas he must invoke instance methods on a particular instance.

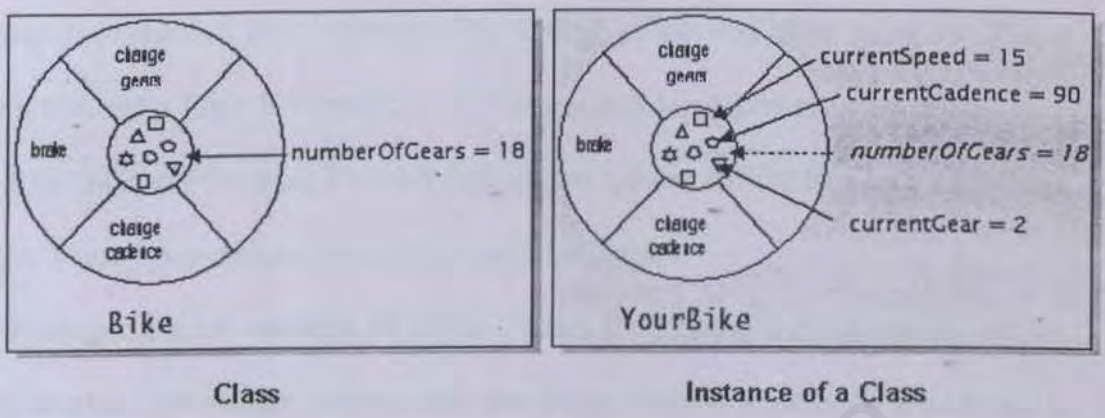


**Figure 6.5: An Object and a Class.**

One would probably notice that the illustrations of objects and classes look very similar. And indeed, the difference between classes and objects is often the source of some confusion. In the real world, it's obvious that classes are not themselves the objects they describe: A blueprint of a bicycle is not a bicycle. However, it's a little more difficult to differentiate classes and objects in software. This is partially because software objects are merely electronic models of real-world objects or abstract concepts in the first place. But it's also because the term "object" is sometimes used to refer to both classes and instances.

In the figures, the class is not shaded, because it represents a blueprint of an object rather than an object itself. In comparison, an object is shaded, indicating that the object exists and that one can use it.

### 6.2.2 System Coding

After researches and studies have been done, a decision was made to code the simulation system using the Java programming language, and to be able to run the simulation as a standalone windows application and as an applet which can be executed

using any Java-enabled web browsers. The coding phase was done using the Xinox Software's JCreator Light Edition (LE) v2.50 integrated development environment.

The Quantum Growing Network Simulation system utilizes two Java 2 Platform packages. These two packages are java.awt and java.applet.

Package java.awt contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components.

Some components fire events when a user interacts with the components. The AWTEvent class and its subclasses are used to represent the events that AWT components can fire.

A container is a component that can contain components and other containers. A container can also have a layout manager that controls the visual placement of components in the container. The AWT package contains several layout manager classes and an interface for building your own layout manager.

Package java.applet provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.

The applet framework involves two entities: the applet and the applet context. An applet is an embeddable window with a few extra methods that the applet context can use to initialize, start, and stop the applet.

The applet context is an application that is responsible for loading and running applets. For example, the applet context could be a Web browser or an applet development environment.

After deciding of the packages and Java built-in classes that can be used for the simulation system, it was time to decide on the entities or classes that can represent the Quantum Growing Network, and also its final outcome or deliverable, which is the visual of the quantum growing network. The result is to be able to visualize a network with nodes, virtual states linking nodes together, and free links branching out from active nodes and the algorithm of how the network grows (according to the description in Chapter 5).

The coding of this Java simulation system, after its final enhancement and refinement, is divided into five classes:

- Public class quantumnetwork, which extends Applet.
- Class GraphCanvas, which extends Canvas.
- Class GraphControls, which extends Panel.
- Class VisualNetwork.
- Class VisualNetworkNode, which extends Rectangle.

Public class quantumnetwork is the start class of the applet, in a sense it contains the main method, and the Java file is named after it. The main method will setup the application frame and start running the application. It also calls the init() method of the applet, to construct the layout and settings regarding the layout. Classes GraphCanvas and GraphControls are also started to be called from this class.

Class GraphControls which extends Panel is the class which constructs and manages the event performed on the three buttons of the simulation system. The three buttons are the "Visualize", "Next", and "Clear". The panel is responsible for communication between the user and the canvas GraphCanvas. When there is a request

from user, the panel redirect request to the canvas, and the canvas will call the corresponding functions in the class (network) VisualNetwork and display result over there.

Classs GraphCanvas contains the important paint() method. This method allows the network to be painted or in other word, visualized on the canvas of the system. Class VisualNetwork is actually the main working class of the entire simulation system. It is the class that actually constructs, manages and represents the whole network that is visualized on the canvas.

The network VisualNetwork is composed by different variables and objects. It includes:

- Root – stores the value of the root of the network.
- networkSize – stores the value of the size of the network.
- Perent – stores a reference to the parent (previous) node.
- NextNode – stores a reference to the next node.
- node – stored the graphical object of a particular node.
- positionFactor – stores the relative center position of a node. The actual position of a node is equal to this position factor + the center position of the parent node.
- nextPositionFactor – stores the next node's relative center position. So the actual position of the next node is equal to this next position factor + the position factor of a node.

Class VisualNetwork is defined as the manager of the quantum growing network being visualized. It constructs and takes care of the objects and variables related to the

network. It also manages the positioning of the nodes accordingly and tidily and draw related arcs which represent the virtual states and free links according to algorithm.

Class VisualNetworkNode which extends Rectangle, constructs and draws a node onto the network. It sets out the color and label of each node.

Refer to APPENDICES A and B for full source code and screenshots of the Quantum Growing Network Simulation system.

## 6.3 Coding Style

### 6.3.1 Formatting and Indenting Codes

Formatting and indenting codes is constantly associated to good coding practice. A code that is written without proper formatting or indenting will function or what as well as a formatted code. However, this can make exceptionally difficult to see where an error is coming from. Indentation principally makes the structure of the code stand out and easier to be read. This eventually will help in detecting and removing the common programming errors.

JCreator provides user a good formatting and indenting facility where user is associated to format and indent codes automatically in the environment while coding a Java source.

### 6.3.2 Commenting Codes

Comment is not part of the program code and it does not command any program execution. However, comments will slow down the program execution because the compiler has to read and then skip the comment lines each time.

In spite of such shortcoming, comments are still applied as part of common practice in documenting the system's coding. And indeed it is a good and recommended

practice. Commenting will help the author or the reader of the code to understand what and why the coding was written. In addition, this also makes it easier for others especially collaborating programmers to understand the coding.

Comments are usually included before or at the side of each block of the code describing it purposes. In the Java programming language, the // is used as the prefix of a single line comment while /* and */ are used as prefix and postfix of a multiple lines comment.

Figure below shows the formatted, indented and commented code in the JCreator development environment:


**Figure 6.6: Formatted, Indented and Commented Code in JCreator.**

# Chapter 7: System Testing

System testing is a crucial phase in the development of the Quantum Growing Network Simulation system, as it tells whether the coding of the system is successfully implemented, whether the executing system visualizes the growing network accurately and according to the algorithm and whether the code needed to be modified, enhanced, deleted, added or debugged to visualize an accurate or better growing network.

## 7.1 Compiling and Executing

Once the coding of the system is completed, where all the classes designed are fully coded, the Java source code needs to be compiled to see whether there are any bugs or errors in the coding.

If the Java application can be successfully compiled without any error, then the testing phase can proceed to executing the application.

Otherwise, if the Java application is compiled with error(s), the testing phase needs to jump to the debugging phase, before it is recompiled again.

## 7.2 Debugging

Once the application is compiled with error(s), the error message(s) need to be scrutinized to identify where the errors have occurred in the source code. The errors might caused by syntax mistakes, such as the left out of semi-colons, curly braces and other symbols used in the Java programming language. Some errors might also caused by logical errors such as errors in referencing, errors in calling methods, or errors in passing arguments.

The process of debugging is to check on those mistakes and correct them. It is s a process of eliminating errors or bugs from the source code in order for the system to compile successfully.

The compiling-debugging process is an iterative process, which is very common and normal in system programming. Only a successfully compiled piece of source code is able to proceed to the execution process in the testing.

Some commercial Integrated Development Environments come with debugging module which will pace and greatly help the debugging process tremendously.

## 7.3 Accuracy of Execution

After the source code has all the errors eliminated and compiled successfully, it will then be executed or in other words run. The target of the execution process is to allow users to use the system or interact with the system through the system interface.

In the context of this Quantum Growing Network Simulation system, the compiled source is executed to check and verify the correctness and accuracy of the interface, the response of the system in handling user's click on one of the buttons, whether the growing network can be painted (visualized) on the screen (canvas), whether the nodes and links can appear and whether all the components of the network appear in a tidy arrangement or position and according to the algorithm.

If the system does not visualize properly according to the algorithm or response to user's click, the testing phase will go back to the debugging-compiling-executing process until the growing network is able to work and visualizes accurately according to the algorithm.

If the system does visualize the growing network according to the algorithm, but not in an orderly manner, the code needs to be debugged in order to position the nodes, the links and the entire network tidily on the screen (canvas).

Some other modifications other than what were mentioned above, that can be done here would include setting or modifying the colors of the background, the nodes and the links, and also the setting of layout and panel.

## 7.4 Multiplatform Testing

The Java programming language is a multiplatform language. This means the same piece of Java source code can be executed in any Windows, UNIX, Linux, and Apple operating systems as long as the appropriate Java Development Kit is installed.

After the Quantum Growing Network Simulation system is successfully executed in the development environment, which is the Windows environment, and performs accurately, as a Java application, it is then executed in the UNIX platform.

The testing was conducted on the Silicon Graphics' Tezro machine, which is running the Irix operating system (a distribution of UNIX), and the result was the same as how it executes in the Microsoft Windows environment.

This simulation program also utilizes the java.applet packages which will enable the Java application to run on any Java-enabled web browsers (as long as the Java Run Time is available in the operating system which runs the browser). Tests were successfully conducted on the Internet Explorer, Netscape, Opera and Mozilla browsers. Applet of the Quantum Growing Network Simulation runs and performs normally and accurately as how it would perform if it is executed as a Java application.

# Chapter 8: Conclusion and Future Work

Overall I would classify this final year project as a success and insightful. I have set out everything that I planned to achieve and in certain areas, such as accuracy and clear and intuitive visualization, I have actually exceeded my expectations.

Through this project, I have gained a lot of knowledge and exposure regarding the findings, researches and the coming of a new era in human life, and more specifically the computing world. Quantum computing or quantum computation is an interesting, challenging and brain-quenching altogether.

Nevertheless, the outcomes will greatly bring a tremendous revolution to the working and daily lifestyle of mankind. Processing of tasks, whether they are now done automatically (electronically) or manually, will definitely see a huge advancement and leap in speed and accuracy once quantum algorithms can be successfully implemented.

Apart from that, once a quantum computer is realized to put into operation, surely, a bigger picture of multi-simultaneous-parallel processing will be seen, something which will never be achieved using even the most powerful super computer which runs using silicon chips.

Despite of all its great and "mystic" promises and anticipation, there are still a lot of works, endeavors, struggles and barriers to break and go through to bring the theories and hypotheses into real applications, which laymen can see and use.

## 8.1 Problems Encountered

The success of the Quantum Growing Network Simulation system does not come without problems and constraints. Much effort had been placed into understanding,

eradicating, and solving problems encountered through the whole course of the system development.

The first and foremost problem faced in the system development was to understanding the algorithm or the working process of a quantum growing network as proposed by Zizzi. Without doubt, the mathematics to describe how a quantum growing network initiates and grow, is complicated and much time needs to be spent in getting a deep idea and understanding of it.

Besides that, I have no prior knowledge in the Java programming language before the implementation of the system started. I needed to get a deep detailed idea of object oriented programming in Java before coming out with the architectural basis design of the Quantum Growing Network Simulation system.

Objects of the system were to be identified and verified of their applicability and flexibility to be coded. Once the coding of the system started, I encountered a lot of syntax and logical flow errors, which consequently, forced me to spend a huge amount of time in debugging, referring to documentation and even seeking help from experts in forums.

Last but not least, after the final code was successfully compiled and executed, I was taken aback by the non-aligned and jumbled up arrangement of nodes, links and finally the entire network. Works to align and position nodes properly and tidily to better visualize the algorithm of the quantum growing network were carried out.

## 8.2 System Limitations and Future Enhancement

Although the Quantum Growing Network Simulation system project is rated, overall, as a success, it still comes with a number of limitations, and there are still a lot of areas which can be further enhanced.

The system was developed utilizing the Java Abstract Windows Toolkit (AWT) and did not make use of the newer and recommended version of Java Swing extension package for its design and layout. Therefore, the future work on the simulation system can actually use Java Swing instead of Java AWT alone.

Besides that, the system is only suitable to visualize the growing network up to node 9 or node 10. Nodes after node 10 will not be visualized on the canvas due to the limit of the canvas size.

Future enhancement for this constraint, by using Java Swing, is to place a vertical and a horizontal scroll bars on the drawing panel. This will enable the user to scroll left and right, up and down to see the later nodes being visualized.

Besides placing scroll bars, a 3D background can be created on the drawing panel to enable the growing network painted on the panel to rotate on the planar, with mouse events over it.

Last but not least, the completed system is just a visualization or simulation system, it will be a great achievement in human history to be able to put the quantum growing network algorithm into real world application, such as web search and system folder tree search. Again, all these are just brilliant ideas, there are much more effort, research and development to be funded and carried out to see the reality.

# Bibliography and References


1. Zizzi, P.A. The Early Universe as a Quantum Growing Network. Padova, Italy: Univeristy of Padova, Department of Astronomy, 2001. See http://arxiv.org/abs/gr-qc/0103002

2. Nielsen Michael A. and Chuang Isaac L. Quantum Computation and Quantum Information. Cambridge, United Kingdom: Cambridge University Press, 2000.

3. Williams, Collin P. and Clearwater, Scott H. Explorations in Quantum Computing. Santa Clara, California: TELOS, 1998.

4. Sommerville, Ian. Software Engineering 6th Edition. Harlow, United Kingdom: Addison Wesley, 2001.

5. Horstmann, Cay S. and Cornell, Gary. Core Java 2 Volume 1-Fundamentals. Palo Alto, California: Sun Microsystems Press, Prentice Hall, 2003.

6. Sun's Java Resources. http://java.sun.com .

7. ArXiv e-Print Homepage (Cornell University Archive): http://arxiv.org .

8. Los Alamos National Library: http://xxx.lanl.gov .

9. Oxford University Centre for Quantum Computation: http://www.qubit.org .

10. California Institute of Technology (Caltech) Institute for Quantum Information Center: http://www.iqi.caltech.edu .

11. Extreme Programming Model Homepage: http://www.extremeprogramming.org .

12. MathWorks' MATLAB: http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml .

13. MATHEMATICA Homepage: http://documents.wolfram.com .

14. Source code resources: http://www.sourceforge.net .

15. Quantum Computing at the Department of Computing, Imperial College, London: http://www.doc.ic.ac.uk/~ids/quantum_computing.html .

16. Quantum Computer Simulators Hompage: http://www.dcs.ex.ac.uk/~jwallace/simtable.htm .

# APPENDIX A: SOURCE CODE

```
/**************************************************************************
  ------------------------------------------------------------------------
  BEGIN OF FILE "quantumnetwork.java"
  ------------------------------------------------------------------------
  Faculty of Computer Science and Information Technology
  University of Malaya (UM), Kuala Lumpur

  WXES 3182 - Final Year Project
  Simulation of Quantum Growing Network

  Student Name: ONG BOON TEONG
  Matric No: WEK 010223
  NRIC No: 810920-08-5209
  Supervisor: Prof.Ir.Dr.Selvanathan
**************************************************************************/

//Importing and using Java classes*/
import java.awt.*;
import java.applet.*;

//Class Visual NetworkNode
//Contruct, draw, and represent a network node

class VisualNetworkNode extends Rectangle
{
      //Contruct a rectangle to draw a node (boundaries of a node)
      VisualNetworkNode(int n)
      {
            super(0, 0, 40, 40);
      }

      void draw(int i, Graphics g)      //Draw the node
      {
            g.setColor(Color.yellow);  //Set the colour of a node
            g.fillOval(x, y, width, height);

            g.setColor(Color.black);   //Set the colour of a node outline
            g.drawOval(x, y, width, height);

            //Fill a node with an appropriate integer
            int textWidth =
            g.getFontMetrics().stringWidth(String.valueOf(i));
            g.drawString(String.valueOf(i), x+(width/2-textWidth/2), y+25);

      }
}
//End of class VisualNetworkNode

//Class VisualNetwork
//Represent and maintain the whole visualized network

class VisualNetwork
{
      private int root, networkSize;    //Root value and the size of the
network
      private VisualNetwork parent;     //Parent node reference
      private VisualNetwork nextNode;   //Child or next node reference
      private VisualNetworkNode node;   //Graphical representation of the node
      private int positionFactor;       //Position to draw itself
```

```
private int nextPositionFactor;   //Position to draw the next node

VisualNetwork(int r)        //VisualNetwork constuctor
{
      root = r;     //Initialize(construct) all the variable
      networkSize = 1;
      node = new VisualNetworkNode(r);
      positionFactor = 0;
      nextPositionFactor = 0;
}

VisualNetwork newNextNode(int item)//Create new node branch and node
{
      nextNode = new VisualNetwork(item);
      nextNode.parent = this;
      nextNode.node = new VisualNetworkNode(item);
      return nextNode;
}

void setRoot(int r) //Set root value
{
      root = r;
}

int getRoot() //Get root value
{
      return root;
}

void setParent(VisualNetwork wkNetwork) //Set parent node reference
{
      parent = wkNetwork;
}

VisualNetwork getParent()  //Get parent node reference
{
      return parent;
}

void setNextNode(VisualNetwork wkNetwork)//Set next node reference
{
      nextNode = wkNetwork;
}

VisualNetwork getNextNode()       //Get next node reference
{
      return nextNode;
}

void setPositionFactor(int i)     //Set position to draw itself
{
      positionFactor += i;
}

void resetPositionFactor(int i)   //Reset the position
{
      positionFactor = i;
}

int getPositionFactor()    //Get position
{
      return positionFactor;
}
```

```
	//Set position for next node to draw itself
	void setNextPositionFactor(int i)
	{
		nextPositionFactor += i;
	}

	void resetNextPositionFactor(int i)//Reset next node position
	{
		nextPositionFactor = i;
	}

	int getNextPositionFactor()		//Get next node position
	{
		return nextPositionFactor;
	}

	//Check whether passing argument = node value
	boolean notEqualItem(int item)
	{
		return root != item? true : false;
	}

	void insert(int item)		//Inserting a node
	{
		VisualNetwork wkNetwork = this;
		int oldNetworkSize = networkSize;

	//If the network already exists and current node is not the same as its
root
		while(wkNetwork!=null &&  wkNetwork.notEqualItem(item))
		{
			//If current node value is larger than its root value
			if(item > wkNetwork.getRoot())
			{
				//If next node is not avaiable yet
				if(wkNetwork.getNextNode()==null)
				{
					//Construct the next node
					wkNetwork.newNextNode(item);
					//Increment the network size
					networkSize++;
				}
				//Move network reference to the next node
				wkNetwork = wkNetwork.getNextNode();
			}

			//If next node added successfully
			if(networkSize > oldNetworkSize)
			{
				//Position the next node accordingly
				nodesInsertPositioning(wkNetwork);
			}
		}
	}

	void drawNode(int x, int y, Graphics g) //Graphically draw the node
	{
		node.move(x, y);//Move the node to the correct position
		node.draw(root, g);//And call class VisualNetworkNode to draw
itself as a node

		//If the node has parent, meaning if it is not the -1 node
```

```
            if(this.getParent()!=null)
            {
                  if(this.getParent().getRoot()!=-1)//Referring to node 0
                  {
                        //Draw a green line representing a virtual state
                        g.setColor(Color.green);
                        //Line connecting current node to the previous node
                        g.drawLine(x, y+20, x-positionFactor+40, y+50);

                  }

                  //Draw a green line representing a virtual state
                  g.setColor(Color.green);
                  //Line connecting current node to node -1
                  g.drawLine(x+20, y, 285, 95);


                  //Distance between free links of a node
                  int linksPosition = 100;

            //setting the correct amount of free links available for each
different node
                  for(int freeLinksLoop = 1; freeLinksLoop <=
2*this.getRoot()+1; freeLinksLoop++)
                  {
                        //Draw a blue line representing a free link
                        g.setColor(Color.blue);
                        //Blue line branching out from a current node
                        g.drawLine(x+20, y+40, x+20-linksPosition, y+120);

                        linksPosition -= 20;//Position the next free link
                  }
            }
      }

      //Positioning related nodes after a node is inserted
      void nodesInsertPositioning(VisualNetwork nt)
      {
            VisualNetwork wkNetwork = nt;     //To check all nodes are
positioned correctly


      wkNetwork.setPositionFactor(80+wkNetwork.getParent().getNextPositionFact
or());
            while(wkNetwork.getParent()!=null)
            {
                  if(wkNetwork.getRoot() < wkNetwork.getParent().getRoot())
                  {
                        if(wkNetwork.getParent().getNextNode()!=null)
                        {

      wkNetwork.getParent().getNextNode().setPositionFactor(80);
                        }
                        wkNetwork.getParent().setNextPositionFactor(80);
                  }
                  wkNetwork = wkNetwork.getParent();
            }
      }

}
//End of class VisualNetwork

//Class GraphCanvas
```

```
//Construct a canvas for the nwtwork to be painted

class GraphCanvas extends Canvas
{
	VisualNetwork mainNetwork, currentNetwork;
	private int count;

	//Java 2D paint function to paint nodes and network on the canvas
	public void paint(Graphics g)
	{
		Rectangle r = bounds();
		VisualNetwork wkNetwork = currentNetwork;

		if(currentNetwork==null)
		{
			return;
		}

	 if(wkNetwork.getParent()==null) //Start of the network
	 {
		wkNetwork.drawNode(260, 60, g);   //Draw the first node (node -1)
	 }

	 if(wkNetwork.getNextNode()!=null)      //If next node is inserted
	 {
		//Move to the next node and draw itself
		paintNode(wkNetwork.getNextNode(), 20, 440, g);
	 }
    }

	//Position and draw upcoming nodes
	void paintNode(VisualNetwork nt, int x, int y, Graphics g)
	{
		//Set the position and draw itself
		nt.drawNode(x+nt.getPositionFactor()-20, y, g);

		//Continue the positioniong and drawing as long as next node is
available
		if(nt.getNextNode()!=null)
		{
			paintNode((VisualNetwork)nt.getNextNode(),
x+nt.getPositionFactor(), y-30, g);
		}
	}

	public void insert(int node)      //Inserting method
	{
		if(mainNetwork==null)      //Inserting a new network, begin the
network
		{
			mainNetwork = new VisualNetwork(node);
			currentNetwork = mainNetwork;
			count=node;
		}
		else   //If a network already exists
		{
			mainNetwork.insert(node);
			count=node;
		}

		repaint();
	}
```

```
	public void clear() //Clear the network and its nodes
	{
		mainNetwork = currentNetwork = null;

		repaint();
	}
}
//End of class GraphCanvas

//Class GraphControls
//Construct controls or buttons of the program and managed the action performed
on each button

class GraphControls extends Panel
{
	GraphCanvas canvas;

	public GraphControls(GraphCanvas canvas)//Construct 3 buttons
	{
		this.canvas = canvas;
		add(new Button("Visualize"));	//Visualize button
		add(new Button("Next"));	//Next button
		add(new Button("Clear"));	//Clear button

		NodeValue = 0;
	}

	public boolean action(Event ev, Object arg)//Manage action on each
button
	{

		if(ev.target instanceof Button)
		{
			String label = (String)arg;

			if(label.equals("Visualize"))//Visualize button is clicked
			{
				canvas.insert(getRootValue());
			}

			else if(label.equals("Next"))
			{
				canvas.insert(getNodeValue());//Next button is
clicked
				setNodeValue();
			}

			else if(label.equals("Clear"))	//Clear button is clicked
			{
				canvas.clear();
				resetNextNodeValue();
				setNodeValue();
			}

			return true;
		}
		return false;
	}

	public static int getRootValue()  //Get root value
	{
		return RootValue;
	}
```

```
        public int getNodeValue()  //Get node value
        {
                return NodeValue;
        }

        public void setNodeValue() //Set node value
        {
                NodeValue = nextNodeValue;
                nextNodeValue++;
        }

        void resetNextNodeValue()  //Reset node value
        {
                nextNodeValue = 0;
        }

        public int getNextNodeValue()     //Get next node value
        {
                return nextNodeValue;
        }

        private static int RootValue = -1;//Initialize all the variables
        private int NodeValue;
        private int nextNodeValue = 0;

}
//End of class GraphControls

//Class quantumnetwork
//Construct an application window or an applet and its layout

public class quantumnetwork extends Applet
{
        GraphControls controls;

        //Initiate the applet or the application window
        public void init()
        {
                setLayout(new BorderLayout());
                //Set the application background color
                setBackground(Color.white);
                GraphCanvas c= new GraphCanvas();//Construct canvas
                //Place canvas that is to be painted, on the center of the
windows
                add("Center", c);
                //Place buttons on the bottom of the windows
                add("South", controls= new GraphControls(c));
        }

        public void start() //Start the application
        {
                controls.enable();
        }

        public void stop()  //Stop the application
        {
                controls.disable();
        }

        public static void main(String args[])  //Main function
        {
                //Construct an application frame
```

```
            Frame f= new Frame("Quantum Growing Network");
            //Reference to the class
            quantumnetwork QuantumNetworkApp= new quantumnetwork();

            QuantumNetworkApp.init();  //Begin the application
            QuantumNetworkApp.start();

            f.add("Center", QuantumNetworkApp);
            f.resize(1000,700);//Set the size of application window or applet
            f.show();
        }
}
//End of class quantumnetwork

/**************************************************************************
  ------------------------------------------------------------------------
  END OF FILE "quantumnetwork.java"
  ------------------------------------------------------------------------
**************************************************************************/
```

# APPENDIX B: USER MANUAL

## Preparation

Before you can begin using the Quantum Growing Network Simulation system, there are a few preparations to make. First and foremost, you need to make sure that Java Software Development Kit (JDK) is installed in your operating system.

At the time of the development of this system, the version of Java 2 Standard Edition (J2SE) Development Kit that was used is 1.4.2_02. Newer versions of the development kit may have been developed and are usually backward compatible with older versions. In other words, the Quantum Growing Network Simulation system should be able to run in newer JDK versions although it is developed using the 1.4.2_02 version.

To get the J2SE Development Kit installation file, you can go to Sun's Java website at http://java.sun.com and look for downloads link for J2SE or you can go straight to the download site at http://java.sun.com/j2se/1.4.2/download.html (still available at time of development). Besides the installation file, you can also download the Java documentation and the free NetBeans Java Integrated Development Environment (IDE).

Make sure you download the correct JDK according to the operating system you are using. Otherwise, the development kit would not set up properly and yield installation errors.

Although a Java source code file can be created using any text editor, for example the Windows Notepad, as long as it is saved with a filename bearing the extension .java, it is recommended to create it using an IDE; be it a commercial or non-

commercial one. This is because an IDE provides facilities such as code formatting and indenting, code syntax completion, colors for different syntax of the code, and even a debugger which will definitely pace up the development and debugging phases. In addition it also provides a better visual to view and look at the coding and classes created on the project tree pane of the IDE, and greatly help in compiling and executing the code without having to go through complex and much troublesome command line compiling and executing.

This simulation system was developed under the JCreator Light Edition (LE) v2.50 IDE produced by Xinox Software. The Light Edition is a freeware available for downloads at http://www.jcreator.com . Other non-commercial IDEs such as NetBeans and Eclipse and commercial ones like Borland JBuilder can also be used.

## Installation of Tools

After having downloaded all the appropriate required installation files, the JDK and JCreator, you can begin installing them into your operating system. The installation files might also be bundled and zipped together with other files such as the documentation files and the installation guide files.

Extract the files into a destination folder. Open the folder and read the guide and documents files available before you start installing the particular tool by double clicking on the executable installation file.

Follow the on-screen instructions, read the agreements and agree to them to proceed with the next steps of installation until all the installations are completed and finished.

It is recommended to install the tools to the default folders as would have appeared on the screen during the installation process. Changing the destination folders might cause some problems with path setting of the Java environment later on.

After you have finished with the installation of the tools, run the JCreator IDE for the first time. When you run it for the first time, the program will ask whether you want to associate any files with JCreator. Choose to associate Java file to JCreator, so by default any Java file will open in JCreator whenever an user clicks to open it.

## Compiling and Executing

Once you have installed and setup all the tools and IDE to run the Quantum Growing Network Simulation system, you can actually start using it.

Nevertheless, before using, you need to get the Java file created, then compiled and executed to visualize the quantum growing network. To do so, you will have to open the Java source file (named quantumnetwork.java) in JCreator. Once the file is opened in JCreator, the source code of the system will appear on the right window pane while the tree of classes and methods on the left window pane as shown below:


**Figure B1: The JCreator Development Environment.**

If you do not have the softcopy of the Java source file (quantumnetwork.java), you can actually click open the JCreator IDE, then click on the "File" and "New" tabs subsequently on the top left side of the IDE windows. Choose "Files" tab and then click on the Java file icon, key in the correct Java filename (quantumnetwork, remember it is case sensitive) and save it into a location and click "OK". On the right pane of the windows now you can type the source code (refer APPENDIX A). Save the file after you have completed typing.

Figure below shows how the IDE window looks like when you want to create a new Java source file:

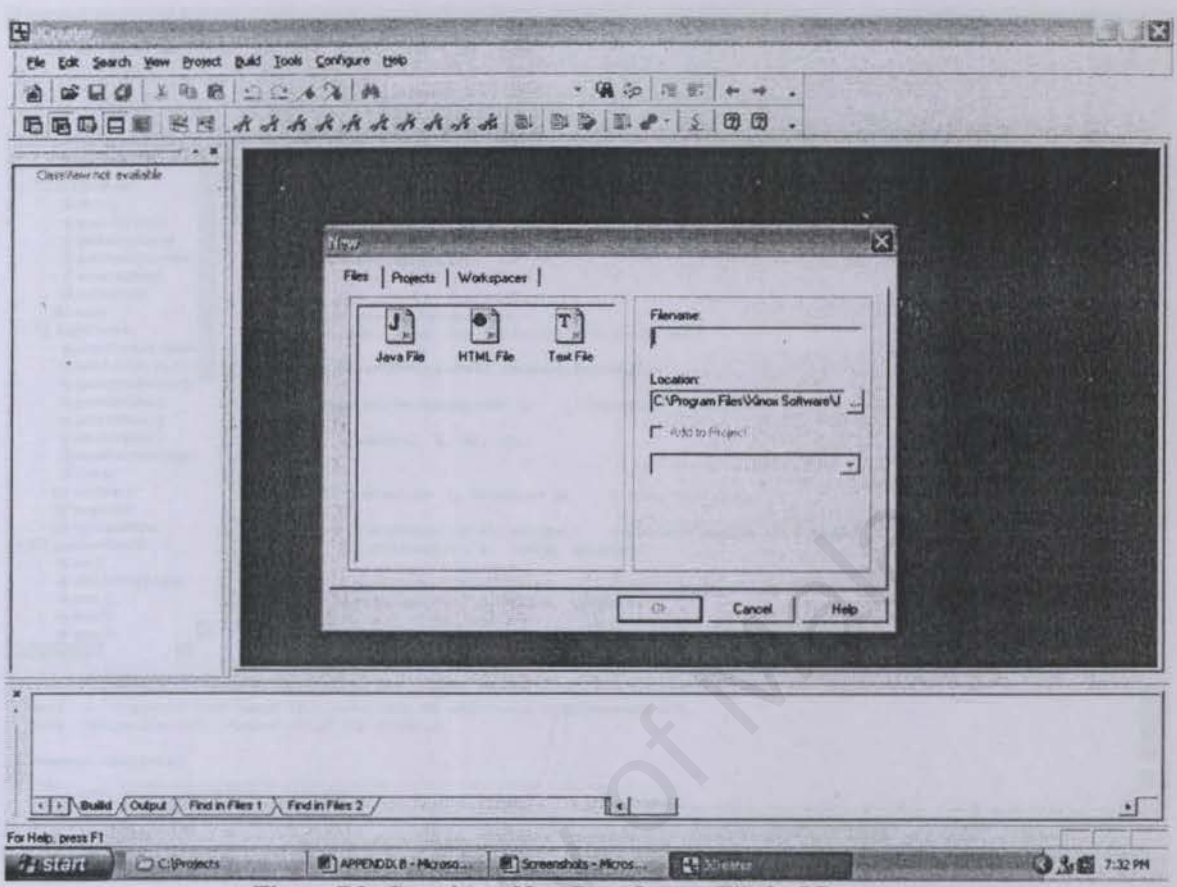


**Figure B2: Creating a New Java Source File in JCreator.**

When the source is ready and is saved, you can start to compile it. Click on the "Build" tab on the top of the window and then "Compile File". Wait a while until the compilation message appears on the bottom of the IDE window as shown at Figure below:


**Figure B3: Compilation in JCreator.**

After completing the compilation process, then you can proceed to the execution process. In order to execute and run the simulation program, click on the "Build" tab one more time and choose "Execute File". A command prompt window will appear together with the simulation program window. The starting of the Quantum Growing Network simulation system is as below:

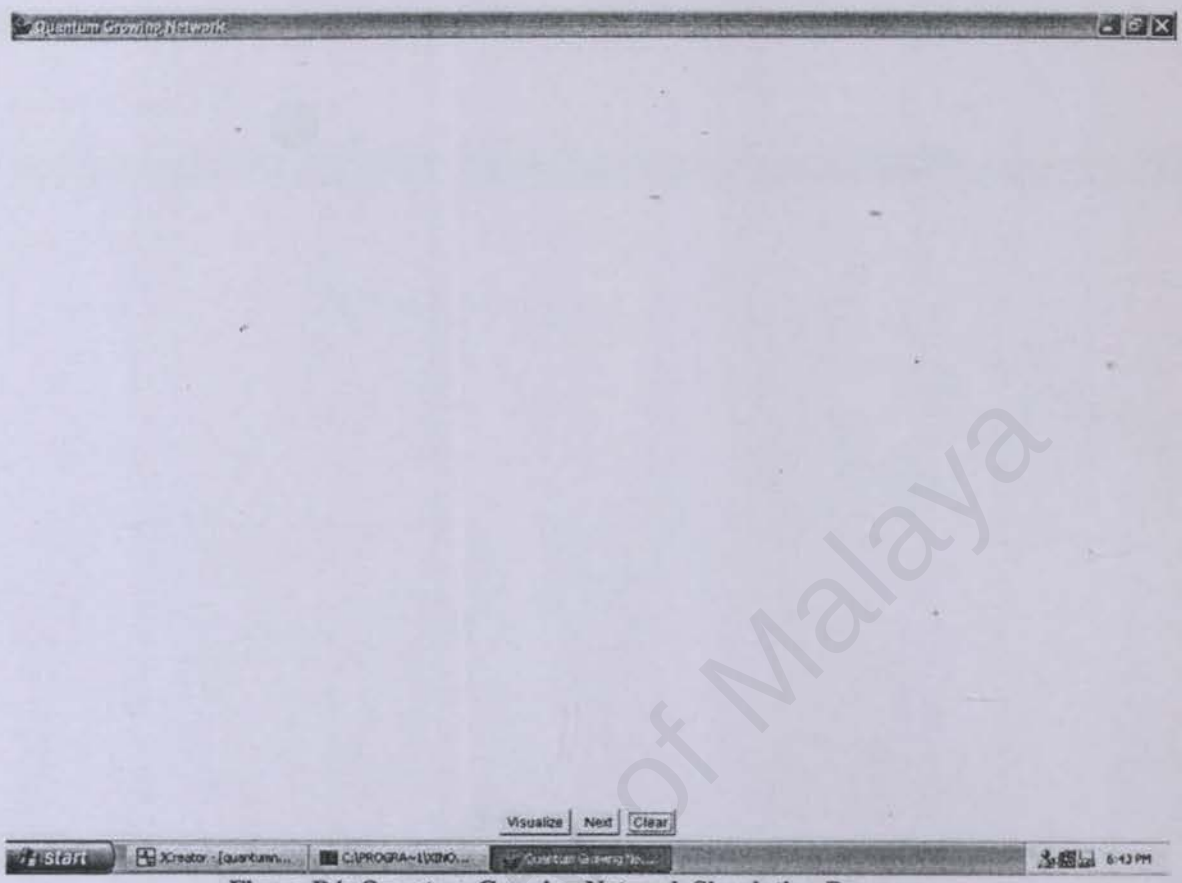


**Figure B4: Quantum Growing Network Simulation Program.**

The window of the simulation system contains 3 buttons, the "Visualize", "Next", and "Clear" buttons. The background color of the system is set to white as shown above.

## Using the Simulation Program

After the simulation program is executed, you may wonder how to visualize and understand the quantum growing network. Click on the "Visualize" button to start visualization. Node -1 will appear as shown below:


**Figure B5: The Simulation Program with the First Node.**

The network will grow to the next node when the "Next" button is clicked. Continue on clicking the "Next" button until the network grows to node 10 (the simulation program is only suitable to view up to node 10). The "Clear" button is to clear the network off from the canvas or screen to begin visualization again.

In the visualization, you will notice green lines connecting a particular node to the previous node and node -1. These green lines or links are actually denoting the virtual stated as described by Zizzi.

Meanwhile, there are blue lines or links branching out from a particular node but the number of lines from different nodes is different. These denote free links as mentioned by Zizzi. For node n, there are 2n+1 free links. Close the command prompt window to disable and shut down the program.

Below is a figure showing how it looks when the quantum growing network grows to node 10:

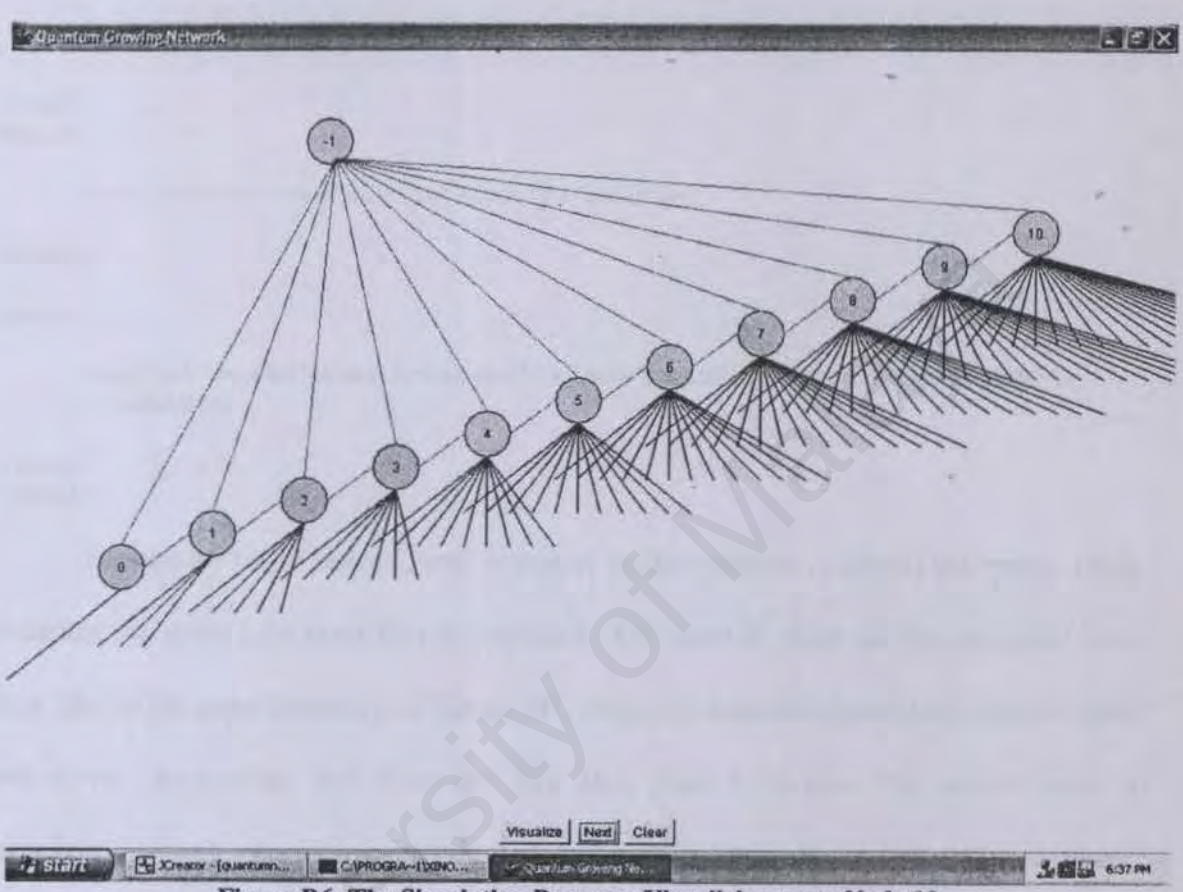


**Figure B6: The Simulation Program Visualizing up to Node 10.**

## Running as an Applet on Web Browsers

There is a need sometimes to be able to run the simulation online using a web browser. In order to run the Quantum Growing Network Simulation program as an applet, the web browser must be Java-enabled and the operating system must have the Java Runtime Environment (JRE) installed (usually the JRE comes bundled together with the Standard JDK).

Firstly, to execute the applet, you need to compile the source file (Java source code) into class files. Then, you will have to create a web page where the applet will be

loaded. Basic HTML knowledge is required to do so. To embed the applet into the web page, the <applet> tag is used.

Below is a source HTML file example of how to embed the applet:

```
<html>
<head>

      <title>Quantum Growing Network</title>

</head>

<body>

      <applet code="quantumnetwork.class" width="950" height="580">
      </applet>

</body>
</html>
```

The applet tag is bolded, and is placed in the <body> of an HTML page. Code identifies the main Java class that is compiled. You need to place all the compiled Java class files in the same directory as the HTML page you created in your local host or your web server. Remember that there are five Java class files once the source code is compiled, namely quantumnetwork.class, GraphControls.class, GraphCanvas.class, VisualNetwork.class, and VisualNetworkNode.class.

An online applet of the Quantum Growing Network is now placed and loaded into a website, which can be accessed and started at http://www.geocities.com/bt3on9/thesis.htm .

The 2 figures below show how the applet looks like when it is started in Internet Explorer and Netscape Navigator respectively:

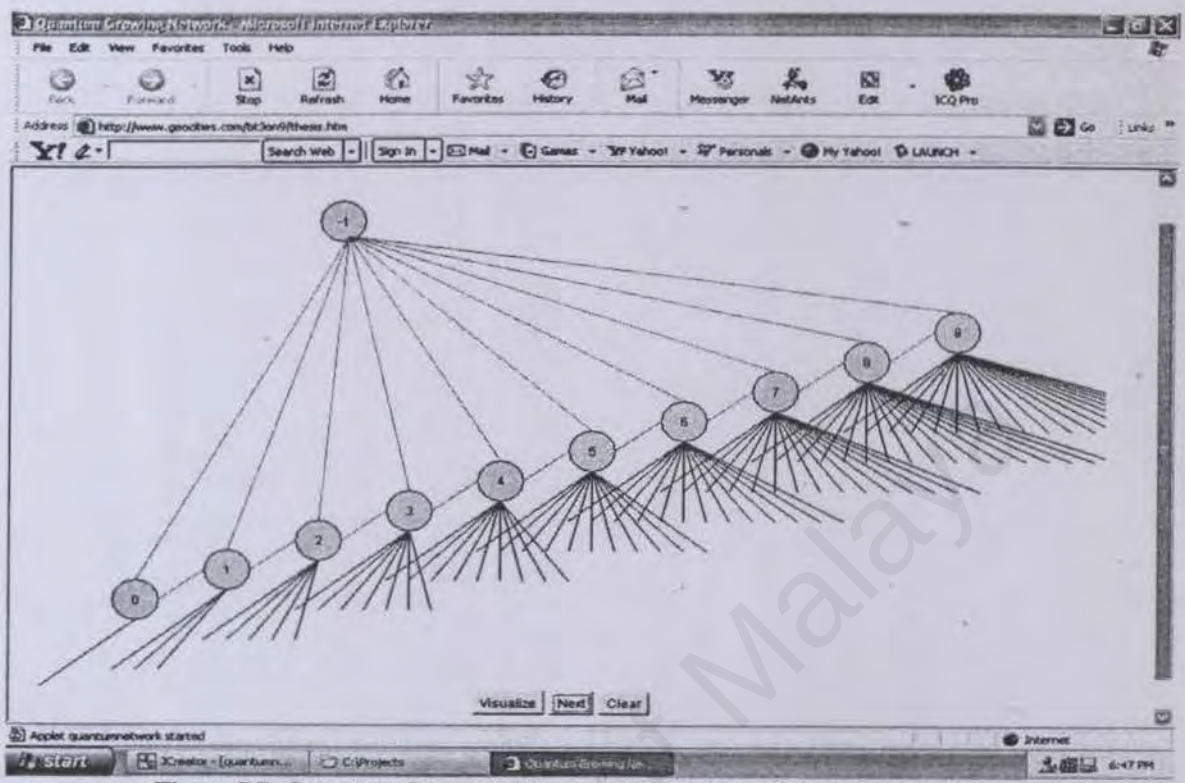


**Figure B7: Quantum Growing Network Applet Runs in Internet Explorer.**

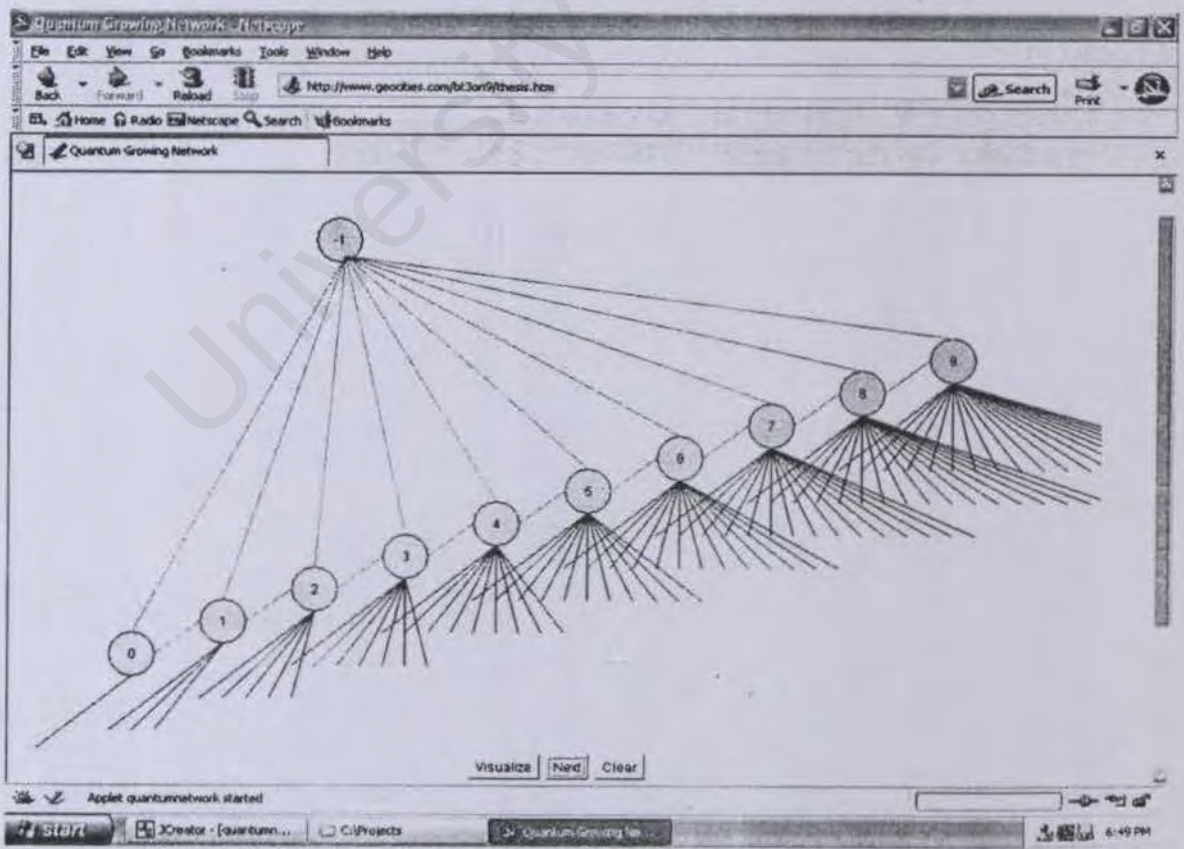


**Figure B8: Quantum Growing Network Applet Runs in Netscape Navigator.**