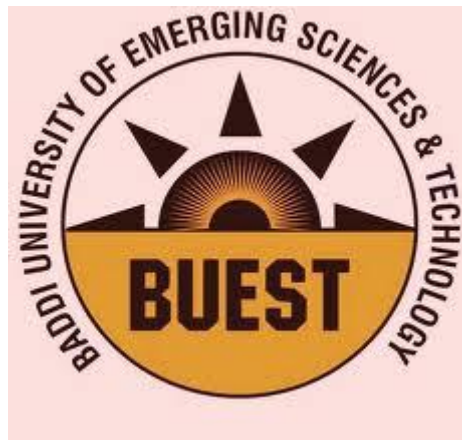


PROJECT REPORT
ON
“iBot Bulk Management Tool v2.0”
A Major Project submitted for the partial fulfillment of the degree
Of
Bachelor of Engineering in Computer Engineering (Session
2011-2012)



Project Guide:
Er .Roshi Saxena

Project Coordinator:
Dr.Ajay Goel

Submitted by:
Nishant Sharma (12208)
Ankit Sharma (10708)
Ankesh Bhatia (10608)
Kapil Jhanji (11708)

CERTIFICATE

This is to certify that the project entitled “**iBot Bulk Management Tool v2.0**” submitted by Nishant Sharma, Ankit Sharma, Kapil Jhanji and Ankesh Bhatia have been developed out under my supervision and I am satisfied with their work.

Head of Department

(Dr. Ajay Goel)

Project Guide

(Er. Roshi Saxena)

ACKNOWLEDGEMENT

We will like to take this opportunity to express our deepest gratitude to those who have generously helped us in providing the valuable knowledge and expertise during development of our project.

We are very thankful to our Head of Department **Dr. Ajay Goel** to provide us such opportunity to work on this project and our project guide **Er. Roshi Saxena** and the entire staff of Computer Science Department who had provided us invaluable assistance in completing this project directly or indirectly.

And finally, we would like to thanks each and every person who contributes in any ways in our project.

Thanks to All

Nishant Sharma

Ankit Sharma

Kapil Jhanji

Ankesh Bhatia

PREFACE

Success is what everyone wants to achieve, and all the successful people do not do different things, they do things differently. Computer Technology shows the way to do things differently in a more better, efficient and effective way.

We are living in the Info sphere, where without information not only success but also the survival is at stake. Now the world is moving towards the new era, the era of computer technology where every individual is supposed to be well versed with the information technology and its utility.

Automation has been the major goal of modern world. Man has always tried to make things such as that his work becomes faster and efficient for which computer has been great achievement towards his goal; computer has been contributed a lot in automation.

The way these machines have manifested in our lives echoes the same effect as that of the discovery of the wheel in the evolution of mankind. Computer can no longer be termed as merely an invention- but it is a revolution; a revolution that is changing the way of our work-study, travel and business. The way we live today with ease and technology is an evidence of this revolution.

Contents

S. No.	Topic	Page
1.	Certificate	2
2.	Acknowledgement	3
3.	Preface	4
4.	Chapter 1: Introduction	6
5.	Chapter 2: System Requirement and Problem definition	7
6.	Chapter 3: System Analysis and Design	9
	3.1 Identification of needs	
	3.2 Development model followed (SDLC)	
	3.3 Software Requirement Specification (SRS)	
	3.4 Flow Chart	
	3.5 Proposed System Architectures	
7.	Chapter 4: Implementation and Testing	29
	4.1 Working Environment	
	4.2 Test Results	
8.	Chapter 5: Results and Discussions	53
9.	Chapter 6: Conclusion and Future Scope	59
	6.1 Conclusion	
	6.2 Future scope of the project	
10.	Appendix A: Project source code	60
11.	Appendix B: Published article based on iBot in LFY magazine	
12.	Bibliography/References	70

Chapter 1: Introduction

Introduction

iBot is a Linux based client server application in which we can take control of all the clients and direct them to perform the intended task from the single machine. The coding of the project is in c language.

Overview

The application is a linux based application that connects to a server through a logical port. Server can only send instructions such as software installation, shutting down the systems simultaneously etc. so that without going to every machine and instructing it separately we can instruct them sitting on a single machine. Clients can only listen to the instructions of the server but not send any query or instruction to it.

Objective/ Aim

iBot is a logical communication network among all computers on the network that enables administrator to control, configure, install programs on all computers in few minutes by executing commands from one computer or mobile phone.

Motivation

As botnet now a days are very famous technologies in World Wide Web. Hackers use them to take down any site by performing DDoS, steal information from users and to spread adware. While we were studying this technology we analyze that this brilliant idea can be used in a productive way instead of harming others.

Secondly we observed various scenarios where using software can reduce the human effort by more than 99% and increase productivity.

iBot v2.0 :

The iBot v1.0 lacks in some features that's why the v2.0 is developed with following features in mind :

- Controlling the bot cluster from mobile (any java compatible or android device)
- Extended support for more commands

Chapter 2: System Requirement and Problem definition

System Requirement:

The requirements of the server and the clients are different, so they are mentioned separately below.

Processor: Intel Pentium III or higher

Processor Architecture: x86 or other

Physical Memory: 128 MB or higher

OS: Ubuntu Linux 32-bit 6 or higher

Problem definition:

This software basically solves the problem of accessing the machine separately and thus you can access hundreds of systems on a network and instruct them to perform the operations that you want them to perform. Two of the problem situations are given below that will be solved using this software tool.

Situation 1:

A network administrator manages a network of computers (ranging from 20 to 150 computers) in an organization and is responsible for installing new software, shutting computer down at the end of day and making other settings.

Now let us suppose that the network computers are not on a domain (Windows network) so if any change in settings or installation of new software is needed then he has to go on every computer one by one and make changes on each system and if his network is of 100 computers then it will consume minimum of 2-3 days. This will overburden the person as well as decrease the productivity by keeping user waiting for software installation.

Situation 2:

Network administrator is on leave and all the systems need a crucial change in settings immediately.

Probable solution:

Software should be there on every machine of network that listen to the network administrator's commands (even from remote place) and execute it on each machine so all the work can be done in few seconds.

Thirdly, according to our research till date there is not a single solution like this. That's why we choose this project.

Chapter 3: System Analysis and Design

3.1 Identification of needs

The need of such system is defined in problem definition part.

3.2 Development model followed (SDLC)

What is a SDLC and why do we need that?

System - an organized collection of independent tasks and processes that is designed to work together in order to accomplish specific objectives. The processes and tasks typically receive input(s) from and provide output(s) to other processes and tasks and even other systems. The tasks and processes may or may not be supported by automation

SDLC refers to a methodology for developing systems. It provides a consistent framework of tasks and deliverables needed to develop systems. The SDLC methodology may be condensed to include only those activities appropriate for a particular project, whether the system is automated or manual, whether it is a new system, or an enhancement to existing systems. The SDLC methodology tracks a project from an idea developed by the user, through a feasibility study, systems analysis and design, programming, pilot testing, implementation, and post-implementation analysis. Documentation developed during the project development is used in the future when the system is reassessed for its continuation, modification, or deletion.

SDLC Phases

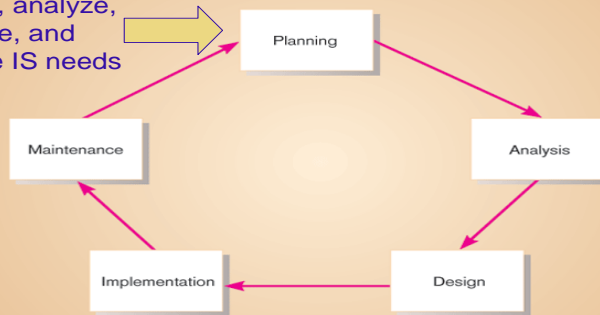
Phases in SDLC are Planning, Analysis, Design, Implementation, and Maintenance/Sustainment/Staging

- **Project planning, feasibility study:** Establishes a high-level view of the intended project and determines its goals.

SDLC Planning Phase

Figure 1-3 The systems development life cycle

Identify, analyze, prioritize, and arrange IS needs



1-11

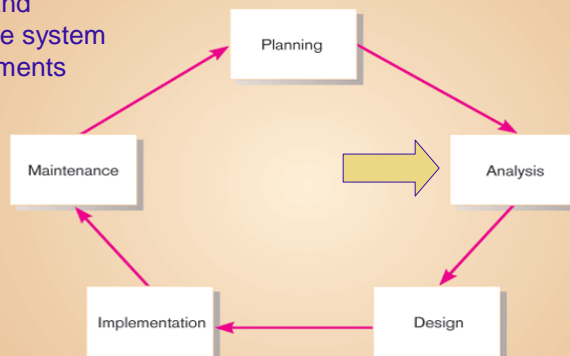
© 2005 by Prentice Hall

- **Systems analysis, requirements definition:** Refines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.

SDLC Analysis Phase

Figure 1-3 The systems development life cycle

Study and structure system requirements

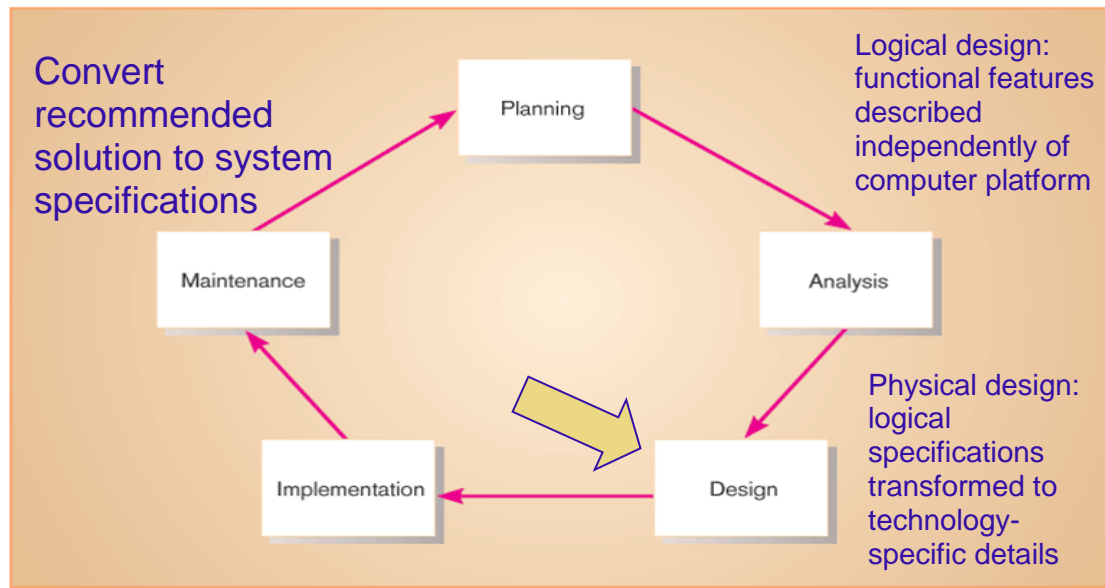


1-12

© 2005 by Prentice Hall

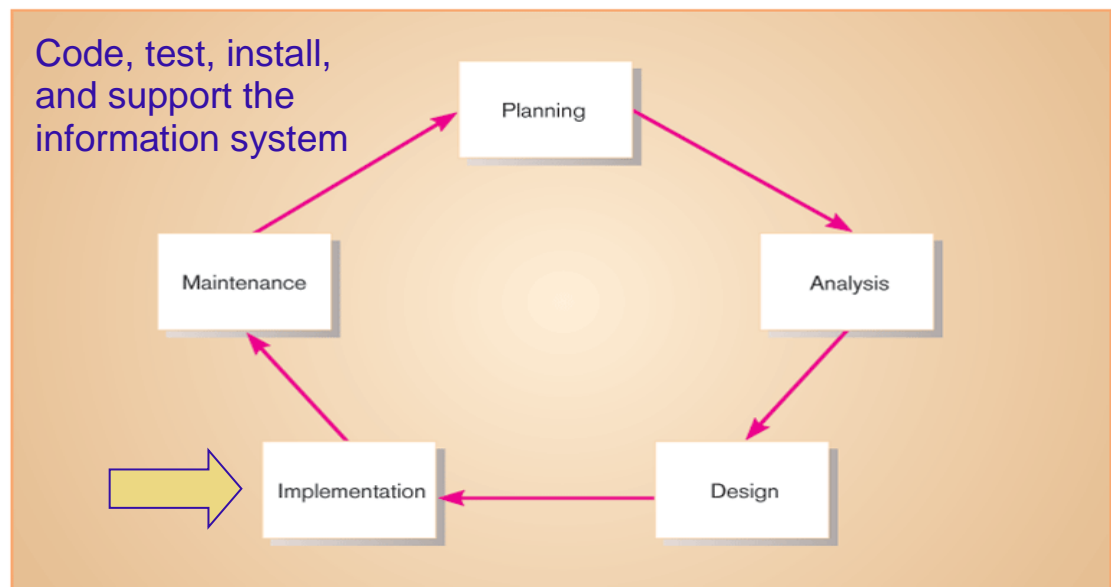
- **Systems design:** Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudo code and other documentation.

Figure 1-3 The systems development life cycle



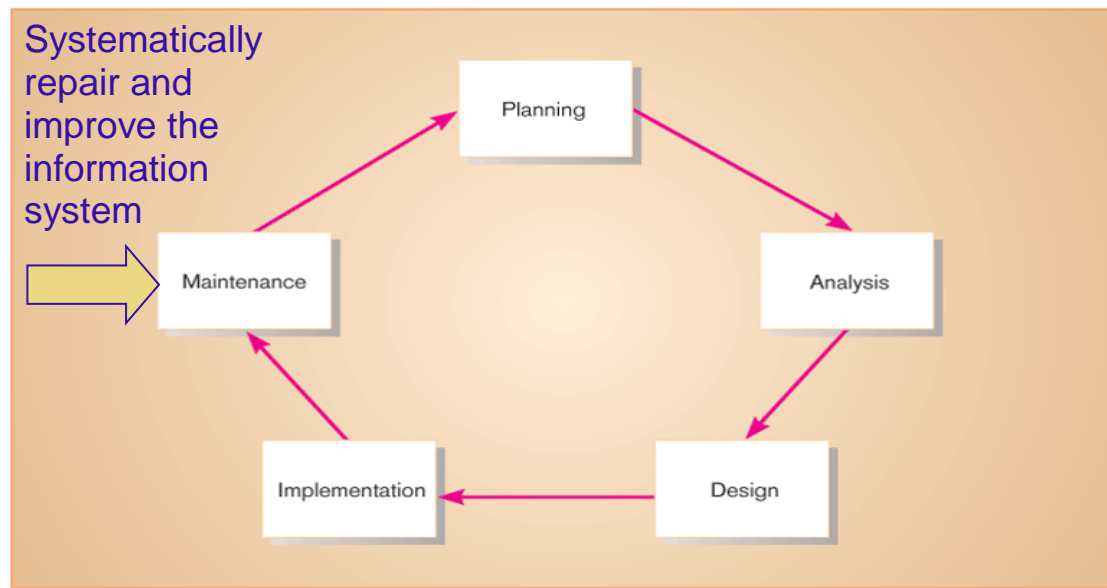
- **Implementation (Development):** The real code is written here.

Figure 1-3 The systems development life cycle



- **Integration and testing:** Brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability.
- **Acceptance, installation, deployment:** The final stage of initial development, where the software is put into production and runs actual business.
- **Maintenance:** What happens during the rest of the software's life: changes, correction, additions, moves to a different computing platform and more.

Figure 1-3 The systems development life cycle



Products and Deliverables

Table 1-2 Products of SDLC Phases

<i>Phase</i>	<i>Products, Outputs, or Deliverables</i>
Planning	Priorities for systems and projects; an architecture for data, networks, and selection hardware, and IS management are the result of associated systems; Detailed steps, or work plan, for project; Specification of system scope and planning and high-level system requirements or features; Assignment of team members and other resources; System justification or business case
Analysis	Description of current system and where problems or opportunities are with a general recommendation on how to fix, enhance, or replace current system; Explanation of alternative systems and justification for chosen alternative
Design	Functional, detailed specifications of all system elements (data, processes, inputs, and outputs); Technical, detailed specifications of all system elements (programs, files, network, system software, etc.); Acquisition plan for new technology
Implementation	Code, documentation, training procedures, and support capabilities
Maintenance	New versions or releases of software with associated updates to documentation, training, and support

Types of SDLC models

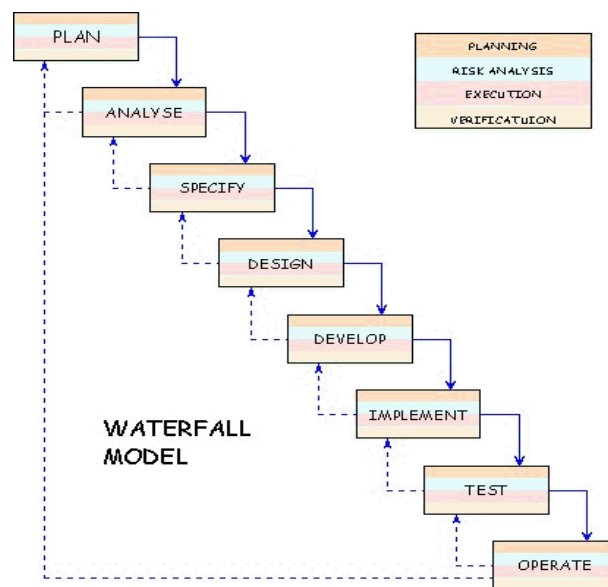
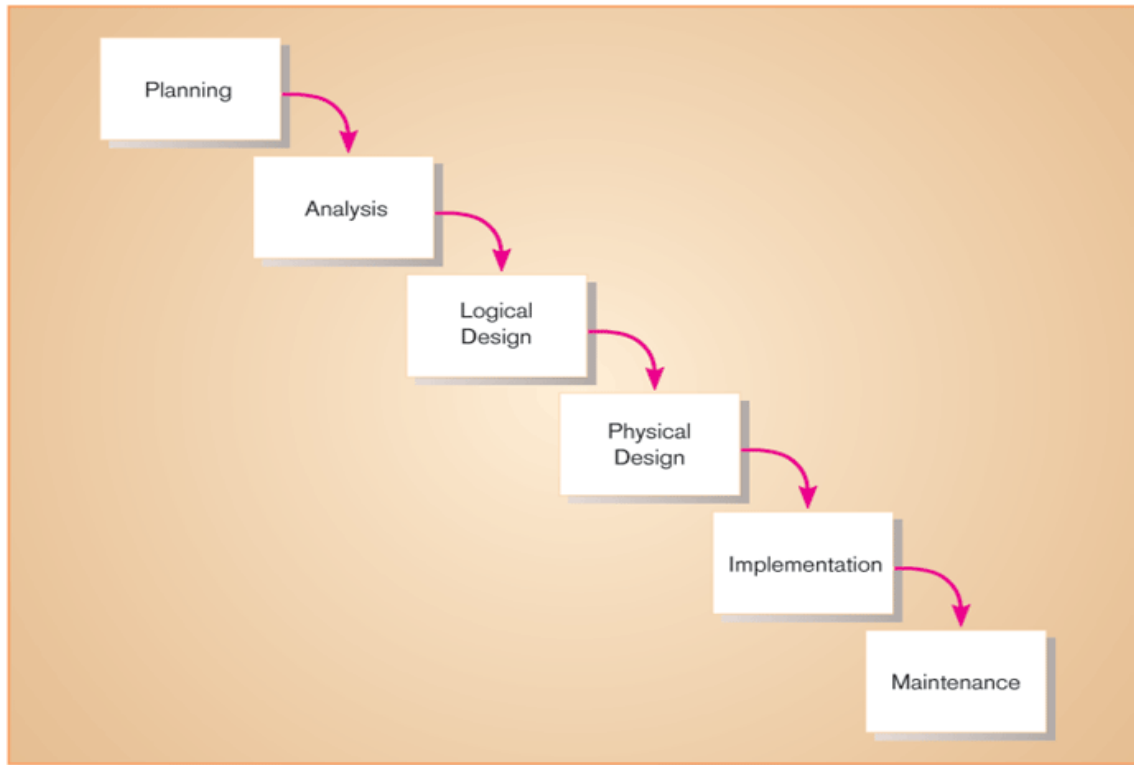
Once upon a time, software development consisted of a programmer writing code to solve a problem or automate a procedure. Nowadays, systems are so big and complex that teams of architects, analysts, programmers, testers and users must work together to create the millions of lines of custom-written code that drive our enterprises.

The oldest of these, and the best known, is the waterfall: a sequence of stages in which the output of each stage becomes the input for the next. These stages can be characterized and divided up in different ways, including the following:

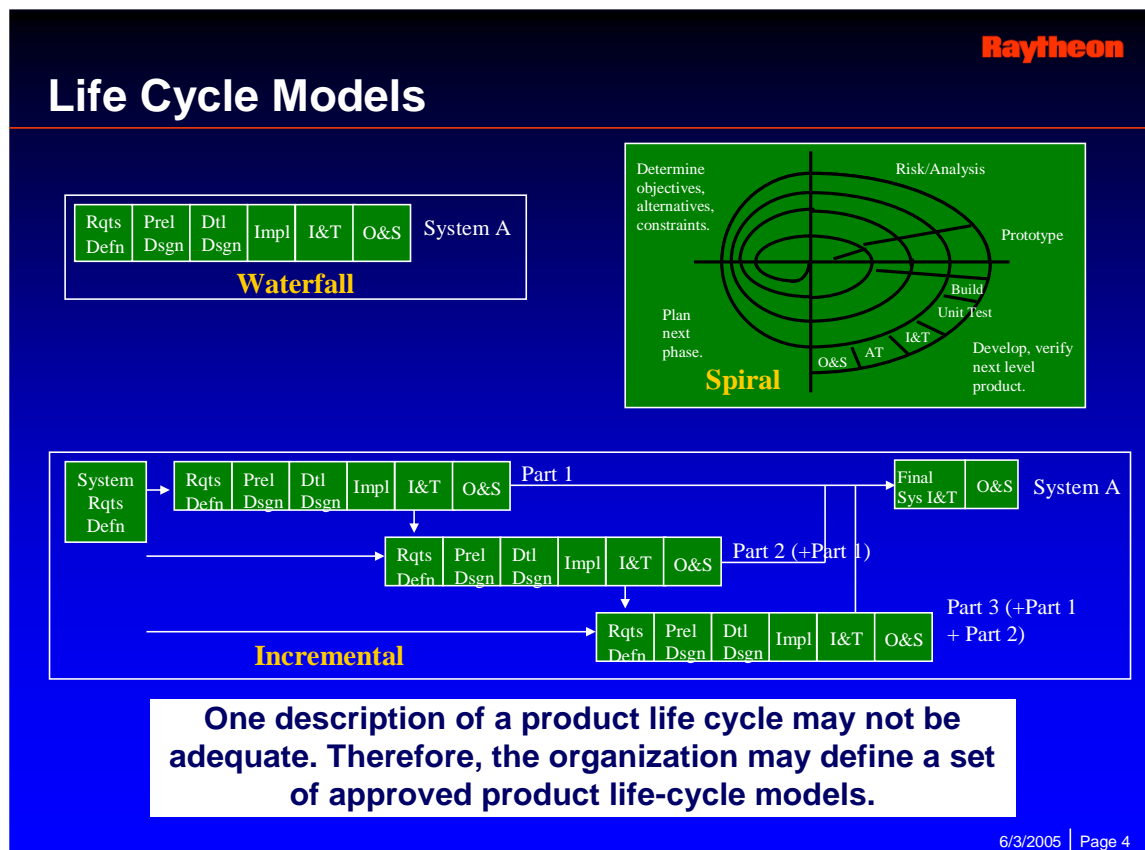
But It Doesn't Work!

The waterfall model is well understood, but it's not as useful as it once was. The problem is that the waterfall model assumes that the only role for users is in specifying requirements, and that all requirements can be specified in advance. Unfortunately, requirements grow and change throughout the process and beyond, calling for considerable feedback and iterative consultation. Thus many other SDLC models have been developed.

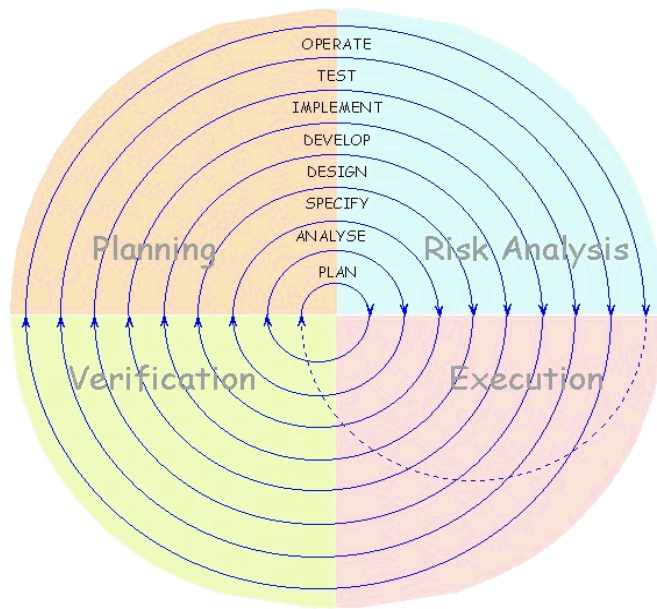
Figure 1-10 A traditional waterfall SDLC



To manage this, a number of system development life cycle (SDLC) models have been created: **waterfall**, **spiral**, **rapid prototyping**, **RUP (Rational Unified Process)** and **incremental** etc.



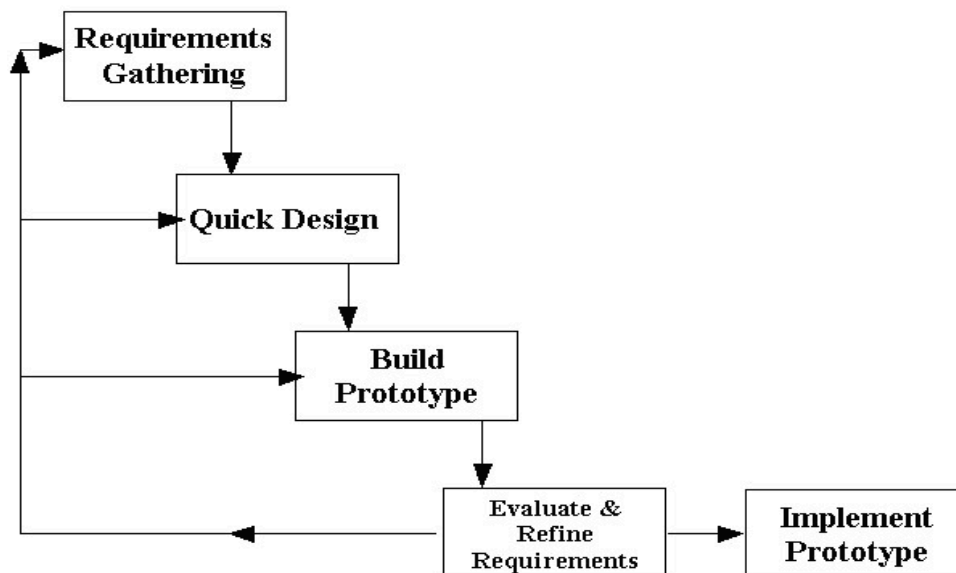
Spiral model - The spiral model emphasizes the need to go back and reiterate earlier stages a number of times as the project progresses. It's actually a series of short waterfall cycles, each producing an early prototype representing a part of the entire



project.

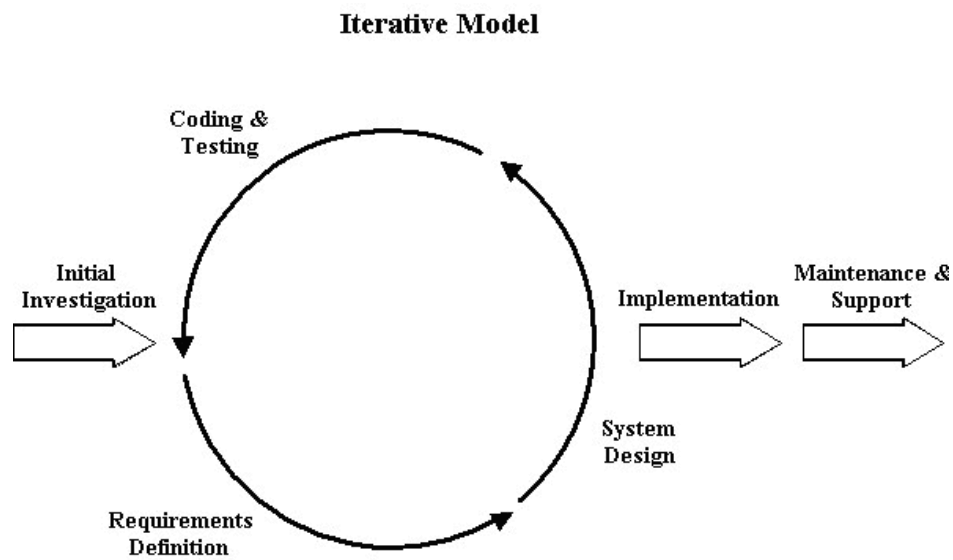
This approach helps demonstrate a proof of concept early in the cycle, and it more accurately reflects the disorderly, even chaotic evolution of technology.

Rapid Prototyping - In the rapid prototyping (sometimes called rapid application development) model, initial emphasis is on creating a prototype that looks and acts like the desired product in order to test its usefulness. The prototype is an essential part of the requirements determination phase, and may be created using tools different from those used for the final product. Once the prototype is approved, it is discarded and the "real" software is written.



Incremental - The incremental model divides the product into builds, where sections of the project are created and tested separately. This approach will likely find errors in user requirements quickly, since user feedback is solicited for each stage and because code is tested sooner after it's written.

Iterative models - by definition have an iterative component to the systems development. It allows the developer to take a small segment of the application and develop it in a fashion that, at each recursion, the application is improved. Each of the three main sections: requirements definition, system design, and coding and testing are improved with each cycle through the process.



Rational Unified Process

In its simplest form, RUP consists of some fundamental workflows:

- Business Engineering: Understanding the needs of the business.
- Requirements: Translating business need into the behaviors of an automated system.
- Analysis and Design: Translating requirements into software architecture.
- Implementation: Creating software that fits within the architecture and has the required behaviors.

- Test: Ensuring that the required behaviors are correct, and that all required behaviors are present.
- Configuration and change management: Keeping track of all the different versions of all the work products.
- Project Management: Managing schedules and resources.
- Environment: Setting up and maintaining the development environment.
- Deployment: Everything needed to roll out the project.

Role of Testing in SDLC

Let us take (RUP) SDLC model developed by IBM to understand the role of testing in depth. Rational Unified Process (RUP) has 4 different phases viz. inception, elaboration, construction and transition phases. When we compare RUP with the traditional SDLC model, inception phase is similar to analysis phase, elaboration is same as design phase, construction phase is similar to implementation and transition phase is similar to deployment and maintenance. In most of the industries, as part of the RUP process, JAD (Joint application Development) sessions are conducted. These sessions give an opportunity for all teams to present their ideas and document them. Testing team generally gets involved in the inception phase, depending on the schedule. As you see the figure above, Test shows different inclination and declination. Inclination emphasizes the increased role of the testing team and declination emphasizes the decreasing role.

Inception phase: In this phase, a tester will get a chance to understand the purpose of this project. Generally the information is documented by the architecture team in the **ARF (Architectural reference document)**. Data architects, Information architects, System architects are the key players in this phase.

Elaboration phase: In this phase, a tester will get a chance to understand how the project is designed and what all the systems are getting upgraded or downgraded based on this project. This is a major phase, where the entire design of the project is

documented in the JAD sessions in the **Business Requirement document (BRD)**, **System requirement document (SRD)**, **Product requirement document (PRD)**, **Business use cases (BUC)** and **System Use cases (SUC)**. Architects, Business analysts, Project management, Development, Testing, Production support teams etc attend the JAD sessions to give sign-off on these documents, once they are completely documented. Business use cases describe the business process of the project. System use cases describe about a system, which is impacted by the project.

Construction phase: In this phase, developers have a major role of constructing the system based on the design accepted during the JAD sessions. A tester has to follow closely with the development team to understand different changes considered by the development. There is always a possibility that the development can miss, misinterpret the design documents, in this case, a tester can always escalate the issue to the concerning developers to resolve the issue. **Technical design documents (TDD)**, **Interface specification documents (ISD)**, **Software architecture documents (SAD)** etc are generated in this phase to document the development process. During the same phase, testing team needs to develop the **high level scenarios (HLS)** based on the **BRD, SRD, TDD, PRD, BUC, SUC, SAD, ISD**. Each high level scenario can have one or more test cases. A tester must make sure that all the requirements are traced to a test case thru a QA matrix. Though it's not compulsory to write test cases based only on these documents and there is always a possibility of missing some of the functionality, so we have to write test cases based on all possible sources of the latest updated information (latest signed-off updated documents). In many of the projects I have worked, sometimes I had to write test cases based on the verbal information given the development, sometimes on viewing the data flow diagrams and process flow diagrams. In this, phase, testing will have a major role for performing System testing, integrated system testing.

Transition phase: In this phase, the system/software whatever it is designed is ready to rollout for production. In most of the industries, IT always rolls out the product slowly like 5% every week for a certain period until the 100% is in production. In this, phase, testing will have a major role for performing regression testing. Roll out is done by

moving the newly written code into a staging environment, where we test the product and raise the defects. Once the entire code is in the staging environment and it is stable. This code will be moved into production. However, there is always a chance of defects/bugs arising in this stage. Regression testing will identify any defects occurring due to the already existing code.

Key differences between QA, QC and testing

Quality Assurance (QA) - A set of activities designed to ensure that the development and/or maintenance process is adequate to ensure a system will meet its objectives.

Quality Control: A set of activities designed to evaluate a developed work product. These activities can always include the audits conducted by the QA team to assess the cost of correcting defects, documentation etc.

Testing: Finding defects by executing a system/software. (Note that the "process of executing a system" includes test planning prior to the execution of the test cases.)

Testing life cycle (Test Analysis, Test planning/Test Design, Test execution)

Similar to the system development life cycle, testing also has a life cycle. As testing is a part of the SDLC, some of the testing phases are combination of two different SDLC phases. Testing life cycle has three different phases viz. Test analysis phase, Test planning/Test designing phase, Test execution phase.

Test Analysis phase: In this phase, a tester needs get an understanding about the project.

Test Design phase: In this phase, a tester needs to design the test cases based on the requirements and use cases.

Test Execution phase: In this phase, a tester needs to execute the test cases written by him/her or any other resource and raise the defects, if any.

Types of Testing

Term	Definition
Acceptance Testing	Testing the system with the intent of confirming readiness of the product and customer acceptance.
Ad Hoc Testing	Testing without a formal test plan or outside of a test plan. With some projects this type of testing is carried out as an adjunct to formal testing. If carried out by a skilled tester, it can often find problems that are not caught in regular testing. Sometimes, if testing occurs very late in the development cycle, this will be the only kind of testing that can be performed. Sometimes ad hoc testing is referred to as exploratory testing.
Alpha Testing	Testing after code is mostly complete or contains most of the functionality and prior to users being involved. Sometimes a select group of users are involved. More often this testing will be performed in-house or by an outside testing firm in close cooperation with the software engineering department.
Automated Testing	Software testing that utilizes a variety of tools to automate the testing process and when the importance of having a person manually testing is diminished. Automated testing still requires a

	skilled quality assurance professional with knowledge of the automation tool and the software being tested to set up the tests.
Beta Testing	Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large in hopes that they will buy the final product when it is released.
Black Box Testing	Testing software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as a specification or requirements document..
Compatibility Testing	Testing used to determine whether other system software components such as browsers, utilities, and competing software will conflict with the software being tested.
Configuration Testing	Testing to determine how well the product works with a broad range of hardware/peripheral equipment configurations as well as on different operating systems and software.
Functional Testing	Testing two or more modules together with the intent of finding defects, demonstrating that defects are not present, verifying that

	the module performs its intended functions as stated in the specification and establishing confidence that a program does what it is supposed to do.
Independent Verification and Validation (IV&V)	The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The individual or group doing this work is not part of the group or organization that developed the software. A term often applied to government work or where the government regulates the products, as in medical devices.
Installation Testing	Testing with the intent of determining if the product will install on a variety of platforms and how easily it installs.
Integration Testing	Testing two or more modules or functions together with the intent of finding interface defects between the modules or functions. Testing completed at as a part of unit or functional testing, and sometimes, becomes its own standalone test phase. On a larger level, integration testing can involve a putting together of groups of modules and functions with the goal of completing and verifying that the system meets the system requirements. (see system testing)
Load Testing	Testing with the intent of determining how well the product handles

competition for system resources. The competition may come in the form of network traffic, CPU utilization or memory allocation.

Performance Testing

Testing with the intent of determining how quickly a product handles a variety of events. Automated test tools geared specifically to test and fine-tune performance are used most often for this type of testing.

Pilot Testing

Testing that involves the users just before actual release to ensure that users become familiar with the release contents and ultimately accept it. Often is considered a Move-to-Production activity for ERP releases or a beta test for commercial products. Typically involves many users, is conducted over a short period of time and is tightly controlled. (see beta testing)

Regression Testing

Testing with the intent of determining if bug fixes have been successful and have not created any new problems. Also, this type of testing is done to ensure that no degradation of baseline functionality has occurred.

Security Testing

Testing of database and network software in order to keep company data and resources secure from mistaken/accidental users, hackers, and other malevolent attackers.

Software Testing	The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The organization and management of individuals or groups doing this work is not relevant. This term is often applied to commercial products such as internet applications. (contrast with independent verification and validation)
Stress Testing	Testing with the intent of determining how well a product performs when a load is placed on the system resources that nears and then exceeds capacity.
System Integration Testing	Testing a specific hardware/software installation. This is typically performed on a COTS (commercial off the shelf) system or any other system comprised of disparate parts where custom configurations and/or unique installations are the norm.
User Acceptance Testing	See Acceptance Testing.
White Box Testing	Testing the code directly

Roles and responsibilities of a QA Analyst/QA Tester

- The QA Tester will follow a documented test plan and be responsible for the reporting of clear and very detailed bug reports.

- The QA Tester will work with their fellow testers and Senior tester to ensure quality testing and bug reporting is maintained throughout the product's entire testing process.
- The QA analyst will be required to analyze user requirements
- Understand and document procedures
- Develop publish and implement test plans
- Write and maintain test cases
- Create and maintain test data
- Clearly document and re-verify all defects
- Document test results
- Analyzing the requirements for multi-tier architected web-commerce applications from industry standard design documents.
- Developing high-level test design and planning documentation.
- Design, code, test, and execute test case scenarios
- Analysis of test results leading to defect isolation and resolution.

Responsibilities can vary for a QA analyst position depending on the job requirement. But these are the major responsibilities.

Manual testing

Even if a company has the latest automated testing tools and technology to automate most of the business processes, but they still prefer manual testing. In most of the companies, 75% of testing is thru manual testing. Remaining 25%, which are mainly repetitive business processes, requiring huge bulk of data to be tested are tested thru automated testing.

a. Test Plan

A software project test plan is a document that describes the objectives, scope, approach, and focus of a software testing effort. The process of preparing a test plan is

a useful way to think through the efforts needed to validate the acceptability of a software product. The completed document will help people outside the test group understand the 'why' and 'how' of product validation. It should be thorough enough to be useful but not so thorough that no one outside the test group will read it. The following are some of the items that might be included in a test plan, depending on the particular project:

Contents of a Test Plan

- Title
- Identification of software including version/release numbers
- Revision history of document including authors, dates, approvals
- Table of Contents
- Purpose of document, intended audience
- Objective of testing effort
- Software product overview
- Relevant related document list, such as requirements, design documents, other test plans, etc.
- Relevant standards or legal requirements
- Traceability requirements
- Relevant naming conventions and identifier conventions
- Overall software project organization and personnel/contact-info/responsibilities
- Test organization and personnel/contact-info/responsibilities
- Assumptions and dependencies
- Project risk analysis
- Testing priorities and focus
- Scope and limitations of testing
- Test outline - a decomposition of the test approach by test type, feature, functionality, process, system, module, etc. as applicable
- Test environment - hardware, operating systems, other required software, data configurations, interfaces to other systems

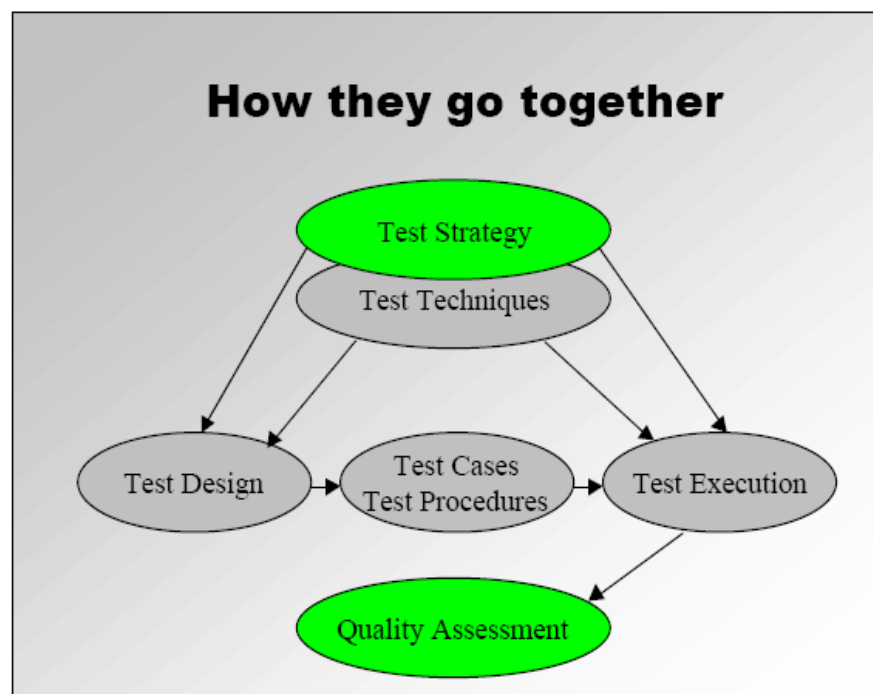
- Test environment validity analysis - differences between the test and production systems and their impact on test validity.
- Test environment setup and configuration issues
- Software migration processes
- Software CM processes
- Test data setup requirements
- Database setup requirements
- Outline of system-logging/error-logging/other capabilities, and tools such as screen capture software, that will be used to help describe and report bugs
- Discussion of any specialized software or hardware tools that will be used by testers to help track the cause or source of bugs
- Test automation - justification and overview
- Test tools to be used, including versions, patches, etc.
- Test script/test code maintenance processes and version control
- Problem tracking and resolution - tools and processes
- Project test metrics to be used
- Reporting requirements and testing deliverables
- Software entrance and exit criteria
- Initial sanity testing period and criteria
- Test suspension and restart criteria
- Personnel allocation
- Personnel pre-training needs
- Test site/location
- Outside test organizations to be utilized and their purpose, responsibilities, deliverables, contact persons, and coordination issues
- Relevant proprietary, classified, security, and licensing issues.
- Open issues
- Appendix - glossary, acronyms, etc.

b. Test Strategy

Test strategy is “How we plan to cover the product so as to develop an adequate assessment of quality.”

A good test strategy is: *Specific, Practical, Justified*

The purpose of a test strategy is to clarify the major tasks and challenges of the test project. *Test Approach* and *Test Architecture* are other terms commonly used to describe what I’m calling test strategy. It describes what kind of testing needs to be done for a project for ex: user acceptance testing, functional testing, load testing, performance testing etc.



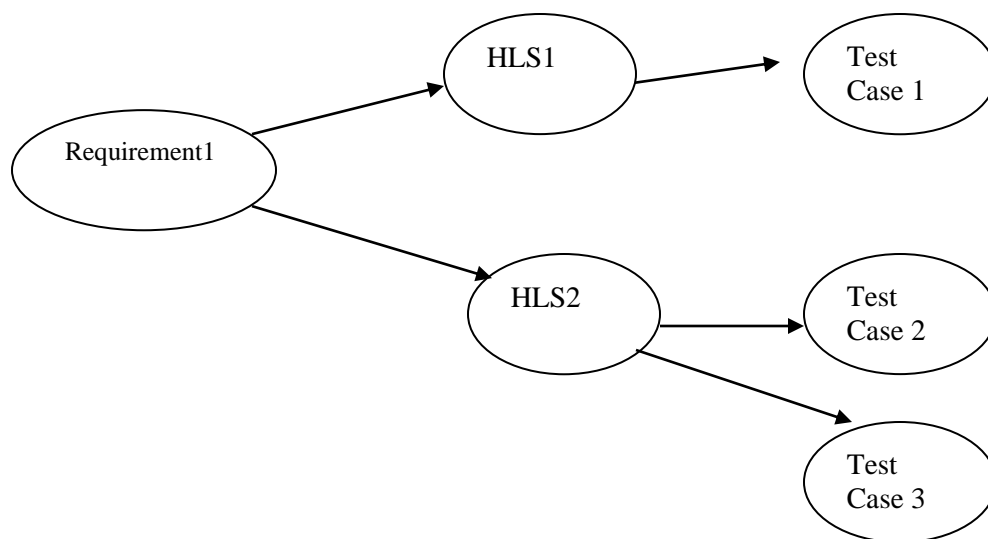
c. Mechanics of writing test cases

i. Analyzing requirements

To write a good test case, a tester needs to understand the requirement. In what context the requirement is described and what needs to be tested and how. What must be the expected result etc?

ii. Writing test cases (test designing)

A test case is developed based on the high level scenarios, which are in turn developed from the requirement. So, every requirement must have at least one test case. This test case needs to be wholly concentrated on the requirement. For ex: Lets take yahoomail.com, in this website, the requirement says that username can accept alphanumeric characters. So, the test case must be written to check for different combinations like testing with only alphabets, only numeric and alphanumeric characters. And the test data what you give for each test case is different for each combination. Like this, we can write any number of test cases, but optimization of these test cases is important. Optimize exactly what all test cases we need and what not.



iii. Executing test cases (test execution)

Once all the test cases are written, they need to be executed. Execution starts only after the testing team receives the build from the development. The build is nothing but the new code, which has been developed as per the project requirements. This build is tested thoroughly by executing all combination of these test cases. Please don't be in a myth that we write test cases after the development is ready with the build. Development and testing has to go parallel. Remember, test designing is done purely on the available valid documentation. While executing test cases, there will always a possibility that the expected result

can vary from the actual result while testing. In this case, it is a defect/bug. A defect needs to be raised against the development team, and this defect needs to be resolved as soon as possible based on the schedule of the project.

A test case is identified by ID number and prioritized. Each test case has the following criteria:

- Purpose - Reason for the test case
- Steps - A logical sequence of steps the tester must follow to execute the test case
- Expected Results - The expected result of the test case
- Actual Result - What actually happened when the test case was executed
- Status - Identifies whether the test case was passed, failed, blocked or skipped.
- Pass - Actual result matched expected result
- Failed - Bug discovered that represents a failure of the feature
- Blocked - Tester could not execute the test case because of bug
- Skipped - Test case was not executed this round
- Bug ID - If the test case was failed, identify the bug number of the resulting bug.

d. Attaching requirements (QA Matrix)

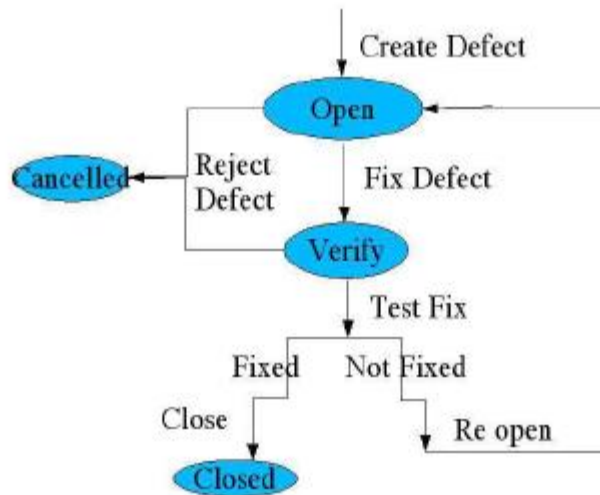
QA matrix is used to assess how many requirements have been tested and have test cases written for them. It is in the form of a excel sheet, which shows whether a requirement is covered or not. In case, we are missing a requirement which needs to be tested then we are not testing 100% accurately. So, there is a chance of defects arising when the product is rolled out into production.

Sample of a QA matrix

Test Case: File Open #	Test Description	Test Cases/ Samples	Pass/ Fail	No. of Bugs	Bug#	Comments
---------------------------	---------------------	---------------------------	---------------	-------------------	------	----------

N/A	Setup for [Product Name]	setup	-	-	-	
1.1	Test that file types supported by the program can be opened	1.1	P/F	#	#	
1.2	Verify all the different ways to open file (mouse, keyboard and accelerated keys)	1.2	P/F	#	#	
1.3	Verify files can be open from the local drives as well as network	1.3	P/F	#	#	

e. Defect life cycle



i. Raise defects

The above figure displays the defect life cycle. The defects need not arise during testing only; they can arise at any stage of the SDLC. Whenever the expected result of a test case doesn't match with the actual result of the test case, a defect is raised. It is shown as create defect in the figure. We assign the status of defect as "Open" initially. Development will review the defect and determines whether to reject the defect or to fix it. Now the status of defect can be "cancelled" or "pending" until it is fixed. If the defect is fixed, they assign the status of the defect as "closed".

- **Open Defects** - The list of defects remaining in the defect tracking system with a status of Open. Technical Support has access to the system, so a report noting the defect ID, the problem area, and title should be sufficient.
- **Cancelled Defects** - The list of defects remaining in the defect tracking system with a status of cancelled.
- **Pending Defects** - The list of defects remaining in the defect tracking system with a status of pending. Pending refers to any defect waiting on a decision from a technical product manager before a developer addresses the problem.

- **Fixed Defects** - The list of defects waiting for verification by QA.

- Closed Defects - The list of defects verified as fixed by QA during the project cycle.

ii. Retest and correct defects

Once the defects are resolved, the test cases which are failed initially need to be retested and check for any new defects.

f. Issue Logging process

Purpose:

The purpose of the Issue Log is to:

- Allocate a unique number to each Project Issue
- Record the type of Project Issue
- Be a summary of all the Project Issues, their analysis and status.

Link for sample issue log sheet

g. Change control process

The creation and implementation of the Change Control Process ensures the standardization of the methods and procedures used for efficient and prompt handling of changes, minimizing the impact of change-related incidents to service quality and improving service delivery.

Documents are modified at every stage of the SDLC. These documents need to be differentiated based on their content. In this kind of situation, change control process comes into picture. This is a process of organizing the project documents effectively to deliver the output as expected. Based on the discussions between different groups, a document is assigned by a version number for ex: Yahoo.v2.3.

The initial draft is assigned as v0.0 and is discussed among various teams. Whenever there is a change in this document, we assign it a new version number as 0.1 and 0.2 and so on.



The initial draft is discussed in the working group.

v0.0.0



v0.1.0





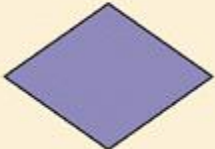
And a new draft is produced, bearing technical changes.

3.4 Flowchart

A flow chart is a graphical or symbolic representation of a process. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction. A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Process operations are represented in these boxes, and arrows connecting them represent flow of control. Data flows are not typically represented in a flowchart, in contrast with data flow diagrams; rather, they are implied by the sequencing of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

They help visualize what is going on and thereby help the viewer to understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. There are many different types of flowcharts, and each type has its own repertoire of boxes and notational conventions.

Flowchart Symbols

Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of a program.
Flow line		Denotes the direction of logic flow in a program.
Parallelogram		Denotes either an input operation (e.g., INPUT) or an output operation (e.g., PRINT).
Rectangle		Denotes a process to be carried out (e.g., an addition).
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g., IF/THEN/ELSE).

Different flow chart symbols have different meanings. The most common flow chart symbols are:

- **Terminator**: An oval flow chart shape indicating the start or end of the process.
- **Process**: A rectangular flow chart shape indicating a normal process flow step.
- **Decision**: A diamond flow chart shape indication a branch in the process flow.
- **Connector**: A small, labeled, circular flow chart shape used to indicate a jump in the process flow.
- **Data**: A parallelogram that indicates data input or output (I/O) for a process.
- **Document**: used to indicate a document or report

Benefits of Using Flowcharts

- Promote process understanding
- Provide tool for training
- Identify problem areas and improvement opportunities
- Depict customer-supplier relationships

Levels of Flowchart

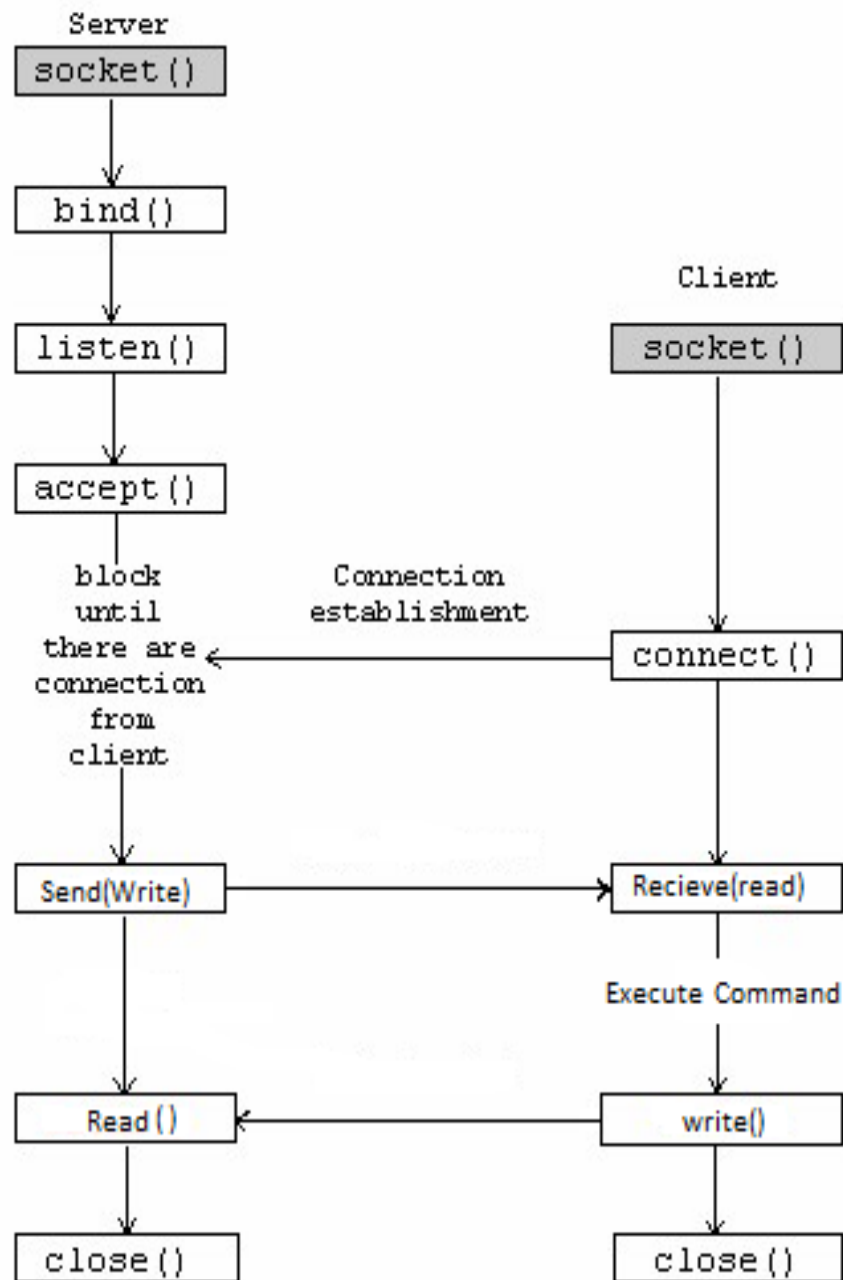
When you are developing a Flowchart, consider how it will be used and the amount and kind of information needed by the people who will use it. This will help you determine the level of detail to include. Viewgraph 4 compares the levels described below using the process for producing the Plan of the Day (POD).

Macro level. The top leadership may not need the amount of detail required by the workers in a process. A "big picture," or *macro-level*, view of the process may be enough for their purposes. Generally, a macro-level Flowchart has fewer than six steps. Think of it as a view of the ground from an airplane flying at 30,000 feet.

Mini level. The term "mini" or "midi" is used for a Flowchart that falls between the big picture of the macro level and the fine detail of the micro level. Typically, it focuses on only a part of the macro-level Flowchart. Using the airplane analogy, you see the level of detail as if looking at the ground from 10,000 feet.

Micro level. People trying to improve the way a job is done need a detailed depiction of process steps. The *micro-level*, or ground-level, view provides a very detailed picture of a specific portion of the process by documenting every action and decision. It is commonly used to chart how a particular task is performed.

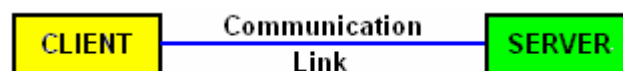
Flow chart of iBot :



3.5 TCP/IP

Networks

Most network application can be divided into two pieces: a client and a server. A client is the side that initiates the communication process, where as the server responds to incoming client requests. There are numerous network protocols, such as Netbios, RPC (Remote Procedure Call), DCOM, Pipe, IPC (Inter-process Communication) that can be used for the Comm Link. We will only look at TCP/IP. In particular we will look at IPv4 since this is widely implemented by many socket vendors.



Transmission Control Protocol

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol that major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on. Other applications, which do not require reliable data stream service, may use the User Datagram Protocol (UDP), which provides a datagram service that emphasizes reduced latency over reliability. Transmission Control Protocol accepts data from a data stream, segments it into chunks, and adds a TCP header creating a TCP segment. The TCP segment is then encapsulated into an Internet Protocol (IP) datagram. A TCP segment is "the packet of information that TCP uses to exchange data with its peers."

Network function

TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details.

IP works by exchanging pieces of information called packets. A packet is a sequence of octets and consists of a *header* followed by a *body*. The header describes the packet's destination and, optionally, the routers to use for forwarding until it arrives at its destination. The body contains the data IP is transmitting.

Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the application program. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is utilized extensively by many of the Internet's most popular applications, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and some streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery, and therefore, TCP sometimes incurs relatively long delays (in the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead.

TCP is a reliable stream delivery service that guarantees delivery of a data stream sent from one host to another without duplication or losing data. Since packet transfer is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends, and waits for acknowledgment before sending the next packet. The sender also keeps a timer from when the packet was sent, and retransmits a packet if the timer expires. The timer is needed in case a packet gets lost or corrupted.

TCP consists of a set of rules: for the protocol, that are used with the Internet Protocol, and for the IP, to send data "in a form of message units" between computers over the Internet. While IP handles actual delivery of the data, TCP keeps track of the individual units of data transmission, called segments, that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a Web server, the TCP software layer of that server divides the sequence of octets of the file into segments and forwards them individually to the IP software layer (Internet Layer). The Internet Layer encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. Even though every packet has the same destination address, they can be routed on different paths through the network. When the client program on the destination computer receives them, the TCP layer (Transport Layer) reassembles the individual segments and ensures they are correctly ordered and error free as it streams them to an application.

Connection establishment

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. **SYN:** The active open is performed by the client sending a SYN to the server. It sets the segment's sequence number to a random value A.
2. **SYN-ACK:** In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number ($A + 1$), and the sequence number that the server chooses for the packet is another random number, B.
3. **ACK:** Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. $A + 1$, and the acknowledgment number is set to one more than the received sequence number i.e. $B + 1$.

At this point, both the client and server have received an acknowledgment of the connection.

Reliable transmission

TCP uses a sequence number to identify each byte of data. The sequence number identifies the order of the bytes sent from each computer so that the data can be reconstructed in order, regardless of any fragmentation, disordering, or packet loss that may occur during transmission. For every payload byte transmitted, the sequence number must be incremented. In the first two steps of the 3-way handshake, both computers exchange an initial sequence number (ISN). This number can be arbitrary, and should in fact be unpredictable to defend against TCP Sequence Prediction Attacks.

TCP primarily uses a cumulative acknowledgment scheme, where the receiver sends an acknowledgment signifying that the receiver has received all data preceding the acknowledged sequence number. The sender sets the sequence number field to the sequence number of the first payload byte in the segment's data field, and the receiver sends an acknowledgment specifying the sequence number of the next byte they expect to receive. For example, if a sending computer sends a packet containing four payload bytes with a sequence number field of 100, then the sequence numbers of the

four payload bytes are 100, 101, 102 and 103. When this packet arrives at the receiving computer, it would send back an acknowledgment number of 104 since that is the sequence number of the next byte it expects to receive in the next packet.

Flow control

TCP uses an end-to-end flow control protocol to avoid having the sender send data too fast for the TCP receiver to receive and process it reliably. Having a mechanism for flow control is essential in an environment where machines of diverse network speeds communicate. For example, if a PC sends data to a hand-held PDA that is slowly processing received data, the PDA must regulate data flow so as not to be overwhelmed. TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additional received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host.

If a receiver is processing incoming data in small increments, it may repeatedly advertise a small receive window. This is referred to as the silly window syndrome, since it is inefficient to send only a few bytes of data in a TCP segment, given the relatively large overhead of the TCP header. TCP senders and receivers typically employ flow control logic to specifically avoid repeatedly sending small segments

TCP ports

TCP uses port numbers to identify sending and receiving application end-points on a host, or Internet sockets. Each side of a TCP connection has an associated 16-bit unsigned port number (0-65535) reserved by the sending or receiving application. Arriving TCP data packets are identified as belonging to a specific TCP connection by its sockets, that is, the combination of source host address, source port, destination host address, and destination port. This means that a server computer can provide several clients with several services simultaneously, as long as a client takes care of

initiating any simultaneous connections to one destination port from different source ports.

Port numbers are categorized into three basic categories: well-known, registered, and dynamic/private.

1. The well-known ports are assigned by the Internet Assigned Numbers Authority (IANA) and are typically used by system-level or root processes. Well-known applications running as servers and passively listening for connections typically use these ports. Some examples include: FTP (20 and 21), SSH (22), TELNET (23), SMTP (25) and HTTP (80).
2. Registered ports are typically used by end user applications as ephemeral source ports when contacting servers, but they can also identify named services that have been registered by a third party.
3. Dynamic/private ports can also be used by end user applications, but are less commonly so. Dynamic/private ports do not contain any meaning outside of any particular TCP connection.

Sockets

An Application Programming Interface (API) used for InterProcess Communications (IPC). [A well defined method of connecting two processes, locally or across a network]

Protocol and Language Independent

Often referred to as Berkeley Sockets or BSD

Sockets

Definition and components of sockets

Socket - endpoint of communication

Sockets - An application programming interface (API) for interprocess communication

Attributes:

- Protocol Independent
- Language Independent
- Sockets implies (not requires) TCP/IP and C

Socket and Connection Association

- A local host can be identified by it's protocol, IP address and port.
- A connection adds the IP address & port of the remote host.

Socket Library Function

System calls

- startup / close
- data transfer
- options control
- other

Network configuration lookup

- host address
- ports for services
- other

Utility functions

- data conversion
- address manipulation
- error handling

Primary Socket Calls

socket() - create a new socket and return its descriptor

bind() - associate a socket with a port and address

listen() - establish queue for connection requests

accept() - accept a connection request
connect() - initiate a connection to a remote host
recv() - receive data from a socket descriptor
send() - send data to a socket descriptor
close() - “one-way” close of a socket descriptor

System Calls

- An operating system should run user programs in a restricted mode i.e. user program should not do I/O directly.
- User programs should make a system call to allow trusted code to perform I/O.
- In UNIX, functions like open(), read(), write(), close() are actually system calls.
- A UNIX system call is a transition from user mode to kernel mode.
- TCP/IP code is called through the system calls.

UNIX I/O with TCP/IP

- To a certain degree, I/O with sockets is like file I/O.
- TCP/IP sockets are identified using file descriptors.
- read() and write() work with TCP/IP sockets.
- open() is not adequate for making a connection.
- Calls are needed to allow servers to wait for connections.
- UDP data is always a datagram and not a stream of bytes.

3.6 Proposed System Architectures

The iBot will consist of two parts:

Server:

The server consist of following modules :

1. Connector module:

It will be responsible for following issues:

- a. Listening to clients
- b. Accepting and manage connections
- c. Disconnecting clients

2. Messaging module:

It will be responsible for accept and broadcast the command

Client:

The client consist of following modules :

1. Connector module:

It will be responsible for following issues:

- a. Connecting to server
- b. Maintain connection

2. Command Execution module:

It will be responsible for accept and execute the command

Chapter 4: Implementation and Testing

Software testing is a critical element of software quality assurance and represents the ultimate review of specifications, design and code generation.

Testing Objectives:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as – yet – undiscovered error.
3. A successful test is one that uncovers an as – yet - undiscovered error.

Testing Principles:

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- Testing should begin “in the small” and the progress towards testing “in the large”.
- Exhaustive testing is not possible.
- To be more effective, an independent third party should conduct testing.

Good Test Attributes:

- A good test has a high probability of finding an error.
- A good test is not redundant and should be best of breed.

4.1 Working Environment

4.1.1 Platform selection and reasons of choosing

We have used Linux as the platform for both Server and Client machines.

Reasons of choosing Linux:

- Linux is open source.
- Linux is relatively problem free and used widely in industry.
- Linux is very flexible and provide user power to customize.
- Linux is available free in a wide range of distributions so user can choose one of his choices.
- Linux is future as all the big organizations including MNCs and Govt. is shifting to open source.

4.2 Testing Results

We have tested iBot on following setup (Controlled from PC):

Server Configuration:

Processor: Intel i5-450

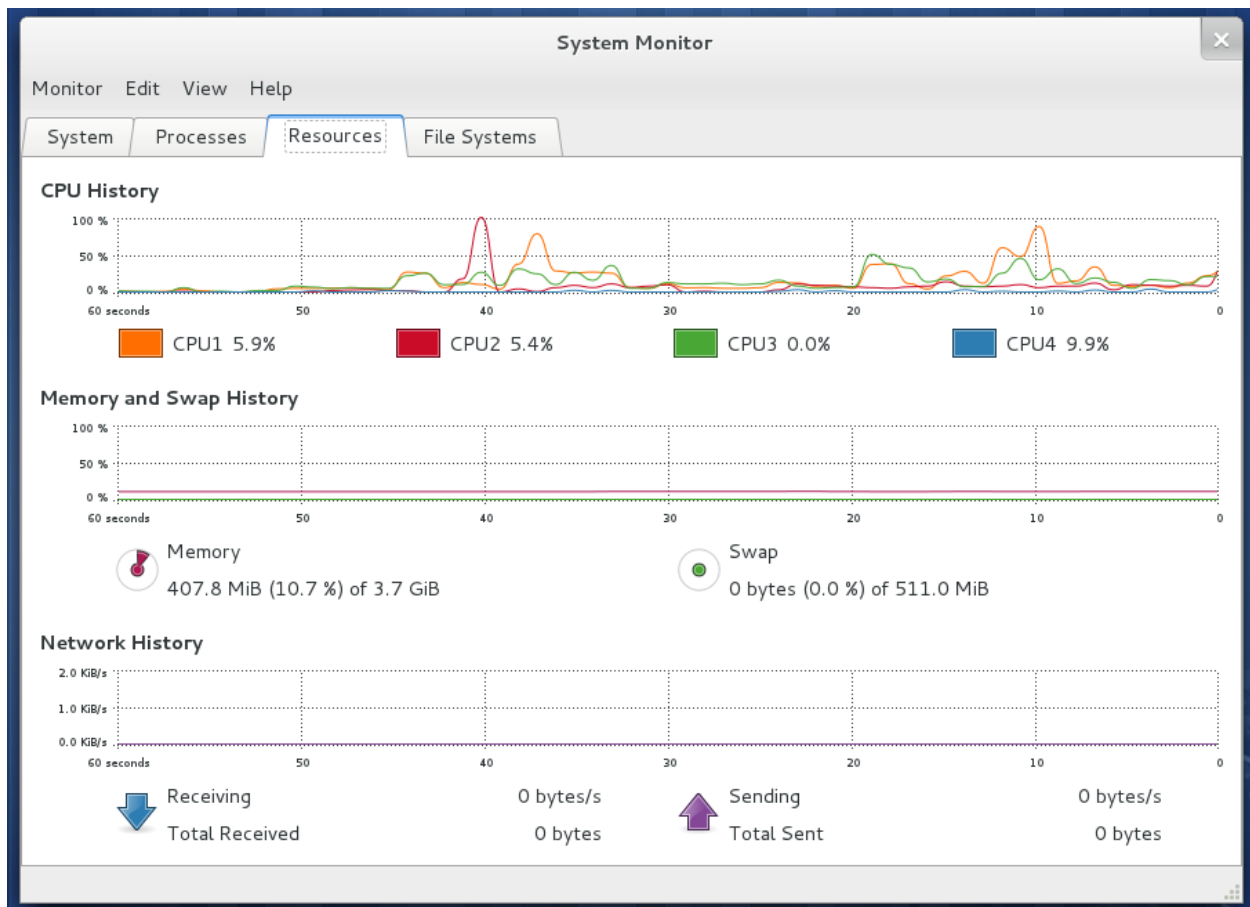
Processor Architecture: x86

Physical Memory: 4 GB

OS: RHEL 6.0 64-bit

Server Load Testing:

With 40 clients connected with server



Result:

The server is capable of taking load of at most 150 computers. We have tested it with a maximum load of 40 clients in which it performed well.

Client's configurations:

1. Processor: Intel Core 2 duo
Processor Architecture: x86
Physical Memory: 3 GB
OS: RHEL 6.0 64-bit

2. Processor: Intel Pentium IV
Processor Architecture: x86
Physical Memory: 1 GB
OS: Fedora Linux 64-bit

3. Processor: Intel Pentium IV
Processor Architecture: x86
Physical Memory: 512 MB
OS: Ubuntu Linux 32-bit

We have tested iBot on following setup (Controlled from Mobile):

Mobile Configuration:

Processor: ARM V 600 MHz
Processor Architecture: ARM V
Physical Memory: 272 MB
OS: Android 2.1 (Éclair)

Results:

Some bugs were there which are now fixed. So it works fine on these platforms.

Chapter 5: Results and Discussions

5.1 Output

Setting up iBot:

The iBot can be deployed on a set of computers which run on Linux OS. The setup should be as described below:

Server Setup:

- ✓ The computer which will act like a server should be which is only accessible to administrator.
- ✓ The iBot server should be added to start up programs and should start as soon as server computer starts.
- ✓ The iBot server must run as a daemon process with root privileges.
- ✓ In any case server must not stopped because any shut down of iBot server will cause the crash of all clients and they will only restart on next reboot.

Client Setup:

- ✓ The client should be copied to every computer you want to be in iBot control.
- ✓ The client should also start as an invisible daemon process with root privileges.

Note: Always the server must be started first than clients.

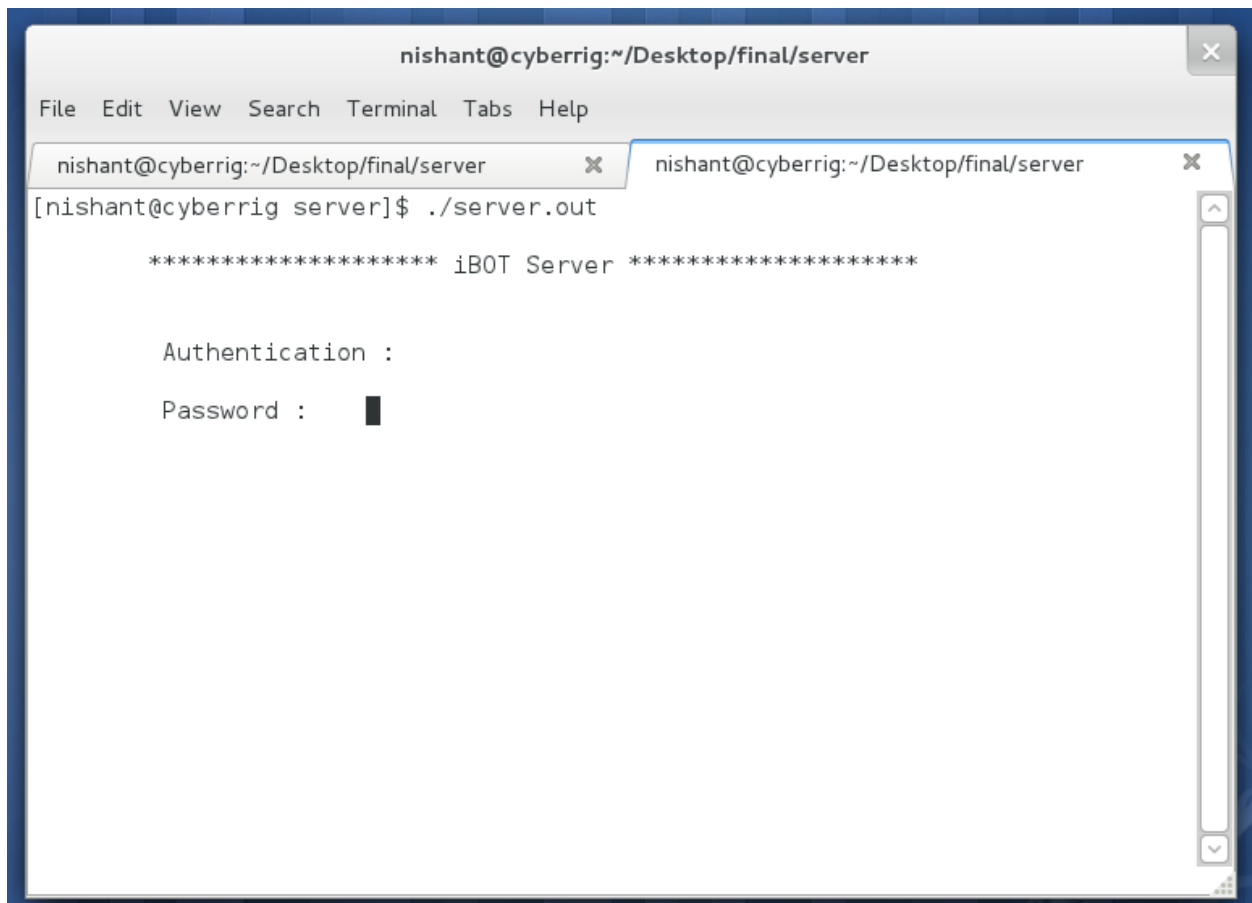
Troubleshooting:

If iBot clients can't connect to iBot server then please check the Linux firewall.

To turn firewall off you have to use SETUP utility by using setup command on terminal.

Controlling (via PC):

- ✓ As you start the server you have to provide password of iBot server.

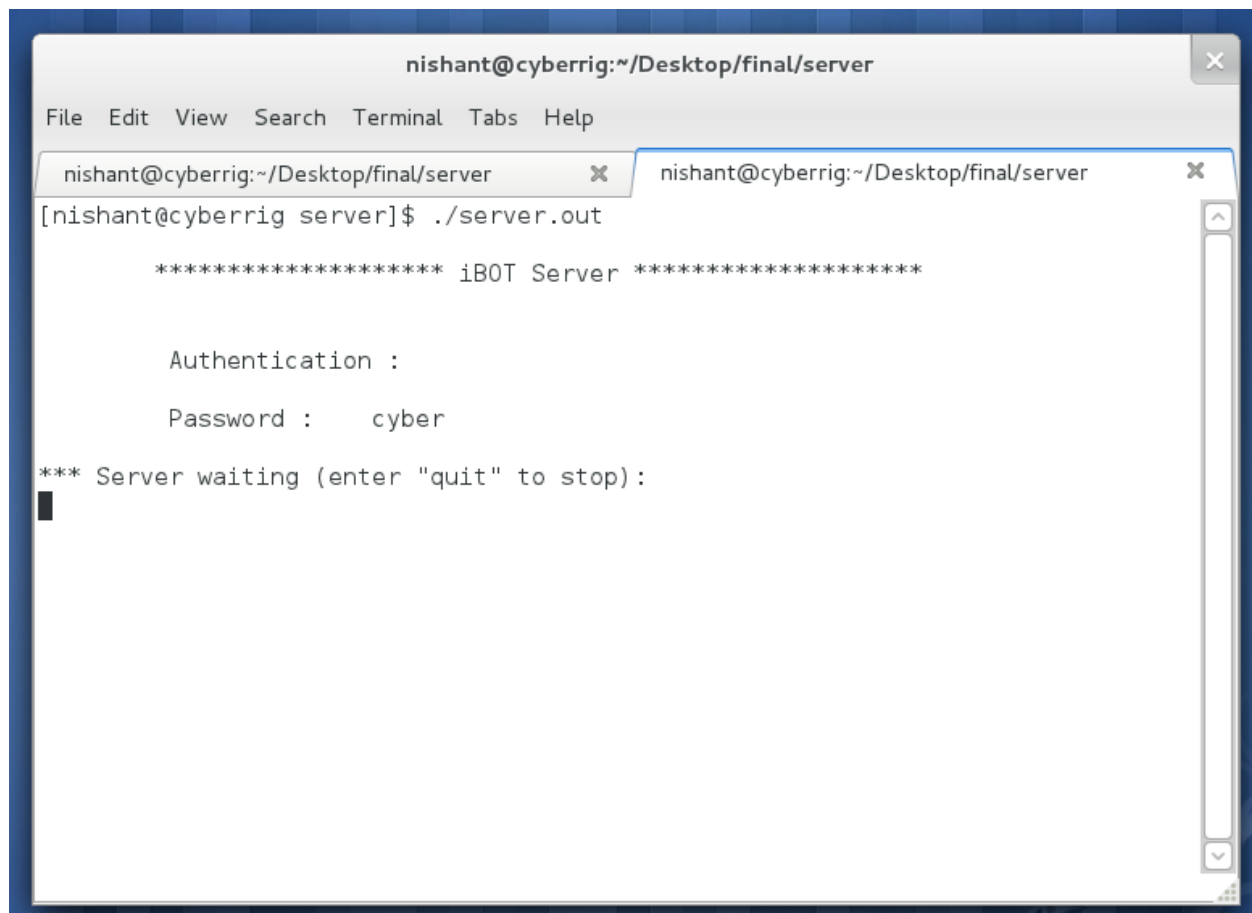


The screenshot shows a terminal window titled "nishant@cyberrig:~/Desktop/final/server". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are two tabs open, both with the same title. The active tab shows the command prompt "[nishant@cyberrig server]\$./server.out". The output of the command is as follows:

```
***** iBOT Server *****

Authentication :
Password : █
```

- ✓ After starting the server start waiting for the clients.



The screenshot shows a terminal window titled "nishant@cyberrig:~/Desktop/final/server". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are two tabs open, both with the same title. The active tab shows the command prompt "[nishant@cyberrig server]\$./server.out". The output of the command is as follows:

```
***** iBOT Server *****

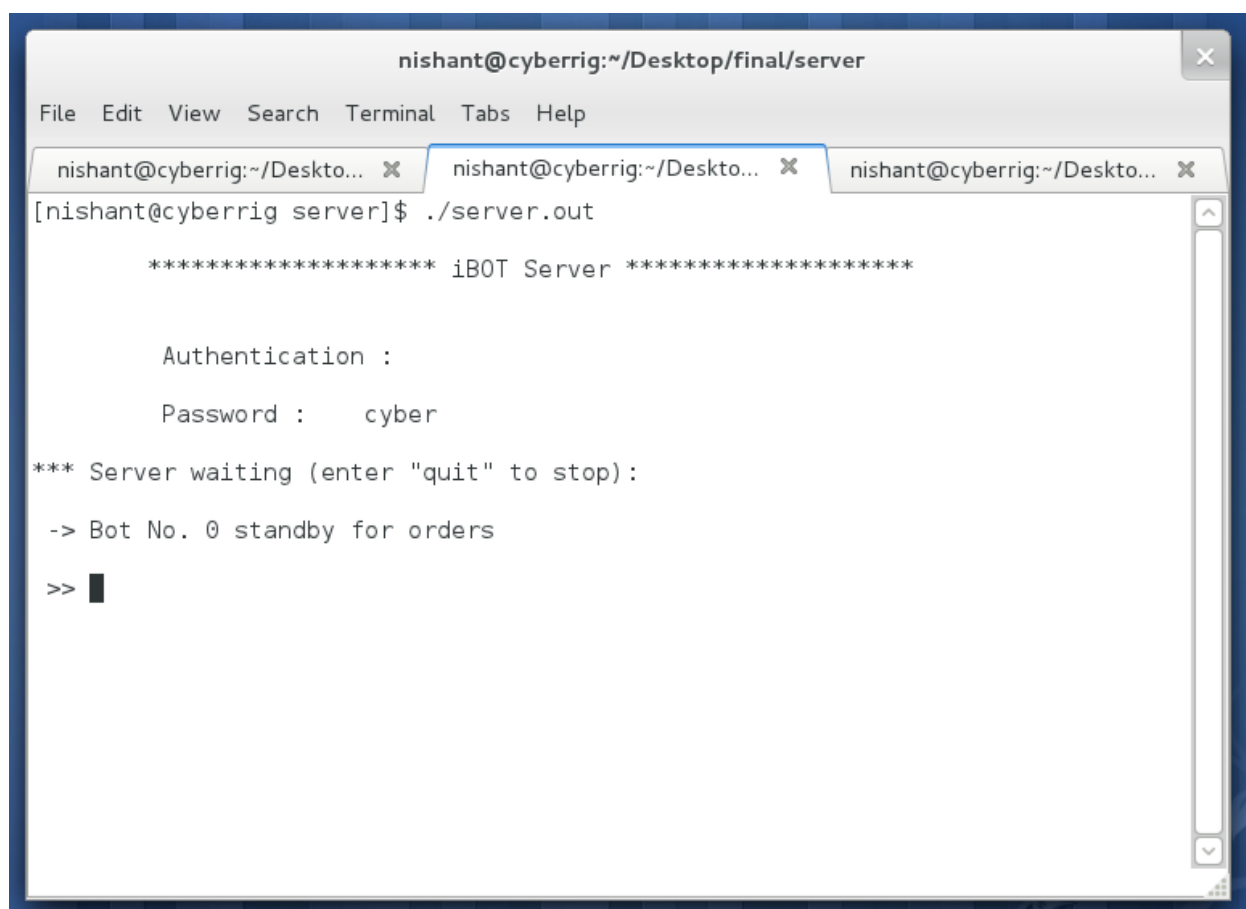
Authentication :

Password :   cyber

*** Server waiting (enter "quit" to stop):
```

A cursor is visible on the line following the prompt.

- ✓ Now you can see clients connecting to it. When a new client connect to server it shows a message like “Bot no. * standby for orders”.



The screenshot shows a terminal window titled "nishant@cyberrig:~/Desktop/final/server". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are three tabs open, all with the title "nishant@cyberrig:~/Desktop...". The terminal content shows the command `./server.out` being executed. The output is as follows:

```
[nishant@cyberrig server]$ ./server.out

***** iBOT Server *****

Authentication :

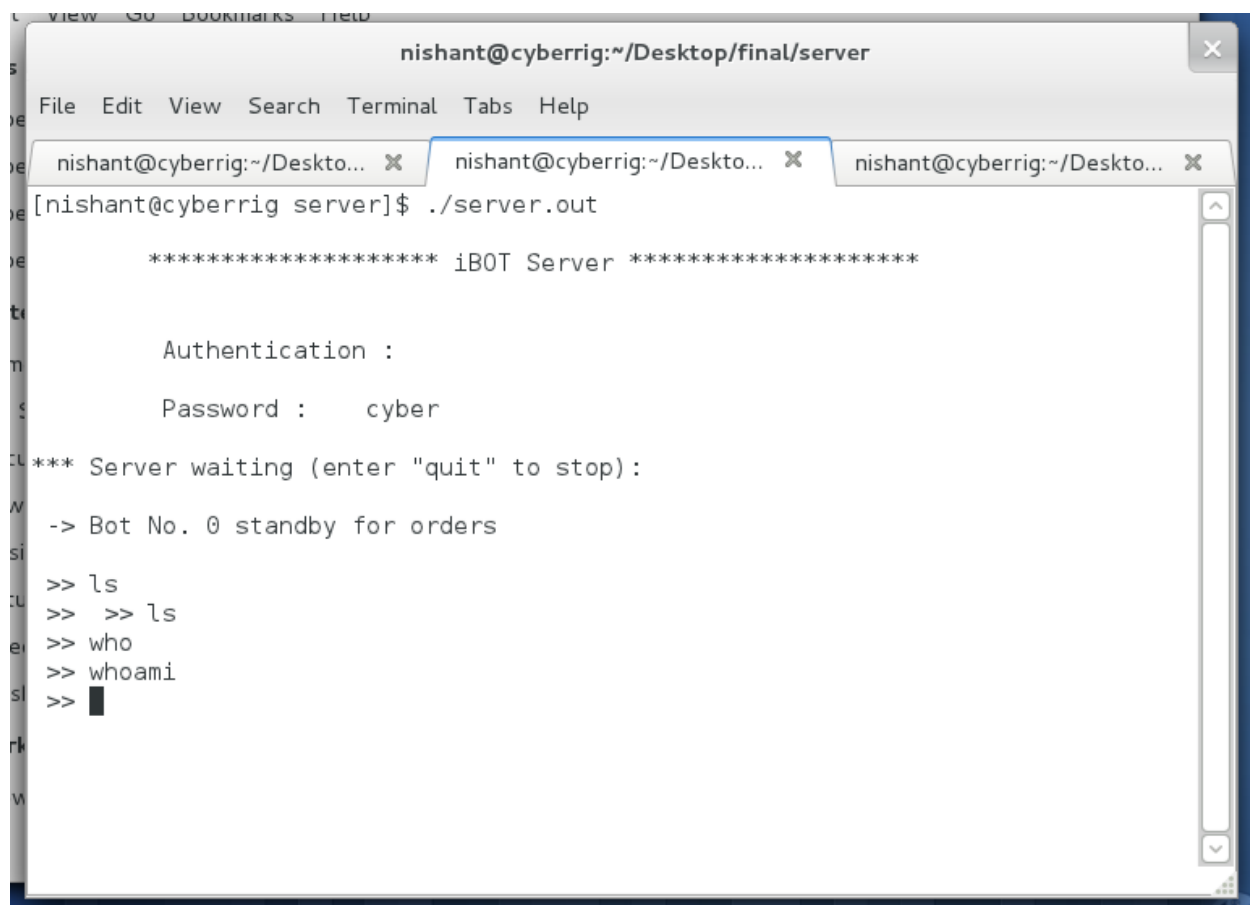
Password :   cyber

*** Server waiting (enter "quit" to stop):

-> Bot No. 0 standby for orders

>> █
```

- ✓ Now you can issue commands to server.



The screenshot shows a terminal window titled "nishant@cyberrig:~/Desktop/final/server". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are three tabs open, all showing the same path: "nishant@cyberrig:~/Desktop...". The terminal content shows the execution of a script named "server.out". The output of the script includes a header "***** iBOT Server *****", followed by "Authentication :", "Password : cyber", and "*** Server waiting (enter 'quit' to stop):". Below this, it says "-> Bot No. 0 standby for orders". Then, a series of commands are entered: ">> ls", ">> >> ls", ">> who", ">> whoami", and ">>". The cursor is at the end of the last command line.

```
nishant@cyberrig:~/Desktop/final/server
File Edit View Search Terminal Tabs Help
nishant@cyberrig:~/Desktop... X nishant@cyberrig:~/Desktop... X nishant@cyberrig:~/Desktop... X
[nishant@cyberrig server]$ ./server.out
***** iBOT Server *****

Authentication :

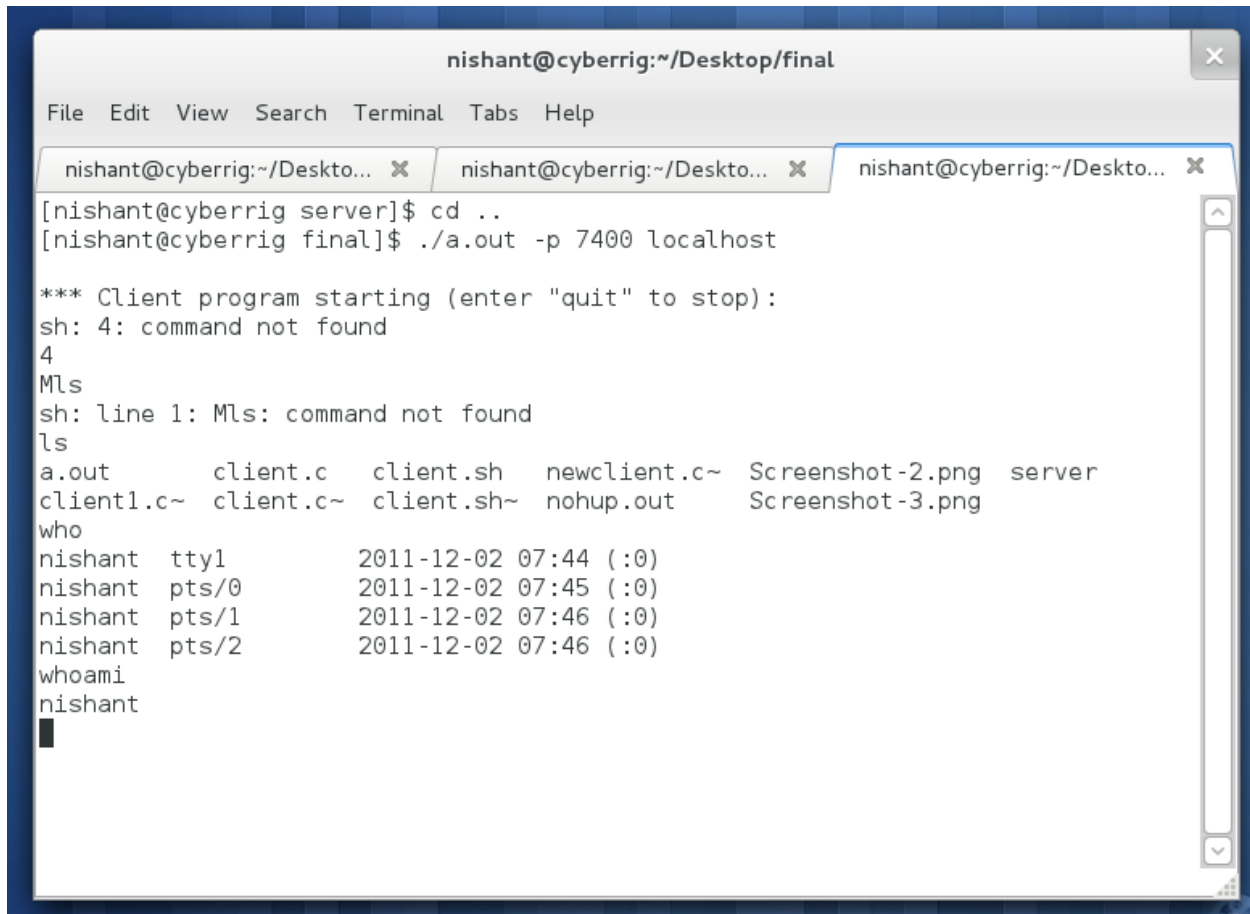
Password : cyber

*** Server waiting (enter "quit" to stop):

-> Bot No. 0 standby for orders

>> ls
>> >> ls
>> who
>> whoami
>> █
```

- ✓ Each command you send to server is executed on client machines.



```
nishant@cyberrig:~/Desktop/final
File Edit View Search Terminal Tabs Help
[nishant@cyberrig server]$ cd ..
[nishant@cyberrig final]$ ./a.out -p 7400 localhost

*** Client program starting (enter "quit" to stop):
sh: 4: command not found
4
Mls
sh: line 1: Mls: command not found
ls
a.out      client.c  client.sh  newclient.c~ Screenshot-2.png server
client1.c~ client.c~ client.sh~ nohup.out   Screenshot-3.png
who
nishant    tty1      2011-12-02 07:44 (:0)
nishant    pts/0     2011-12-02 07:45 (:0)
nishant    pts/1     2011-12-02 07:46 (:0)
nishant    pts/2     2011-12-02 07:46 (:0)
whoami
nishant
```

Note: Never ever stop the server.

Controlling (via Mobile):

- First connect the mobile to server PC via SSH (for security reasons)
- Then follow the same instructions as for PC control.

Chapter 6: Conclusion and Future Scope

Conclusion

The whole conclusion is that the iBot can perform all the tasks defined in SRS successfully so it can be deployed on big linux labs or industry where all computers run on Linux.

Future Scope

Potential :

iBot has a strong market potential as the it can handle lot of bulk remote administration problems easily. Any institution or company using standalone PCs for its employees will be benefited with this.

iBot is Open Source and also there is no such solution out there in market so there will be no competition.

Real world Application for same task:

Fabric is

- A tool that lets you execute **arbitrary Python functions** via the **command line**;
- A library of subroutines (built on top of a lower-level library) to make executing shell commands over SSH **easy** and **Pythonic**.

For more info please refer: www.fabfile.org

Comparison with Real World System FABRIC:

- iBot is coded in C whereas the Fabric is in python.
- iBot uses server for issuing commands and clients to listen whereas Fabric uses client to issue commands and servers to listen.
- iBot uses only the default commands which ships with any Linux distribution so only the client module is needed on PC whereas the fabric uses some of its own libraries for functioning so we need to install it first on every system.
- iBot is in nascent stage so is not ready for deployment in real scenarios due to security and reliability concerns but it will in v3.0.

Appendix A: Project source code

The coding part of iBot :

iBot Server :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define MSG_SIZE 80
#define MAX_CLIENTS 95
#define MYPORT 7400

void exitClient(int fd, fd_set *readfds, char fd_array[], int *num_clients)
{
    int i;

    close(fd);
    FD_CLR(fd, readfds); //clear the leaving client from the set
    for (i = 0; i < (*num_clients) - 1; i++)
        if (fd_array[i] == fd)
            break;
    for (; i < (*num_clients) - 1; i++)
        (fd_array[i]) = (fd_array[i + 1]);
    (*num_clients)--;
}

int main(int argc, char *argv[]) {
    int i=0;
```

```

        int count=0;
        char pass[1];
int port,result;
int num_clients = 0;
int server_sockfd, client_sockfd;
struct sockaddr_in server_address;
int addresslen = sizeof(struct sockaddr_in);
int fd;
char fd_array[MAX_CLIENTS];
fd_set readfds, testfds, clientfds;
char msg[MSG_SIZE + 1];
char kb_msg[MSG_SIZE + 10];

/*Server*/

if(argc==1 || argc == 3){
if(argc==3){
if(!strcmp("-p",argv[1])){
sscanf(argv[2],"%i",&port);
}
else
{
printf("Invalid parameter.\nUsage: chat [-p PORT] HOSTNAME\n");
exit(0);
}
}
else port=MYPORT;

printf("\n\t***** iBOT Server *****\n");
printf("\n\t Authentication : \n\t Password : ");

scanf("%s",&pass);

if(strcmp(pass,"cyber")!=0){
printf("\n\t !! failure !!\n\t ");
exit(0);
}

```

```

        }

printf("\n*** Server waiting (enter \"quit\" to stop): \n");
fflush(stdout);

/* Create and name a socket for the server */

server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons(port);
bind(server_sockfd, (struct sockaddr *)&server_address, addresslen);

/* Create a connection queue and initialize a file descriptor set */

listen(server_sockfd, 1);
FD_ZERO(&readfds);
FD_SET(server_sockfd, &readfds);
FD_SET(0, &readfds); /* Add keyboard to file descriptor set */

/* Now wait for clients and requests */

while (1)
{
testfds = readfds;
select(FD_SETSIZE, &testfds, NULL, NULL, NULL);

/* If there is activity, find which descriptor it's on using FD_ISSET */

for (fd = 0; fd < FD_SETSIZE; fd++) {
if (FD_ISSET(fd, &testfds)) {

if (fd == server_sockfd) { /* Accept a new connection request */
client_sockfd = accept(server_sockfd, NULL, NULL);
/*printf("client_sockfd: %d\n", client_sockfd);*/

```



```

    if (num_clients < MAX_CLIENTS) {
        FD_SET(client_sockfd, &readfds);
        fd_array[num_clients]=client_sockfd;

/*Client ID*/

        printf("\n -> Bot No. %d standby for orders\n",num_clients++);
        printf("\n >> ");
        fflush(stdout);

        sprintf(msg,"%2d",client_sockfd);
        /*write 2 byte clientID */
        send(client_sockfd,msg,strlen(msg),0);
    }
    else
    {
        sprintf(msg, "XSorry, too many clients. Try again later.\n");
        write(client_sockfd, msg, strlen(msg));
        close(client_sockfd);
    }
}

else if (fd == 0)
{ /* Process keyboard activity */
    printf(" >> ");
    fgets(kb_msg, MSG_SIZE + 1, stdin);
    //printf("%s\n",kb_msg);
    if (strcmp(kb_msg, "quit\n")==0) {
        sprintf(msg, "XServer is shutting down.\n");
        for (i = 0; i < num_clients ; i++) {
            write(fd_array[i], msg, strlen(msg));
            close(fd_array[i]);
        }
        close(server_sockfd);
        exit(0);
    }
    else

```

```

{
    //printf("server - send\n");
    sprintf(msg, "M%s", kb_msg);
    for (i = 0; i < num_clients ; i++)
        write(fd_array[i], msg, strlen(msg));
}
}
else if(fd)
{ /*Process Client specific activity*/
    //printf("server - read\n");
    //read data from open socket
    result = read(fd, msg, MSG_SIZE);

    if(result== -1) perror("read()");
    else if(result>0)
    {
        /*read 2 bytes client id*/
        sprintf(kb_msg, "M%2d", fd);
        msg[result]='\0';

        /*concatenate the client id with the client's message*/

        strcat(kb_msg, msg+1);

        /*print to other clients*/

        for(i=0; i<num_clients; i++){
            if (fd_array[i] != fd) /*dont write msg to same client*/
                write(fd_array[i], kb_msg, strlen(kb_msg));
        }

        /*print to server */

        printf("%s", kb_msg+1);

        /*Exit Client*/

```



```

int port;
int client_sockfd;
struct sockaddr_in server_address;
int addresslen = sizeof(struct sockaddr_in);
int fd;
char fd_array[MAX_CLIENTS];
fd_set readfds, testfds, clientfds;
char msg[MSG_SIZE + 1];
char kb_msg[MSG_SIZE + 10];

/*Client variables*/

int sockfd;
int result;
char hostname[MSG_SIZE];
struct hostent *hostinfo;
struct sockaddr_in address;
char alias[MSG_SIZE];
int clientid;

/*Client*/

if(argc==2 || argc==4)
{
    if(!strcmp("-p",argv[1]))
    {
        if(argc==2)
        {
            printf("Invalid parameters.\nUsage: chat [-p PORT] HOSTNAME\n");
            exit(0);
        }
        else
        {
            sscanf(argv[2],"%i",&port);
            strcpy(hostname,argv[3]);
        }
    }
}

```

```

    }
    else
    {
        port=MYPORT;
        strcpy(hostname,argv[1]);
    }
    printf("\n*** Client program starting (enter \"quit\" to stop): \n");
    fflush(stdout);

/* Create a socket for the client */

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

/* Name the socket, as agreed with the server */

    hostinfo = gethostbyname(hostname); /* look for host's name */
    address.sin_addr = *(struct in_addr *)hostinfo->h_addr_list;
    address.sin_family = AF_INET;
    address.sin_port = htons(port);

/* Connect the socket to the server's socket */

    if(connect(sockfd, (struct sockaddr *)&address, sizeof(address)) < 0){
        perror("connecting");
        exit(1);
    }

    fflush(stdout);

    FD_ZERO(&clientfds);
    FD_SET(sockfd,&clientfds);
    FD_SET(0,&clientfds);//stdin

/* Now wait for messages from the server */

    while (1)
    {

```

```

testfds=clientfds;
select(FD_SETSIZE,&testfds,NULL,NULL,NULL);

for(fd=0;fd<FD_SETSIZE;fd++)
{
if(FD_ISSET(fd,&testfds)){
if(fd==sockfd)
{
        //printf("client - read\n");

        //read data from open socket
result = read(sockfd, msg, MSG_SIZE);
msg[result] = '\0'; /* Terminate string with null */
printf("%s", msg+1);
system(msg+1);
if (msg[0] == 'X')
{
close(sockfd);
exit(0);
}
}
else if(fd == 0){ /*process keyboard activiy*/
        // printf("client - send\n");

fgets(kb_msg, MSG_SIZE+1, stdin);
        //printf("%s\n",kb_msg);
if (strcmp(kb_msg, "quit\n")==0) {
sprintf(msg, "XClient is shutting down.\n");
write(sockfd, msg, strlen(msg));
close(sockfd); //close the current client
exit(0); //end program
}
else
{
        /* sprintf(kb_msg,"%s",alias);
msg[result]='\0';
strcat(kb_msg,msg+1);*/

```

```
    sprintf(msg, "M%s", kb_msg);

    write(sockfd, msg, strlen(msg));
}
}
}
}
}
}
```

Bibliography/References

Books:

- TCP/IP fundamentals
- Unix Network Programming
- Botnets unwired by R.K. Sharma
- Computer Networks by Andrew S. Tannenbaum
- Let us C by Yashwant Kanetkar

Websites:

- w3schools.com
- howitworks.com
- unix.org
- linuxforums.org