A Report on the "Handwriting Recognition For Indic Languages"

at

Red Hat Software Services (India) Pvt Ltd

Submitted by

Amit Kumar Jha

PRN: 08030247131

MBA *(Systems)*

(2008-10)

Symbiosis Centre for Information Technology
(a constituent member of Symbiosis International University (SIU), estd., under Section 3 of UGC Act, 1956 by Notification No. F.9-12/2001-U-3 of Govt. of India)
Year of submission 2009

CONTENTS

CHAPTER – 1
INTRODUCTION TO THE PROJECT

1.1)Purpose/Objectives:

Already there is existing handwriting recognition software called Tomoe. Tomoe interprets only input characters which are of simplified Chinese or enhanced Japanese types. The reason for this is that there are only simplified Chinese and unihan dictionary support existing for the current recognition system. This project is aimed at extending the same recognition capabilities for Indic languages like Devanagari and others. In brief the following were the objectives of this project:-

1.1.1) To develop an application/Modify an already existing application like Tomoe to include the provision for Devanagari language recognition.

1.1.2) Prepare the database for inclusion of the Devanagari script recognition.

1.1.3) Formatting of the modified database .

1.1.4) Words in Devanagari script like (Ksh) which are a combination of two or more words are to be directly interpreted.

1.1.5) Modification in the Engine for Devanagari script recognition.

1.2)Scope:

This Project is limited to developing recognition capabilities of Devanagari Script only. Although once this is done, Red Hat will try to upgrade this to include other Indic languages also. Apart from this the technologies used have to be open source in nature . This is also a clause for development. The time period allocated for execution was 3.5

months (from 6th april 2009 to 20 july 2009) . Although I will be associated with this project even after the completion of my
official internship period. I would continue to contribute to this project and make this project efficient enough to be used commercially.

1.3)Theme of the research Problem:

Theme :- Handwriting recognition is the ability of a computer to receive and interpret intelligible handwritten input from different sources (for ex. Touch screens). Handwriting recognition is implemented using specific software called Recognition Engines. Whenever any input is provided to the computer using touchpad or mouse, the search engine interprets the handwriting.

Most engines also provide a list of candidate characters which acts as an option list for the user. This list gives the user choices of matching characters which he might want to input .There is very high potential usage of these recognition systems in the field of optical character recognition (OCR), PDA's, Palm tops and Tablet computers.

Already there is existing handwriting recognition software called Tomoe. Tomoe interprets only input characters which are of simplified Chinese or enhanced Japanese types. The reason for this is that there are only simplified Chinese and unihan dictionary support existing for the current recognition system. This project is aimed to extend the same recognition capabilities for Indic languages like Devanagari and others.

1.4)Business Aspects:- There have been few handwriting recognition initiatives in the past which have failed . Maintaining a proper success rate in recognition of input is critical. This can be achieved only by using a very efficient algorithm for pattern recognition. Major business usages for this application would be in *personal digital assistant (PDA),* tablet PCs and also advanced Mobile phones. All these gadgets can be equipped with handwriting recognition engine supporting their regional or national dialect. This could be a massive USP of electronic gadget manufacturers.

Also these engines are nowadays applied for intelligent document recognition (IDR) in various leading corporations across the globe .IDR basically is contextual understanding and segregation of incoming documents. It has extensive usage in Business Intelligence and Business process Improvement.

Another major utilization of handwriting recognition engines is in the field of image recognition. For example, facial recognition software can be built using the same principle, with little modifications. A dictionary of faces of criminals can be maintained and when on description by an informant a user inputs image features on a PDA or Laptop, the engine gives options to select a matching face.

This technology is already being used by both foreign and top national agencies but it is not very accurate. Also the usage of such technology is extremely costly and is limited only to elite agencies like FBI, CIA, RAW, CBI and Special Forces. Improving the algorithm can lead to more demand and usage of this technology.

1.5)Research Problem under study:

The elementary problem faced in this task was to port the existing application (Tomoe, which is a Japanese handwriting recognition software) for recognizing a new language, Devanagari. Another Major Research problem faced in this project was to determine ways of improving the algorithm for recognition in the Tomoe application chosen as the benchmark for the development our project.

1.6)Methodology of conduct of Research:

1.6.1) I did a preliminary study of the Tomoe application (its source code basically), Tomoe is a handwriting recognition library for Chinese and Japanese characters. The Tomoe project provides the library itself as well as a character stroke database,handwriting recognition canvas for GTK applications, a character stroke data editor, and modules for common Unix/Linux input methods.

1.6.2) Tomoe takes advantage of the data structures and object oriented programming features provided by the glib library.

1) The following is the understanding which I got from my study:-

Tomoe consists of the following classes, for which I have also described the functionalities in brief :-

· **TomoeContext -** Interface to the character recognition functions.

· **TomoeWriting -** A handwritten character. You fill this in with coordinates from the
user's handwriting.

TomoeQuery - The query that you pass to a TomoeContext to perform recognition. Contains the TomoeWriting that you wish to perform recognition on.

TomoeChar - Tomoe's character wrapper.

TomoeCandidate - A candidate character, together with a rating. A list of these is returned to you from TomoeContext after an attempted recognition.

The initial step would be to prepare a database for Devanagari. This would be done by studying the various script components(alphabets) in Devanagari . A list of corresponding characteristic points and the angles for each alphabet in Devanagari script is to be prepared and stored
in a formatted manner in the existing database of Tomoe application .

Once the above step is done ,then upon entering an alphabet of Devanagari script in the Tomoe Canvas ,the handwriting recognition function/method would be called. This function would get the Input points from the
canvas and the stored characteristic points & angles from the database(also called dictionary in Tomoe). Both these entities would form the parameters for the handwriting recognition function.

Then a matching of the Input parameters (via canvas) and the stored parameters(via database) would be done by the function. A score would be given for each degree of match.(for ex the greater the degree of match ,the lower can be the score).A cut off would be decided for this score, beyond which the comparison would not at all be considered valid. Lastly the function would return the List of prospective candidates (which had a score within the decided cutoff limit) and these would be displayed for the user to select.

1.7)INTRODUCTION OF THE COMPANY

Company Profile

Founded in 1993, Red Hat is the leader in enterprise Linux and is the most recognized open source brand in the world. They serve global enterprises with technology and services made possible by the open source model. Solutions include Red Hat Enterprise Linux, JBoss Enterprise middleware, and a broad range of management and services: consulting, 24x7 support, Red Hat Network. Red Hat's global training program operates worldwide and features RHCE, the global standard Linux certification.

Corporate Fact Sheet for Red Hat

Profile:

Red Hat, the world's leading open source and Linux provider, is headquartered in Raleigh, NC with satellite offices worldwide. Red Hat is leading Linux and open source solutions into the mainstream by making high-quality, low-cost technology accessible. Red Hat provides operating system platforms along with middleware, applications, and management solutions, as well as support, training, and consulting services to customers worldwide and through top-tier partnerships. Red Hat's open source strategy offers customers a long-term plan for building infrastructures that are based on and leverage open source technologies with a focus on security and ease of management.

Red Hat's mission:

To be the catalyst in communities of customers, contributors, and partners creating better technology the open source way.

Founded:

1993

Number of Employees:

Over 2,500 worldwide

Ownership:

Publicly held (NYSE:RHT) as of August, 1999

Website:

www.redhat.com

Headquarters:

1801 Varsity Drive
Raleigh, NC 27606
+1 919 754 3700

Key customers:

Red Hat customers include enterprise organizations, academic and research institutions, and local, state and federal governments. Our customers include:

- New York Stock Exchange
- McKesson
- Amazon.com
- AOL
- Merrill Lynch
- Credit Suisse

- DreamWorks
- VeriSign
- Charles Schwab
- UBS Warburg
- Morgan Stanley

Key partners:

AMD, Dell, Fujitsu, Hitachi, HP, IBM, NEC, Oracle, SAP, Sybase, Symantec, Intel

Key Awards:

- CIO Insight Magazine's Most Valued Vendor, 2007
- CIO Insight Magazine's Most Valued Vendor, 2006
- CIO Insight Magazine's Most Valued Vendor, 2005

Management:

Jim Whitehurst
President and Chief Executive Officer

Charlie Peters
Executive Vice President and Chief Financial Officer

Alex Pinchev
Executive Vice President and President Global Sales, Services and Field Marketing

Paul Cormier
Executive Vice President and President, Products and Technologies

Michael Cunningham
Executive Vice President and General Counsel

DeLisa Alexander
Senior Vice President, People and Brand

Lee Congdon
Chief Information Officer

Brian Stevens
CTO and Vice President, Engineering

Nick Van Wyk
Vice President, Operations and Senior Transformation Executive

Subsidiaries

Red Hat Software Services (India) Pvt Ltd

Red Hat Software Services (India) Pvt Ltd is a subsidiary of the open source software company Red Hat, Inc., USA. Red Hat India has been at the forefront of bringing the latest open source and Linux technology to India. It has enabled adoption of open source technology across segments -- government, enterprise and education."

Offices, partners

Red Hat Software Services (India) Pvt Ltd has six offices in India at Mumbai, Pune, New Delhi, Bangalore, Kolkata, and Chennai and one in Sri Lanka at Colombo. It says its "OEM partners" include IBM, DELL, HP, HCL Infosystems, Wipro Infotech and Acer, among others. And "SI partners" include Wipro Infotech and HCL Infosystems.

Distribution network

Red Hat Software Services (India) Pvt Ltd says it currently has a distribution network of more than 70 channel partners spanning 27 cities across India. Red Hat India's key channel partners include Efensys Technologies, Embee Software, Allied Digital Services, and Softcell Technologies. Key distributors listed by it include Ingram Micro, GT Enterprises, Sonata Software and Integra Microsystems.

Red Hat Learning Services, says the company, are available with "over 70 training partners at over 300 locations across India."

1.8)Profile of the student and Functions carried out :

The profile of the internship was basically software development and Project management. For successful implementation and zero defect delivery of this project it is essential to analyse the entire source code of the existing open source language recognition engine Tomoe. The entire project needs a very clear understanding of the following technologies and languages: C, C++, XML, Ruby on Rails and Fedora 10 is the platform. The assignment required a thorough knowledge of IT project management skills, Software development methodologies and Appropriate Software Engineering practices.

Initially the assignment required a study of the source code for the entire Tomoe application. Since this project was basically of an enhancement category, hence a comprehensive understanding of the existing application is very important. Once the flow of code was understood completely, the changes were decided upon to meet the project objectives. This entire activity set can be called analysis phase. The next phase was to

decide the design and layout of the new system. Next phase involved development and all stages had inherent testing methods to help filter out bugs.

So in essence analysis, design, coding and testing cycle was followed for the entire project. The task required appropriate version control and Project management methodologies.

CHAPTER 2
ANALYSIS PHASE FOR THE PROJECT

2.1)A step-by-step manner the conduct of the project

To begin with, Installation of all the necessary software like Fedora 10 operating system and Tomoe application on Fedora is done, so that the environment for development is set. After this begins the analysis phase , followed by the design and coding phases. The details of these phases are mentioned ahead.

The working environment at Red Hat module under which I was working facilitates agile methodology for software development, i.e working software is preferred over a comprehensive set of documentation (although proper documentation needs to be maintained) and Individuals and Interactions are preferred over following processes and tools.

2.1.1) Analysis Phase:- Tomoe is the application which would be upgraded to suit the Devanagari recognition.

About Tomoe:- Tomoe - Tegaki Online MOji recognition Engine - is a software which provides a handwriting recognition engine and its user interface on open source desktop environment. Here, "Tegaki" means "Handwriting" and "Moji" means "Letter" in Japanese.

2.1.2)Tomoe Source Analysis: (Refer to Appendix section for source)

Dictionary and Recognizer module load

File Name: **tomoe.c (Appendix 1)**
Location: **\tomoe-0.5.1\lib**
Contains: **tomoe_dict_load()**
And tomoe_recognizer_load()

tomoe_dict_load()

File Name: **tomoe-dict.c**

Location: **\tomoe-0.5.1\lib**

Contains: **code for retrieving the dict object**

tomoe_recognizer_load()

File Name: **tomoe-recognizer.c**

Location: **\tomoe-0.5.1\lib**

Contains: **code for retrieving the recognizer object**

ContextObject Creation

File Name: **tomoe-context.c (Appendix 2)**

Location: **\tomoe-0.5.1\lib**

Contains:
tomoe_context_search_by_strokes()

TomoeContext object contains the following values/Information:-

TomoeShelf *shelf;
TomoeRecognizer *recognizer;
TomoeDict *user_dict;

tomoe_context_search_by_stroke

File Name: **tomoe-context.c**

Location: **\tomoe-0.5.1\lib**

Contains: **Requires two parameters i.e TomoeContext & TomoeWriting objects**

TomoeWriting Object is found in \tomoe-0.5.1\lib\tomoe-writing.c. Inside this file there is a function call tomoe_writing_new(),which further calls g_object_new() which creates the new tomoe writing object.

TomoeContext object is found in \tomoe-0.5.1\lib\tomoe-context.c

tomoe_context_search_by_strokes()

File Name: **tomoe-context.c**

Location: **\tomoe-0.5.1\lib**

Contains: **g_list_sort (tomoe_recognizer_search (recognizer object,input),_candidate_compare_func)**

Now the tomoe_recognizer_search() function is present in \tomoe-0.5.1\lib\tomoe-recognizer.c. This file contains tomoe_recognizer_search().

The function tomoe_recognizer_search() contains a statement:- klass->search(recognizer object,input),where klass is a recognizer class object.

klass->search(recognizer object,input)

File Name: **tomoe-recognizer-simple.c (Appendix 3)**

Location: **\tomoe-0.5.1\module**

Contains: **_tomoe_recognizer_simple_get_candidates()**

This function is called from \tomoe-0.5.1\module \tomoe-recognizer-simple-logic.c.

The following are the parameters required by this function:- TomoeRecognizer *recognizer, TomoeDict *dict, TomoeWriting *input .

The return value of this function is a set of candidates which match the Input writing object . The return type is GList.

2.1.3)Design Phase:

From the above mentioned analysis phase , it was clear that the dictionary being used in the application was an XML dictionary.

An XML database is a data persistence software system that allows data to be stored in XML format. This data can then be queried, exported and serialized into the desired format.

Two major classes of XML database exist:

a) XML-enabled: these map all XML to a traditional database (such as a relational database), accepting XML as input and rendering XML as output. This term implies that the database does the conversion itself (as opposed to relying on middleware).

b) Native XML (NXD): the internal model of such databases depends on XML and uses XML documents as the fundamental unit of storage, which are, however, not necessarily stored in the form of text files.

Our project uses a Native XML (NXD) model. XML has several advantages for tomoe: it is human-readable, it is easy to read and generate with a machine, it can be used as a pivot format to generate files in another format. However, this is well-known that storage and processing costs are two disadvantages of XML.

In order to upgrade this application, for making it recognize devanagari script, it was essential to find the from which function the XML for the existing Japanese characters is being loaded.

This XML is being set as the dictionary for the recognizer in the Module package . Inside the module package there is a C file called tomoe-recognizer-simple.c which sets the XML file as the dictionary for the recognizer.

The first implementation step was to prepare the XML dictionary for devanagari characters . This was done using a tool called stroke editor. Then this devanagari XML file was copied in the Data package of the tomoe-0.5.1 package. This same XML dictionary was also placed in the usr/share/data/recognizer directory of the Fedora 10 O/S.

This XML replacement worked out successfully and the first draft of the recognition tool was ready and a demo was given by me at Red Hat. Upon successfully implementing this I was given two new requirements **:-**

a) To modify the frontend of the application developed in a way to facilitate the choice of dictionary (of a particular language)by the end user.

b) To improve the recognition algorithm for the existing application, to suit the devanagari recognition in a better way.

The design of the frontend layout for the new requirement has been prepared, which was approved by my mentor at Red Hat. This design includes providing user an option of selecting a particular language. Upon selecting the language ,the XML dictionary corresponding to the language will be set as the dictionary for the recognizer.

2.2) The technology/tools used

The entire project needs a very clear understanding of the following technologies and languages**: C, C++, XML, Ruby on Rails and Fedora 10** is the platform.

The specific tools required for the successful implementation of this project are:

2.2.1)Tomoe application: Tomoe is a handwriting recognition library for Chinese and Japanese characters (*han* character, *kanji*). It is published under the LGPL license, which means that it is free software that can be used by both proprietary and free applications (provided that you use it as a dynamic library).

The Tomoe project provides the library itself as well as a character stroke database, a handwriting recognition canvas for GTK applications, a character stroke data editor, and modules for common Unix/Linux input methods. Unfortunately, very little programming documentation is provided in Tomoe, and there are almost no source code comments. It can be time consuming to get started, especially if you want to use libtomoe directly from your own code without using the GUI components from Tomoe.

Tomoe takes advantage of the data structures and object oriented programming features provided by the glib library.

2.2.2) Stroke Editor: It is an application which can be used to edit the character dictionary for any script

2.2.3) Cell writer: Cell writer is an open source handwriting recognition tool.Handwriting recognition, like its cousins speech recognition and optical character recognition, is a domain still dominated by proprietary products. Where there are Linux solutions, such as the one in Nokia's Maemo Internet tablets, they are often closed source plugins protected by patent claims.

CellWriter works by learning the writing style of each individual user, so training is a must. If CellWriter doesn't find a ~/.cellwriter file at run time, it will open in training mode so it can learn your character style. If at any time you experience a lot of trouble with a particular character, you can click the Train button and provide more samples -- the app continues to learn from repeated experience.

Train mode sports a matrix of characters, each in its own cell. All you do is draw the appropriate character in its cell, and CellWriter commits it to memory. Trained and un-trained cells are marked in different colors so you can keep track of your work.

2.2.4) Concurrent Versions System

In the field of software development, the **Concurrent Versions System** (**CVS**), also known as the **Concurrent Versioning System**, is a free software revision control system. Version control system software keeps track of all work and all changes in a set of files, and allows several developers (potentially widely separated in space and/or time) to collaborate. Dick Grune developed CVS in the 1980s. CVS has become popular in the open source software world and is released under the GNU General Public License.

Terminology

CVS labels a single project (set of related files) modules it manages in its *repository*. Programmers acquire copies of modules by *checking out*. The checked-out files serve as a *working copy*, *sandbox* or *workspace*. Changes to the working copy will be reflected in the repository by *committing* them. To *update* is to acquire or *merge* the changes in the repository with the working copy.

Limitations

For each commonly listed limitation of CVS there exists a commonly listed reason. The developers of CVS insist that the following properties of CVS are not shortcomings, but features that were carefully planned, designed and implemented into CVS.

CVS does not version the moving or renaming of files and directories. It was implemented this way because in the past refactoring was avoided in development processes. More recently the thinking has changed and refactoring can be managed by an administrator (by directly moving the RCS file in the repository, provided that the administrator knows what he or she is doing) as it is required.

If you develop in Oracle Forms, Cobol, Fortran or even C++ then the CVS reasoning is quite commonly accepted ; if you develop with Java then the CVS reasoning may seem counterintuitive .

No versioning of symbolic links. Symbolic links stored in a version control system can pose a security risk - someone can create a symbolic link index.htm to /etc/passwd and then store it in the repository; when the "code" is exported to a Web server the Web site now has a copy of the system security file available for public inspection. A developer may prefer the convenience and accept the responsibility to decide what is safe to version and what is not; a project manager or auditor may prefer to reduce the risk by using build scripts that require certain privileges and conscious intervention to execute.

- Limited support for Unicode and non-ASCII filenames. Many Unix systems run in UTF-8 and so CVS on such systems handles UTF-8 filenames natively. For programmers working on Unix systems all with the same encoding then this response seems reasonable ; but it causes problems when multiple encodings are used, perhaps because clients are running another OS, such as AS/400 or Windows).

- No atomic commit. The network and server used should have sufficient resilience that a commit can complete without ever crashing. In many code management processes, development work is performed on branches (for example, add feature A1234), and then merged into the trunk after code review - that final merge is 'atomic' and performed in the data center by QA.

The term atomic is sometimes referred to in the transactional database sense where a commit will automatically roll back should it fail for any reason, and sometimes referred to in the sense that each commit can be uniquely identified. If each commit needs to be tracked then this can be handled by modifying the correct trigger.

- Expensive branch operations. CVS assumes that the majority of work will take place on the trunk — branches should generally be short-lived or historical. When used as designed, branches are easily managed and branch operations are efficient and fast.

- CVS treats files as textual by default. Text files should be the primary file type stored in the CVS repository. Binary files are supported and files with a particular file extension can automatically be recognised as being binary.

- No support for distributed revision control or unpublished changes. Programmers should commit changes to the files often for frequent merging and rapid publication to all users.

Over time, developers have wanted to change the CVS code significantly to add new features, refactor the code, alter the operational model and improve developers' productivity. This has led to the phrase YACC: "Yet Another CVS Clone" (itself a play on the Unix command named, VACC, which stands for "yet another compiler compiler").

CVS replacement projects include CVSNT (first released 1998), EVS (first released 2008), OpenCVS(not released as of 27 August 2008) and Subversion (initially released in 2004 and numerous systems to support distributed revision control.

Features

CVS uses a client-server architecture: a server stores the current version(s) of a project and its history, and clients connect to the server in order to "check out" a complete copy of the project, work on this copy and then later "check in" their changes.

Typically, the client and server connect over a LAN or over the Internet, but client and server may both run on the same machine if CVS has the task of keeping track of the version history of a project with only local developers. The server software normally runs on Unix (although at least the CVSNT server supports various flavors of Microsoft Windows and Linux), while CVS clients may run on any major operating-system platform.

Several developers may work on the same project concurrently, each one editing files within their own "working copy" of the project, and sending (or *checking in*) their modifications to the server. To avoid the possibility of people stepping on each other's toes, the server will only accept changes made to the most recent version of a file.

Developers are therefore expected to keep their working copy up-to-date by incorporating other people's changes on a regular basis. This task is mostly handled automatically by the CVS client, requiring manual intervention only when a *conflict* arises between a checked-in modification and the yet-unchecked local version of a file.

If the check in operation succeeds, then the version numbers of all files involved automatically increment, and the CVS-server writes a user-supplied description line, the date and the author's name to its log files. CVS can also run external, user-specified log processing scripts following each commit. These scripts are installed by an entry in CVS's loginfo file, which can trigger email notification or convert the log data into a Web-based format.

Clients can also compare versions, request a complete history of changes, or check out a historical snapshot of the project as of a given date or as of a revision number. Many open-source projects allow "anonymous read access", a feature pioneered byOPENBSD. This means that clients may check out and compare versions with either a blank or simple published password (e.g., "anoncvs"); only the check-in of changes requires a personal account and password in these scenarios.

Clients can also use the "update" command in order to bring their local copies up-to-date with the newest version on the server. This eliminates the need for repeated downloading of the whole project.

CVS can also maintain different "branches" of a project. For instance, a released version of the software project may form one branch, used for bug fixes, while a version under current development, with major changes and new features, can form a separate branch.

CVS uses delta compression for efficient storage of different versions of the same file. The implementation favors files with many lines (usually text files) - in extreme cases the system may store individual copies of each version rather than deltas.

2.3) pre-reading done by the student to supplement their understanding on the project

Some of the links which I found highly useful in executing this project are :

a) http://www.mblondel.org/journal/2007/07/25/%E3%81%84%E3%82%8D%E3%81%84%E3%82%8D/#more-68
b) http://www.linux.com/archive/feature/120867
c) http://anderssonj.com/
d) http://sourceforge.net

2.4) Procedures adopted for the Project:

The software development methodology adopted for this project execution was AGILE. The choice of the methodology has a major impact on the efficiency of the software application development process.

2.5) What is AGILE?

The Emergence of Agile Methods

2.5.1) Underlying beliefs:

- . Software development is predominantly a design activity
- . Characteristics of individuals are the first-order influence on project activities
- . Modern software development can't rely on
- individuals – it requires effective teams
- . Customers are unlikely to know what they want in
- detail before they start
- . Customers need to be able to change requirements
- without unreasonable penalties

- . The process needs to be flexible
- . Responsibility should be aligned with authority

The methodologies falling under the “agile” umbrella all address these issues in slightly different ways, however the agile methodologies do have common underlying values and principles. The Agile Alliance expressed the values in the Agile Manifesto.

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan
-

2.5.2) The 12 underlying principles for AGILE methodology

1)· Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

·2) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

·3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.

4) Business people and developers must work together daily throughout the project.

·**5)**Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

. ..The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7) Working software is the primary measure of progress.

8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9)Continuous attention to technical excellence and good design enhances agility.

10) Simplicity--the art of maximising the amount of work not done--is essential.

11) The best architectures, requirements, and designs emerge from self-organising teams.

12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly."

2.5.3) The Classification of AGILE methodology Adopted for the project execution

Scrum

Scrum, named for the scrum in Rugby, was initially developed by Ken Schwaber and Jeff Sutherland, with later collaborations with Mike Beedle. Scrum provides a project management framework that focuses development into 30-day Sprint cycles in which a specified set of Backlog features are delivered.

The core practice in Scrum is the use of daily 15-minute team meetings for coordination and integration. Scrum has been in use for nearly ten years and has been used to successfully deliver a wide range of products.

Scrum activities

Backlog :Manage / Update Backlog of Project requirements and priorities

Sprints : work unit that are required to achieve a requirement defined in the backlog. Time is predefined (30 days). During this time, no change is made to the requirements being worked on

Daily meetings

- Only 15 mins
- All team members attend
- Led by a ''scrum master''
- Three questions are asked
- What has been completed / achieved since the last meeting
- What difficulties are we facing ?
- What is the planned achievement before the next meeting

2.5.4) What are the benefits ?

- Uncover potential problems asap
- Knowledge sharing
- Promotes self organizing team

In scrum methodology regular demos are required for the software application being developed. These demos help in testing the application and detecting any anomalies and requirement mapping

Also its not mandatory to deliver a complete functionality in the case of SCRUM. Partially increments can also be submitted to the client.
The advantage of this is that the user can give early feedback and the entire process can be speeded up .

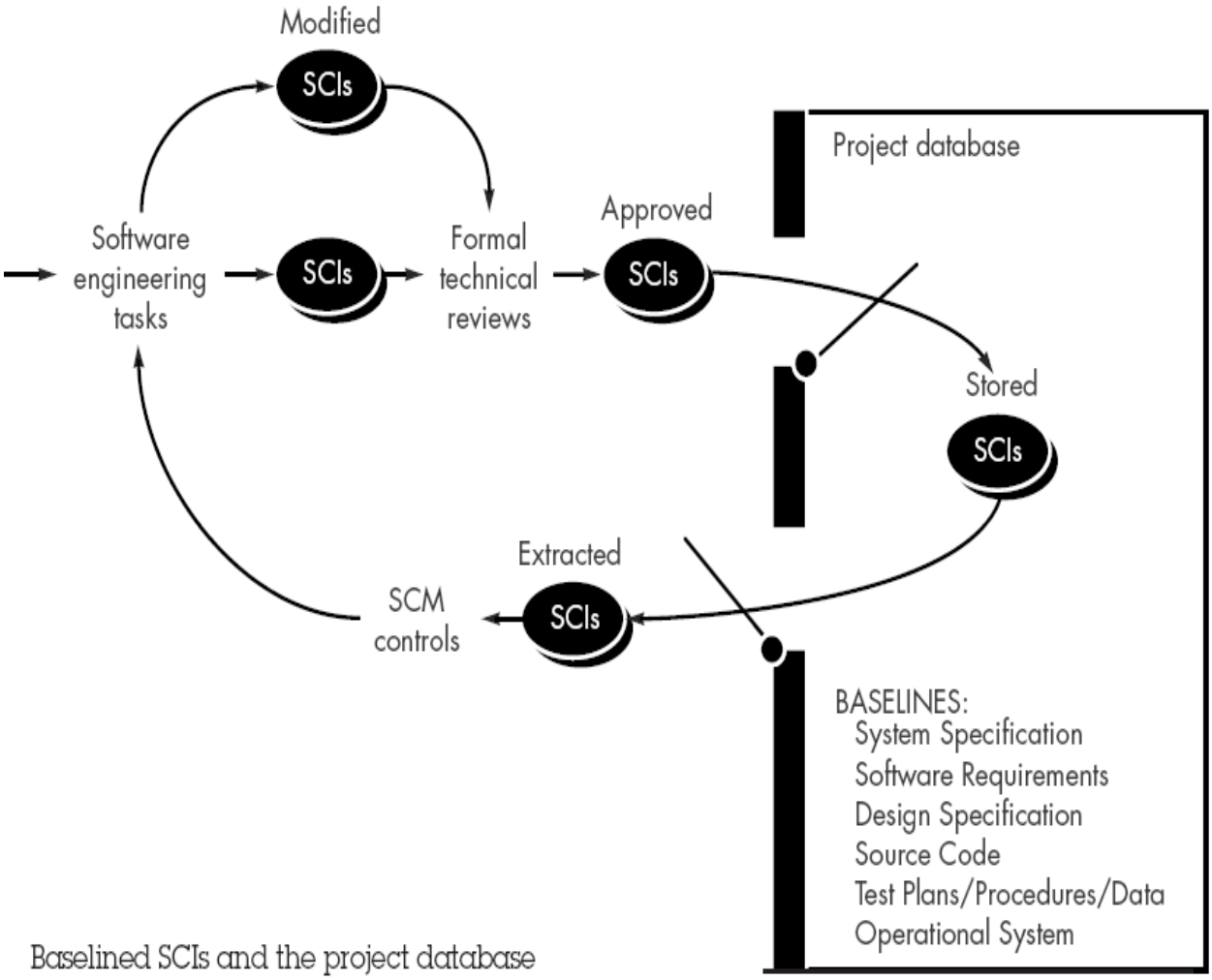
2.5.5) Scrum principles are consistent with the Agile manifesto

- Small working / self organizing teams
- Process very responsive to change
- Process yields frequent software increments that can be inspected, tested, documented and built on
- Development work and the people who perform it are partitioned into clean low coupling partitions
- Constant testing & documentation is performed as the product is built
- The process provides the ability to declare a product ''completed'' when required (instead of saying that it will be ready in 2099)

2.6) SOFTWARE CONFIGURATION MANAGEMENT

Software configuration management (SCM) is an umbrella activity that is applied throughout the software process. Because change can occur at any time, SCM activities are developed to (1) identify change, (2) control change, (3) ensure that change is being properly implemented, and (4) report changes
to others who may have an interest.

Baseline: A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures. In this project I chose Tomoe 0.6.1 as the baseline for further development.



Baselined SCIs and the project database

Figure 1

In the context of software engineering, a baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical

review.

For example, the elements of a *Design Specification* have been documented and reviewed. Errors are found and corrected. Once all parts of the specification have been reviewed, corrected and then approved, the *Design Specification* becomes a baseline. Further changes to the program architecture (documented in the *Design Specification*) can be made only after each has been evaluated and approved.

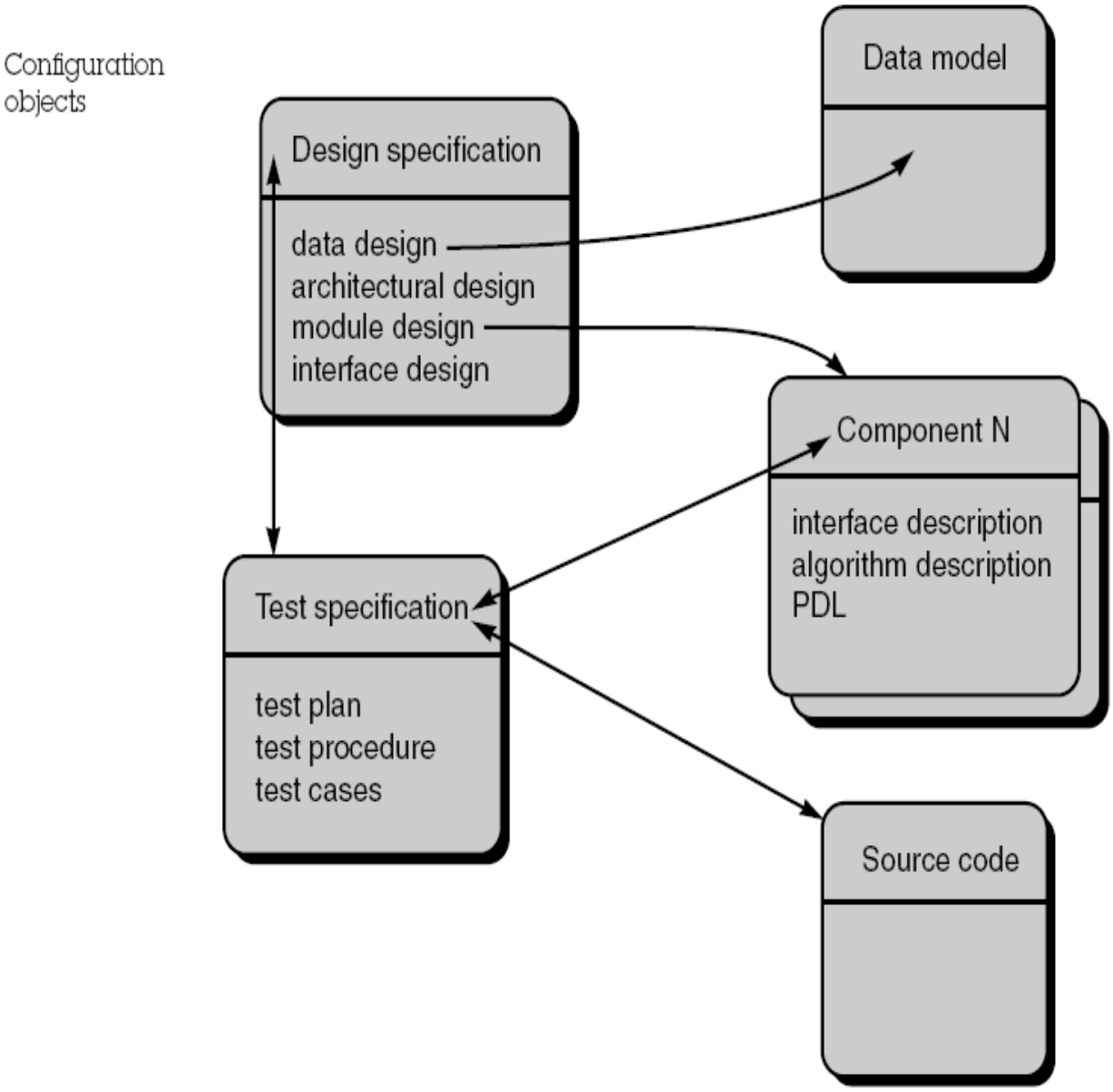


Figure 2

2.7) CHANGE CONTROL :

The change control process

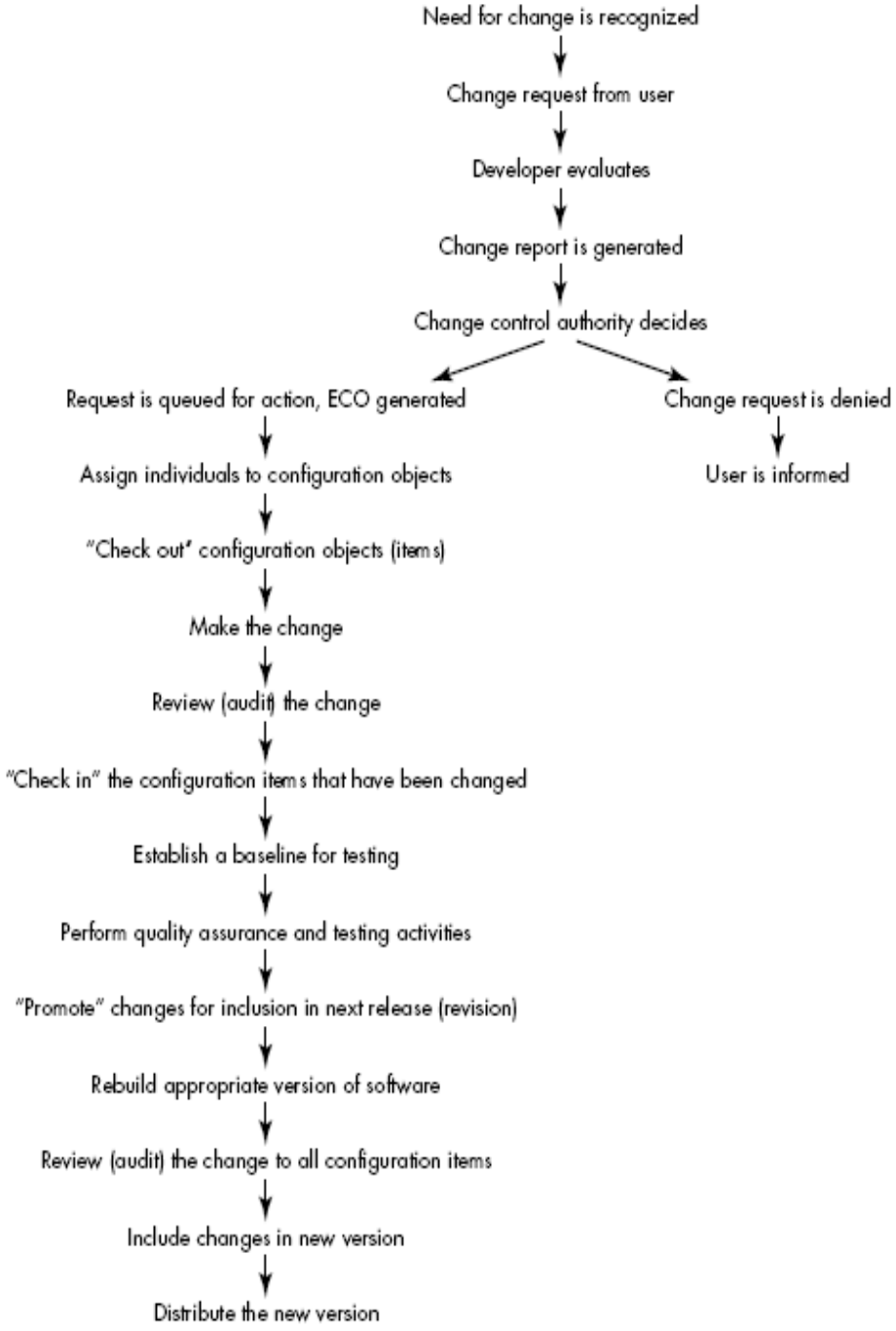


Figure 3

2.8) REQUIREMENT ENGINEERING:

The outcome of the system engineering process is the specification of a computer based system or product at the different levels. But the challenge facing system engineers (and software engineers) is profound: How can we ensure that we have specified a system that properly meets the customer's needs and satisfies the customer's expectations?

There is no foolproof answer to this difficult question, but a solid requirements engineering process is the best solution we currently have. Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system. The requirements engineering process can be described in five distinct steps:-

2.8.1) Requirements Elicitation:

• Assess the business and technical feasibility for the proposed system.

• Identify the people who will help specify requirements and understand their organizational bias.

• Define the technical environment (e.g., computing architecture, operating system, telecommunications needs) into which the system or product will be placed.
• Identify "domain constraints" (i.e., characteristics of the business environment specific to the application domain) that limit the functionality or performance of the system or product to be built.

• Define one or more requirements elicitation methods (e.g., interviews, focus groups, team meetings). In our project, it was team meeting.

• Solicit participation from many people so that requirements are defined from different points of view; be sure to identify the rationale for each requirement that is recorded.

• Identify ambiguous requirements as candidates for prototyping.

• Create usage scenarios to help customers/users better identify key requirements.

2.8.2) Requirements Analysis and Negotiation:

As the requirements analysis activity commences, the following questions are asked and answered:

• Is each requirement consistent with the overall objective for the system/product?

• Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?

• Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?

• Is each requirement bounded and unambiguous?

• Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?

• Do any requirements conflict with other requirements?

• Is each requirement achievable in the technical environment that will house the system or product?

• Is each requirement testable, once implemented?

2.8.3) requirements specification:

The System Specification is the final work product produced by the system and requirements engineer. It serves as the foundation for hardware engineering, software engineering, database engineering, and human engineering.

It describes the function and performance of a computer-based system and the constraints that will govern its development. The specification bounds each allocated system element. The System Specification also describes the information (data and control) that is input to and output from the system.

2.8.4) System Modeling:

It is important to evaluate the system's components in relationship to one another, to determine how requirements fit into this
picture, and to assess the "aesthetics" of the system as it has been conceived.

2.8.5) requirements validation:

The work products produced as a consequence of requirements engineering (a system specification and related information) are assessed for quality during a validation step. *Requirements validation* examines the specification to ensure that all system requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.

The primary requirements validation mechanism is the formal technical review .The review team includes system engineers, customers, users, and other stakeholders who examine the system specification5 looking for errors in content or

interpretation, areas where clarification may be required, missing information, inconsistencies
(a major problem when large products or systems are engineered), conflicting requirements, or unrealistic (unachievable) requirements.

Although the requirements validation review can be conducted in any manner that results in the discovery of requirements errors, it is useful to examine each requirement against a set of checklist questions. The following questions are usually asked in this process:-

• Are requirements stated clearly? Can they be misinterpreted?

• Is the source (e.g., a person, a regulation, a document) of the requirement identified? Has the final statement of the requirement been examined by or against the original source?

• Is the requirement bounded in quantitative terms?

• What other requirements relate to this requirement? Are they clearly noted via a cross-reference matrix or other mechanism?

• Does the requirement violate any domain constraints?

• Is the requirement testable? If so, can we specify tests (sometimes called *validation criteria*) to exercise the requirement?

• Is the requirement traceable to any system model that has been created?

• Is the requirement traceable to overall system/product objectives?

• Is the system specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?

• Has an index for the specification been created?

• Have requirements associated with system performance, behavior, and operational characteristics been clearly stated? What requirements appear to be implicit?

2.8.6) Requirements Management:

Requirements management is a set of activities that help the project team to identify, control, and track requirements and changes to requirements at any time as the project proceeds. Many of these activities are identical to the software configuration management techniques

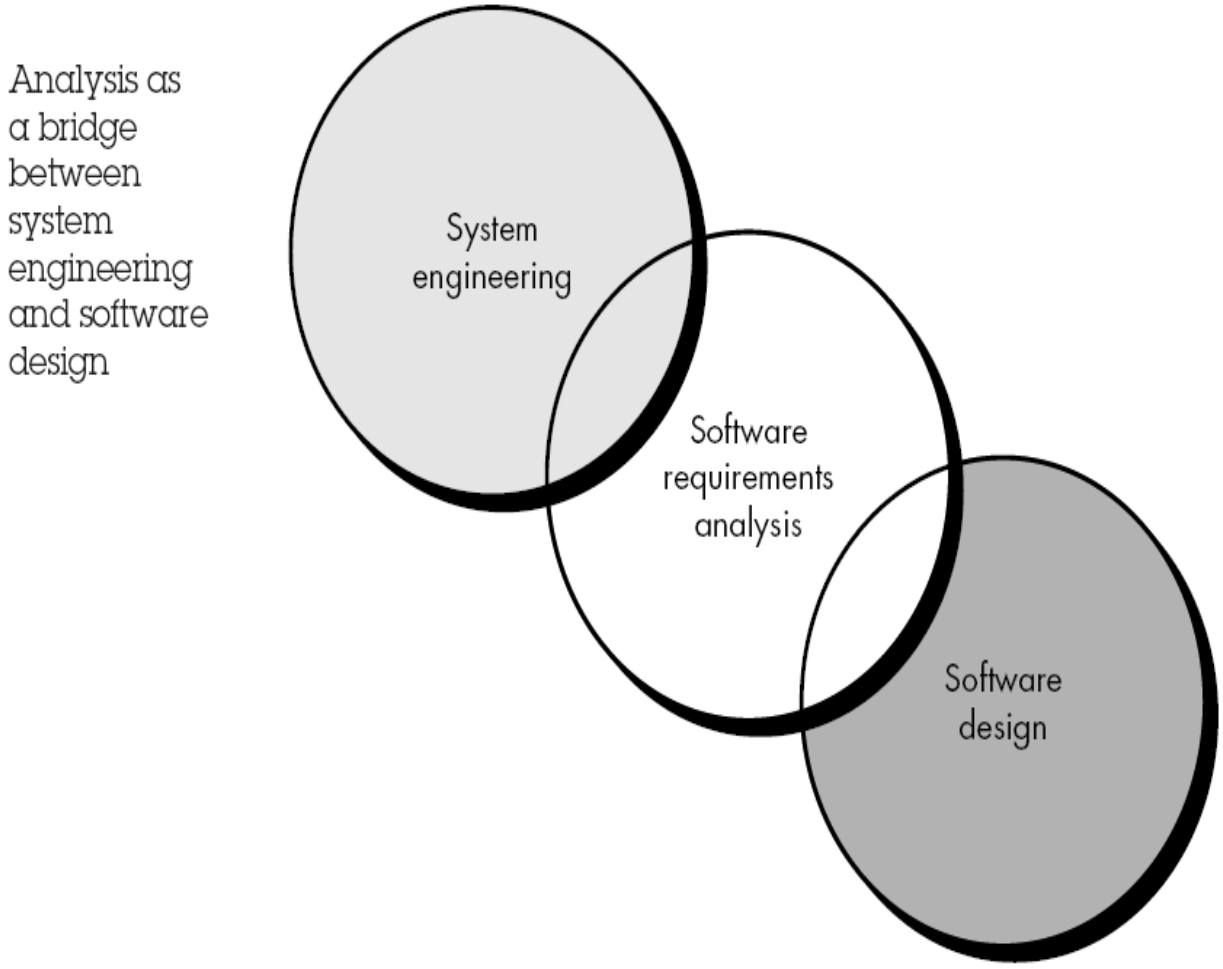


Figure 4

2.9) DESIGN PROCESS:

Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software. Initially, the blueprint depicts a holistic view of software. That is, the design is represented at a high level of abstraction—a level that can be directly traced to the specific system objective and more detailed

data, functional, and behavioral requirements.

As design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction. These can still be traced to requirements, but the connection is more subtle.

Three characteristics that serve as a guide for the evaluation of a good design:

• The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.

• The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.

• The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Each of these characteristics is actually a goal of the design process. But how is each of these goals achieved?

In order to evaluate the quality of a design representation, we must establish technical criteria for good design:

1. A design should exhibit an architectural structure that (1) has been created using recognizable design patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.

2. A design should be modular; that is, the software should be logically partitioned into elements that perform specific functions and subfunctions.

3. A design should contain distinct representations of data, architecture, interfaces, and components (modules).

4. A design should lead to data structures that are appropriate for the objects to be implemented and are drawn from recognizable data patterns.

5. A design should lead to components that exhibit independent functional characteristics.

6. A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.

7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

2.10) PROJECT MANAGEMENT PROCESS:

Project Management Process	Project Classification			
	A	B	C	D
Define				
Conditions of Satisfaction	R	R	O	O
Project Overview Statement	R	R	R	R
Approval of Request	R	R	R	R
Plan				
Conduct Planning Session	R	R	O	O
Prepare Project Proposal	R	R	R	R
Approval of Proposal	R	R	R	R
Launch				
Kick-off Meeting	R	R	O	O
Activity Schedule	R	R	R	R
Resource Assignments	R	R	R	O
Statements of Work	R	O	O	O
Monitor/Control				
Status Reporting	R	R	R	R
Project Team Meetings	R	R	O	O
Approval of Deliverables	R	R	R	R
Close				
Post-Implementation Audit	R	R	R	R
Project Notebook	R	R	O	O

R = Required O = Optional

The use of required and optional parts of the methodology by type of project.

Figure 5

2.11) Milestone review : **a) 6th April to 26th April:-** During this phase I successfully analyzed the source code of the tomoe application. The flow of function calls was prepared in this phase. The technologies involved were understood .(Primarily C and GTK)

b) 27th April to 15th May:- Once the analysis of the application was completed, a design of the new application was prepared. A checklist was prepared consisting of all the changes which are to be propagated to the existing system.

c) 17th May to 20th July :- The necessary source code and API's are being developed to fulfill the two requirements given after the completion of the first draft of the application. This phase also includes manually testing the source code being developed. It also includes research done to find ways to improve the efficiency of the recognition in the Tomoe application.

2.12) APPLICATION SCREENSHOTS:

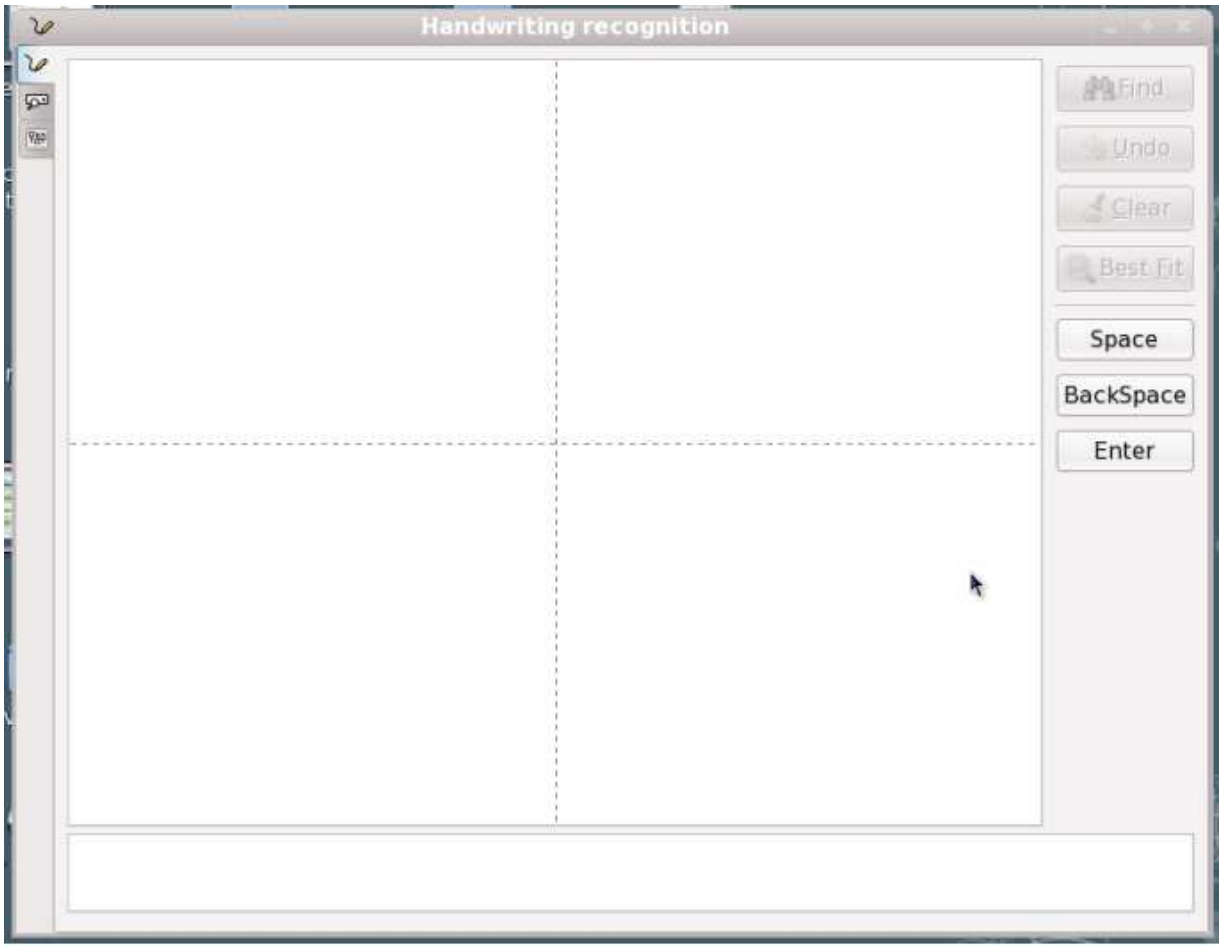


Figure 7

SCREENSHOT 1: The initial loaded User Interface Layout of the Tomoe Application

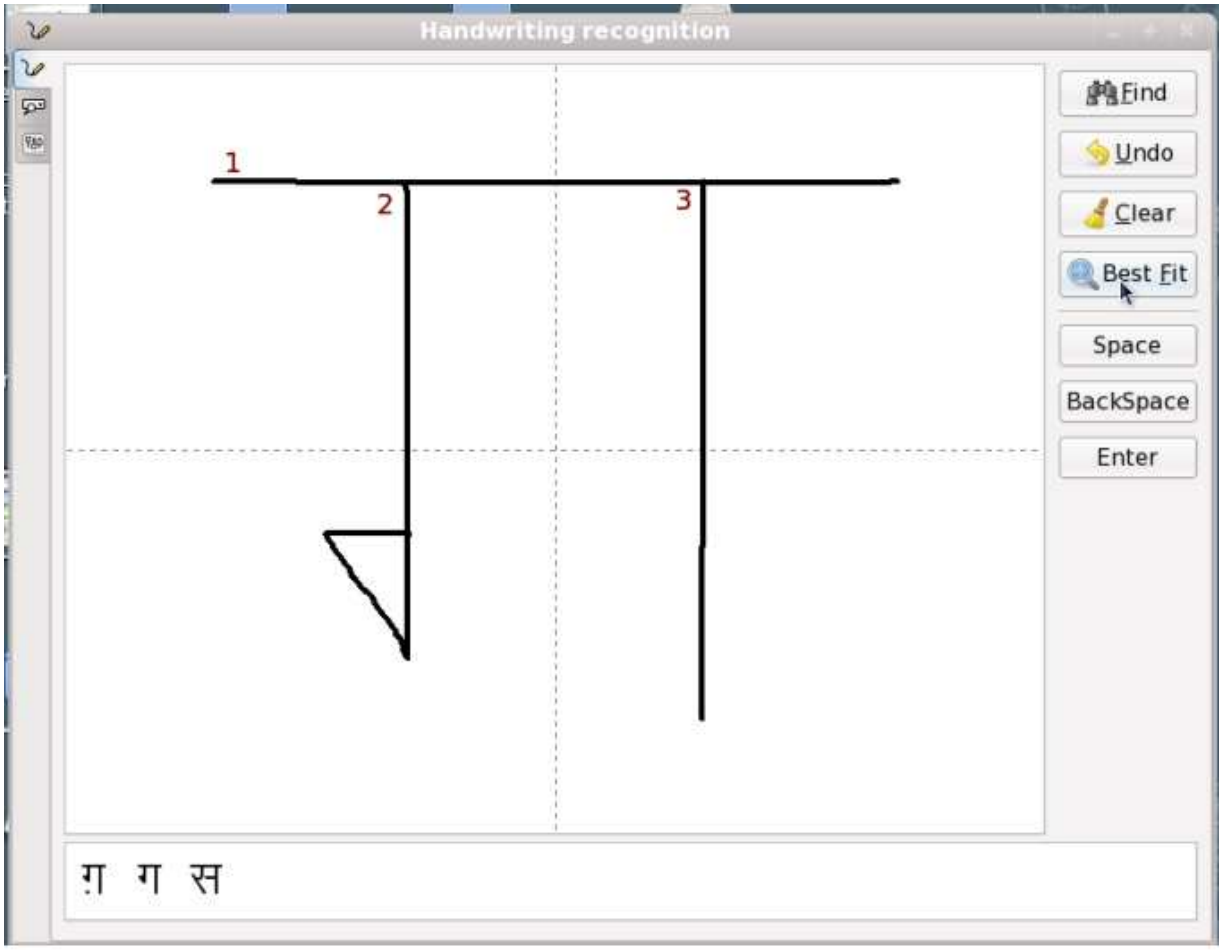


Figure 8

SCREENSHOT 2: The character drawn on the Canvas of Tomoe and 3 corresponding candidates being shown in the Tab below. The user can click on any of the three candidates shown and get it typed on the Linux terminal as well as on any text file opened.

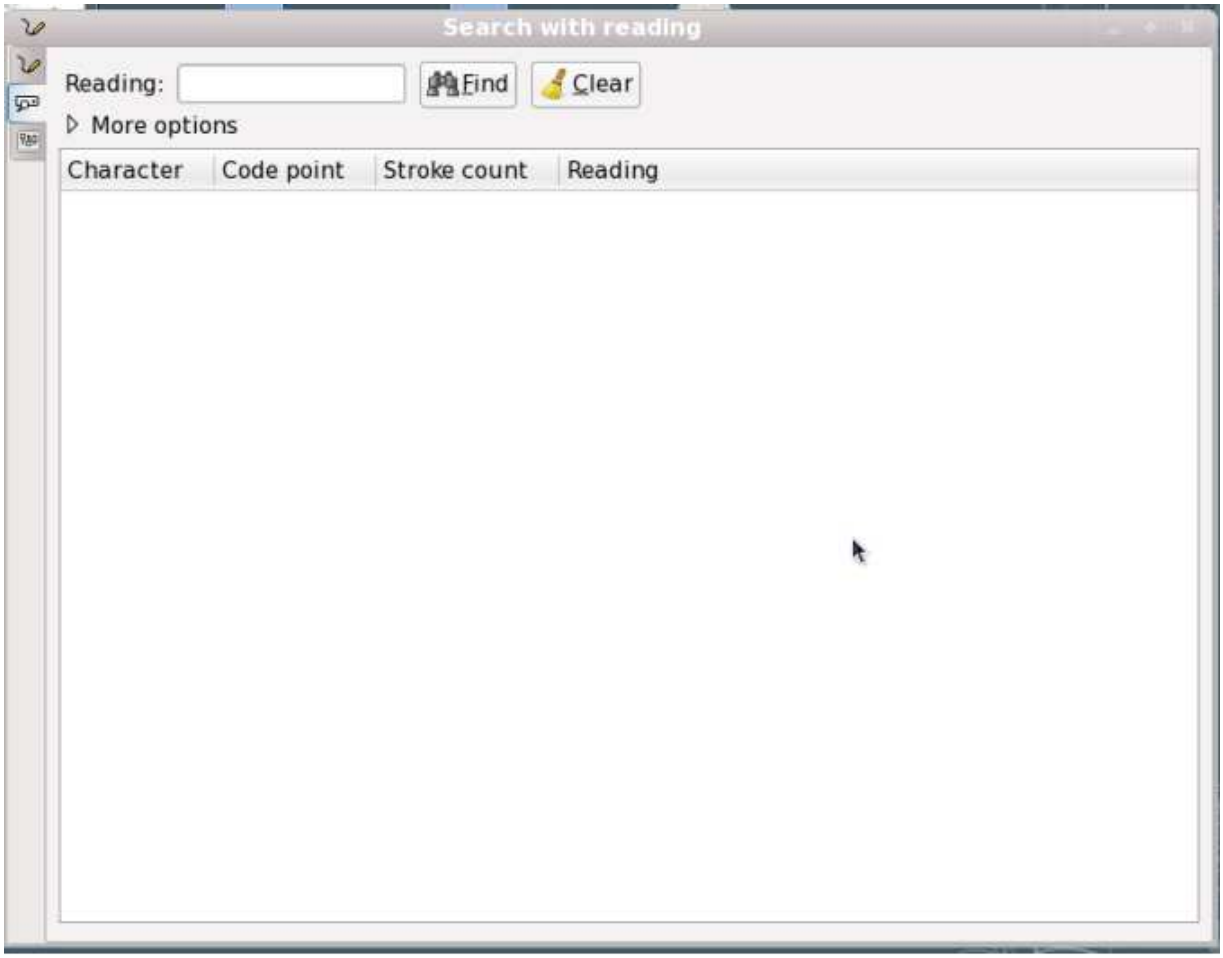


Figure 9

SCREENSHOT 3

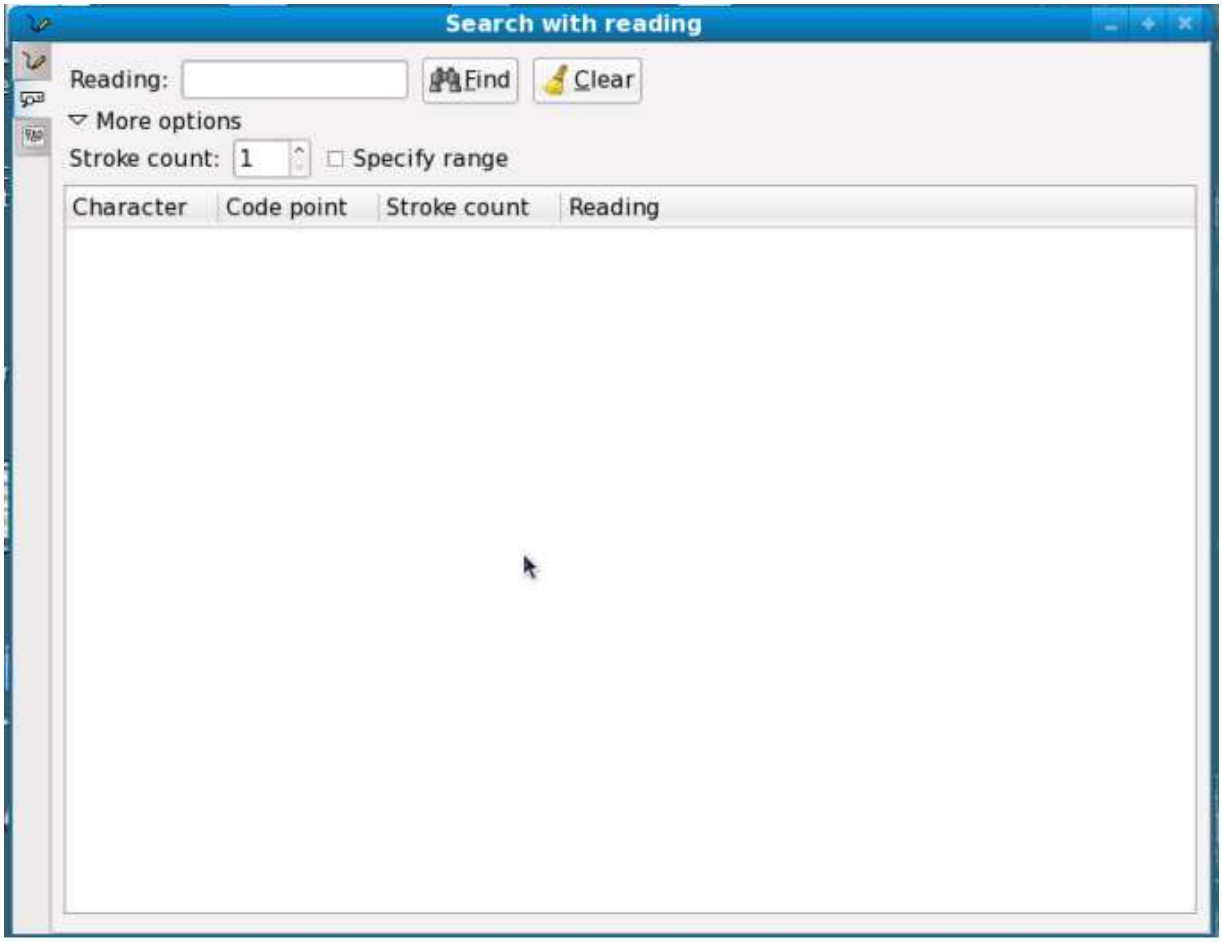


Figure 10

SCREENSHOT 4

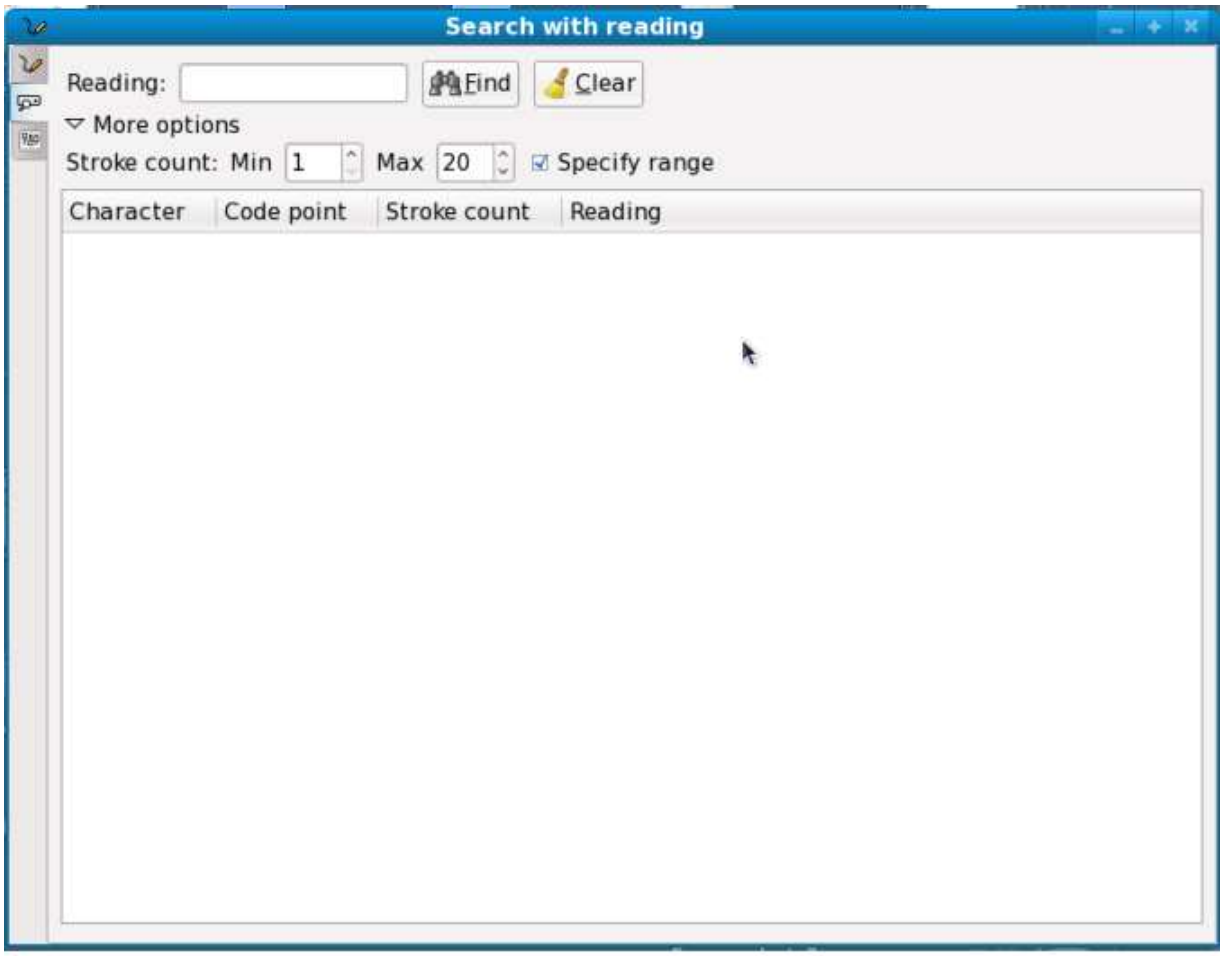

Search with reading
Reading:
Find
Clear
More options
Stroke count: Min 1 Max 20 Specify range
Character
Code point
Stroke count
Reading

Figure 11

SCREENSHOT 5

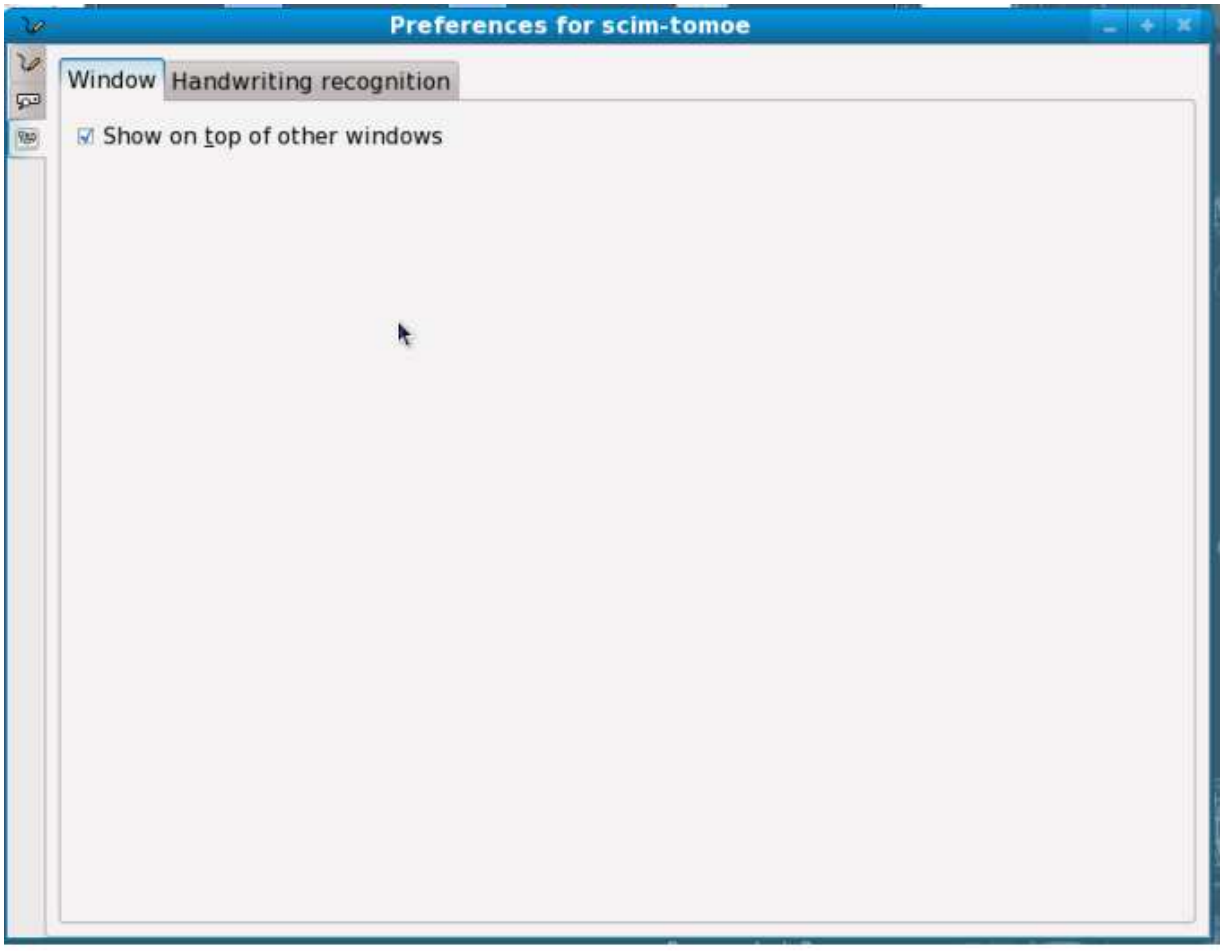


Figure 12

SCREENSHOT 6

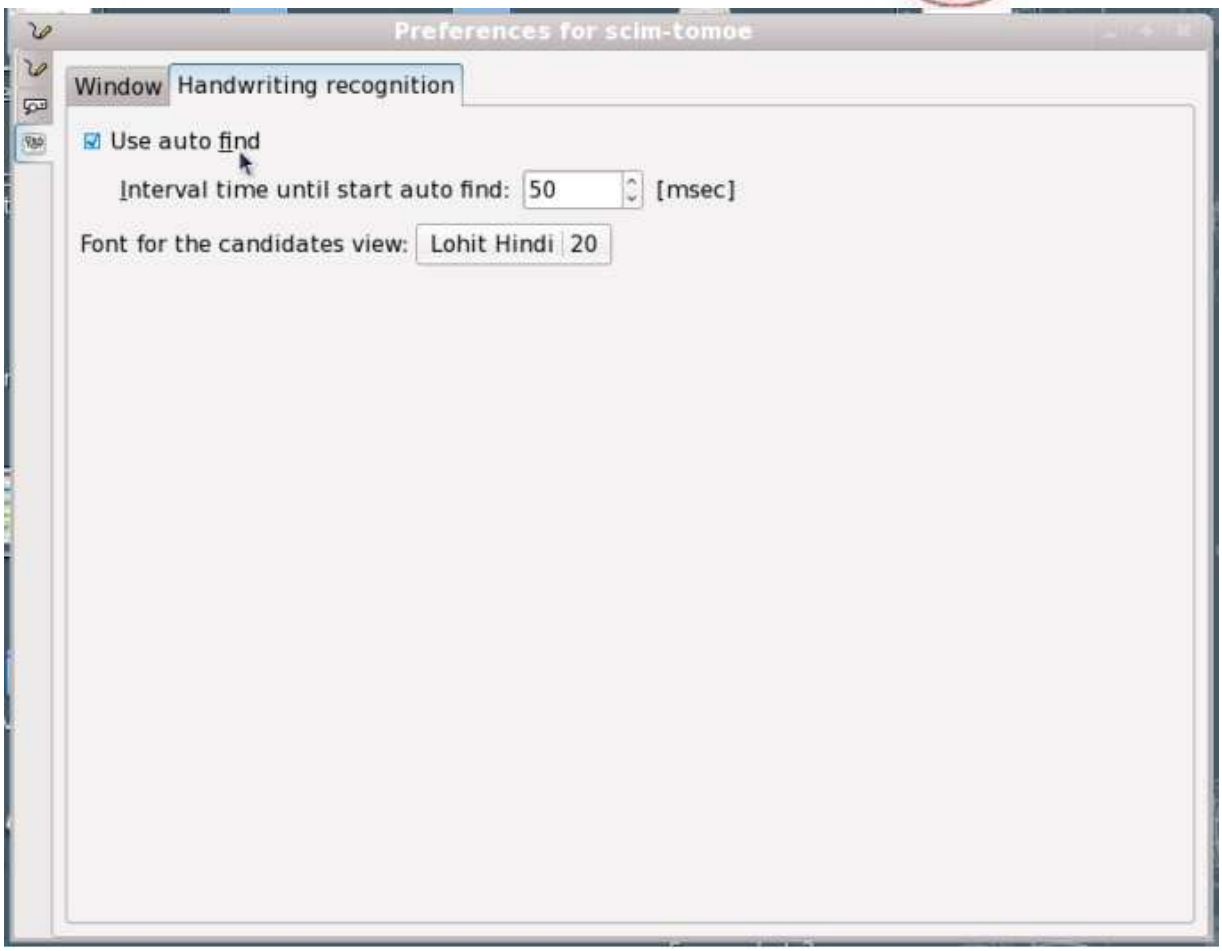


Figure 13

SCREENSHOT 7

CHAPTER 3

Learning experiences on Business / Technology

3.1) Technical or Business Research Problem identified

3.1.1) To identify how to create the XML dictionary for devanagari script

3.1.2) To understand the correlation between recognition efficiency and XML source code .

3.1.3) To come up with recognition algorithm changes ,better suited to devanagari recognition

3.1.4) To come up with the suitable design for incorporating the frontend changes required.

3.2) Approaches to the above problem :

3.2.1) I chose a tool called stroke editor, which is made by Red Hat engineers. By the help of this tool I successfully created an appropriate dictionary for devanagari script.

3.2.2) By analysis of the character recognition, I concluded that for some specific characters in devanagari script the recognition was not very efficient. This was true mostly for such characters which were curvy in nature. The recognition for characters with straight lines and less curves was more efficient.

3.2.3) This requirement is yet to be designed and coded. I plan to incorporate this algorithmic change in the second draft of the application.

3.2.4) The layout for the frontend changes were prepared and approved by Red Hat .

The following were my salient learnings from this project:-

3.2.5) Understood the working of a handwriting recognition system. These concepts are same for any type of pattern recognition, hence this project gave me insight about the functionality of generic recognition systems also. Since I was given the sole

responsibility for the entire project implementation , I got an opportunity to go through the full development life cycle of the software application.

3.2.6) The usage of XML as a dictionary was clearly understood by me. My skill sets with Technologies like XML, C, C++ and Ruby on rails have definitely increased after successful completing the first draft of the project for Red Hat.

3.2.7) Working for Red Hat has helped me understand the open source platforms and technologies. Previously I had experience of only working with proprietary technologies. I gained experience of using open source platform like Fedora 10 and developing customized IT applications on this platform.

3.2.8) The work culture of Red Hat is very different and exciting, as it is a software Product company. It was a great learning experience as I saw small teams handling large and complicated projects efficiently. My mentors at Red Hat were very helpful in every phase of this project implementation. I got a chance to interact with some of the very talented people in IT product development field.

3.2.9) This project has reinforced my learnings on IT project management, Requirements Engineering and software development methodologies. The concepts of version control/configuration management, requirements gathering, effective client communication(as Red Hat was my client).

3.2.10) From delivery point of view I have already developed a working model of the application which was required in the first set of requirements. I am working on the new set of requirements allocated to me, and hope to implement them also successfully before the internship deadline.

3.2.11) By the completion of this project I expect to make this tomoe application support multiple language recognition. The architecture of the application would be changed and rather than supporting only a couple of languages the application design would support multiple scripts. Later on apart from Chinese simplified, Japanese and Devanagari scripts other languages like Marathi, Malayalam , Kannada ,Telugu etc can be successfully identified via this tomoe application.

3.2.12) I am currently working on improving the recognition efficiency of current version of tomoe application. During this phase I am learning how to incorporate the concept of multi level pattern matching in an application using C and XML.

CHAPTER 4

Conclusion

4.1) Contribution:

4.1.1) I have successfully modified the Tomoe recognition engine for the recognition of Devanagari script, which was the objective of the assignment. The application now has the ability to recognize devanagari characters.

4.1.2) The first Implementation step was to prepare the XML dictionary for Devanagari characters. This was done using a tool called Stroke editor. Then this devanagari XML file was copied in the Data package of the tomoe-0.5.1 package. This same XML dictionary was also placed in the usr/share/data/recognizer directory of the Fedora 10 O/S. This XML replacement worked out successfully and the first draft of the recognition tool was ready and a demo was given by me at Red Hat.

4.1.3) For improving the recognition of the tomoe application, I prepared a second XML dictionary. The stroke counts and the coordinates corresponding to each stroke where intentionally kept different from those which are present in the first XML. This was done to increase the probability of a successful match of a candidate.

I also formulated a design to Integrate this newly added layer of dictionary in the Existing version of tomoe application. This required an Indepth analysis of the existing recognizer code, Understanding how the XML parsing is linked to the Recognizer module in the application. Once this XML Linkage was understood, the necessary code changes were made to include a multi level pattern matching in the application.

4.1.4) I was also given the task by my mentor at Red Hat to prepare a completely technical document which includes all the coding changes which I have done till date to build this application successfully. This document should also contain an entire Explanation of the internal working of tomoe engine.

I have successfully prepared this documentation and delivered it on schedule. This document would act as a great help when in the future some open source developer tries to upgrade my version of tomoe.

4.1.5) Gave 3 alternate approaches to improve the recognition algorithm for Tomoe to Red Hat engineering team. Two have already been mentioned earlier. The third one was by implementing the **Hidden Markow model algorithm.**

A **hidden Markov model** (**HMM**) is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved state. An HMM can be considered as the simplest dynamic Bayesian network.

In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but output dependent on the state is visible. Each state has a probability distribution over the possible output tokens.

Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Note that the adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of the model; Even if the model parameters are known exactly, the model is still 'hidden'.

Hidden Markov models are especially known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

CHAPTER 5

APPENDIX

Appendix 1: tomoe.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib/garray.h>
#include "tomoe.h"

static gboolean initialized = FALSE;

/**
 * tomoe_init:
 *
 * Initialize tomoe library.
 *
 */
void
tomoe_init (void)
{
    if (!initialized) {
        initialized = TRUE;
        GTypeDebugFlags debug_flag = G_TYPE_DEBUG_NONE;
        g_type_init_with_debug_flags (debug_flag);
        tomoe_dict_load (NULL);
        tomoe_recognizer_load (NULL);
    }
}
```

```
/**
 * tomoe_quit:
 *
 * Terminate tomoe library.
 *
 */
void
tomoe_quit (void)
{
    if (initialized) {
        initialized = FALSE;
        tomoe_dict_unload ();
        tomoe_recognizer_unload ();
    }
          }
```

Appendix 2: tomoe-context.c

```
#include <stdio.h>
#include <errno.h>
#include <string.h>

#include "tomoe-dict.h"
#include "tomoe-recognizer.h"
#include "tomoe-context.h"
#include "tomoe-config.h"
#include "tomoe-shelf.h"
#include "tomoe-candidate.h"
#include "glib-utils.h"

#define DEFAULT_USER_DICT_CONTENT
          \
"<?xml version =\"1.0\" encoding=\"UTF-8\"?>\n"
          \
"<!DOCTYPE dictionary SYSTEM \"" DATADIR "/dict.dtd\">\n"
     \
"<dictionary name=\"User dictionary\">\n"
          \
"</dictionary>\n"

#define TOMOE_CONTEXT_GET_PRIVATE(obj)
(G_TYPE_INSTANCE_GET_PRIVATE ((obj), TOMOE_TYPE_CONTEXT,
TomoeContextPrivate))
```

```
enum {
    PROP_0,
    PROP_RECOGNIZER
};

typedef struct _TomoeContextPrivate     TomoeContextPrivate;
struct _TomoeContextPrivate
{
    TomoeShelf *shelf;
    TomoeRecognizer *recognizer;
    TomoeDict *user_dict;
};

G_DEFINE_TYPE (TomoeContext, tomoe_context, G_TYPE_OBJECT)

static void         dispose                  (GObject
*object);
static void         set_property             (GObject
*object,
                                              guint
prop_id,
                                              const GValue
*value,
                                              GParamSpec
*pspec);
static void         get_property             (GObject
*object,
                                              guint
prop_id,
                                              GValue
*value,
                                              GParamSpec
*pspec);

static void
tomoe_context_class_init (TomoeContextClass *klass)
{
    GObjectClass *gobject_class;

    gobject_class = G_OBJECT_CLASS (klass);

    gobject_class->dispose      = dispose;
    gobject_class->set_property = set_property;
    gobject_class->get_property = get_property;
```


```
    g_object_class_install_property (
        gobject_class,
        PROP_RECOGNIZER,
        g_param_spec_object (
            "recognizer",
            "Recognizer",
            "The recognizer of the context",
            TOMOE_TYPE_RECOGNIZER,
            G_PARAM_READWRITE | G_PARAM_CONSTRUCT_ONLY));

    g_type_class_add_private (gobject_class, sizeof
(TomoeContextPrivate));
}

static void
tomoe_context_init (TomoeContext *context)
{
    TomoeContextPrivate *priv = TOMOE_CONTEXT_GET_PRIVATE
(context);

    priv->shelf      = NULL;
    priv->recognizer = NULL;
    priv->user_dict  = NULL;
}

/**
 * tomoe_context_new:
 *
 * Create a new #TomoeContext.
 *
 * Return value: a new #TomoeContext.
 */
TomoeContext*
tomoe_context_new(void)
{
    TomoeContext *context;

    context = g_object_new(TOMOE_TYPE_CONTEXT, NULL);

    return context;
}

static void
set_property (GObject *object,
```


```
              guint prop_id,
              const GValue *value,
              GParamSpec *pspec)
{
    TomoeContextPrivate *priv = TOMOE_CONTEXT_GET_PRIVATE
(object);

    switch (prop_id) {
      case PROP_RECOGNIZER:
        if (priv->recognizer)
            g_object_unref (priv->recognizer);
        priv->recognizer = g_value_get_object (value);
        if (priv->recognizer)
            g_object_ref (priv->recognizer);
        break;
      default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
pspec);
        break;
    }
}


static void
get_property (GObject *object,
              guint prop_id,
              GValue *value,
              GParamSpec *pspec)
{
    TomoeContextPrivate *priv = TOMOE_CONTEXT_GET_PRIVATE
(object);

    switch (prop_id) {
      case PROP_RECOGNIZER:
        g_value_set_object (value, priv->recognizer);
        break;
      default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
pspec);
        break;
    }
}

static void
dispose (GObject *object)
```


```
{
    TomoeContextPrivate *priv = TOMOE_CONTEXT_GET_PRIVATE
(object);

    if (priv->shelf)
        g_object_unref (priv->shelf);
    if (priv->recognizer)
        g_object_unref (priv->recognizer);
    if (priv->user_dict)
        g_object_unref (priv->user_dict);

    priv->shelf      = NULL;
    priv->recognizer = NULL;
    priv->user_dict  = NULL;

    G_OBJECT_CLASS (tomoe_context_parent_class)->dispose
(object);
}

static gboolean
ensure_user_dict_file_content (const gchar *user_dict_filename)
{
    if (!g_file_test (user_dict_filename, G_FILE_TEST_EXISTS)) {
        const gchar *content;
        GError *error = NULL;

        content = DEFAULT_USER_DICT_CONTENT;
        if (!g_file_set_contents (user_dict_filename,
                                  content, strlen (content),
&error)) {
            TOMOE_HANDLE_ERROR (error);
            return FALSE;
        }
    }

    return TRUE;
}

static gchar *
ensure_user_dict_file (void)
{
    gchar *tomoe_dir_name, *user_dict_filename;

    tomoe_dir_name = g_build_filename (g_get_home_dir (),
".tomoe", NULL);
```


```
    if (!g_file_test (tomoe_dir_name, G_FILE_TEST_EXISTS)) {
        if (g_mkdir (tomoe_dir_name, 0700) == -1) {
            g_warning ("can't create %s: %s", tomoe_dir_name,
strerror (errno));
        }
    }

    if (!g_file_test (tomoe_dir_name, G_FILE_TEST_IS_DIR)) {
        g_warning ("%s isn't directory: %s", tomoe_dir_name,
strerror (errno));
    }

    user_dict_filename = g_build_filename (tomoe_dir_name,
"dict.xml", NULL);
    g_free (tomoe_dir_name);

    if (!ensure_user_dict_file_content (user_dict_filename)) {
        g_free (user_dict_filename);
        return NULL;
    }

    return user_dict_filename;
}

static TomoeDict *
ensure_user_dict (TomoeShelf *shelf, const gchar *name)
{
    TomoeDict *user_dict;

    g_return_val_if_fail (TOMOE_IS_SHELF (shelf), NULL);

    user_dict = tomoe_shelf_get_dict (shelf, name);
    if (user_dict) {
        g_object_ref (user_dict);
    } else {
        gchar *user_dict_filename;

        user_dict_filename = ensure_user_dict_file ();
        if (user_dict_filename) {
            user_dict = tomoe_dict_new ("xml",
                                        "filename",
user_dict_filename,
                                        "editable", TRUE,
                                        NULL);
```


```
            g_free (user_dict_filename);
        }

        tomoe_shelf_register_dict (shelf, name, user_dict);
    }

    return user_dict;
}

/**
 * tomoe_context_load_config:
 * @ctx: a #TomoeContext.
 * @config_file: the filename of configuration file to load.
 *
 * Load dictionaries into configuration file.
 */
void
tomoe_context_load_config (TomoeContext *ctx, const gchar
*config_file)
{
    TomoeContextPrivate *priv;
    TomoeConfig* cfg;

    g_return_if_fail (ctx);

    priv = TOMOE_CONTEXT_GET_PRIVATE(ctx);
    cfg = tomoe_config_new (config_file);

    if (priv->shelf)
        g_object_unref (priv->shelf);
    priv->shelf = tomoe_config_make_shelf (cfg);
    priv->user_dict = ensure_user_dict (priv->shelf,

tomoe_config_get_user_dict_name (cfg));

    g_object_unref (cfg);
}

static gint
_candidate_compare_func (gconstpointer a, gconstpointer b)
{
    TomoeCandidate *ca = *(TomoeCandidate **) a;
    TomoeCandidate *cb = *(TomoeCandidate **) b;
    return tomoe_candidate_compare (ca, cb);
}
```

```
static GList *
tomoe_context_search_by_strokes (TomoeContext *context,
TomoeWriting *input)
{
    TomoeContextPrivate *priv;
    GList *matched = NULL;

    g_return_val_if_fail (context, matched);
    if (!input) return matched;

    priv = TOMOE_CONTEXT_GET_PRIVATE (context);
    if (!priv->recognizer) {
        priv->recognizer = tomoe_recognizer_new ("simple",
NULL);
        g_return_val_if_fail (TOMOE_IS_RECOGNIZER (priv-
>recognizer), matched);
    }

    matched = g_list_sort (tomoe_recognizer_search (priv-
>recognizer, input),
                           _candidate_compare_func);

    return matched;
}

static GList *
tomoe_context_search_by_dict (TomoeContext *context, TomoeQuery
*query)
{
    TomoeContextPrivate *priv;
    TomoeShelf *shelf;
    GList *names, *name;
    GList *results = NULL;

    if (!context) return results;

    priv = TOMOE_CONTEXT_GET_PRIVATE (context);
    shelf = priv->shelf;
    if (!shelf) return results;

    names = tomoe_shelf_get_dict_names(shelf);
    if (!names) return results;
```


```
    for (name = names; name; name = name->next) {
        TomoeDict *dict;
        dict = tomoe_shelf_get_dict(shelf, name->data);
        results = g_list_concat (tomoe_dict_search (dict,
query), results);
    }
    results = g_list_sort (results, _candidate_compare_func);

    return results;
}

GList *
tomoe_context_search (TomoeContext *context, TomoeQuery *query)
{
    TomoeWriting *writing;

    writing = tomoe_query_get_writing (query);
    if (writing)
        return tomoe_context_search_by_strokes (context,
writing);
    else
        return tomoe_context_search_by_dict (context, query);
}

gboolean
tomoe_context_register (TomoeContext *context, TomoeChar *chr)
{
    TomoeContextPrivate *priv;

    g_return_val_if_fail (TOMOE_IS_CONTEXT (context), FALSE);

    priv = TOMOE_CONTEXT_GET_PRIVATE (context);
    if (!priv->user_dict) {
        g_warning ("user dictionary doesn't exist");
        return FALSE;
    }

    return tomoe_dict_register_char (priv->user_dict, chr);
}

gboolean
tomoe_context_unregister (TomoeContext *context, const gchar
*utf8)
{
    TomoeContextPrivate *priv;
```


```
    g_return_val_if_fail (TOMOE_IS_CONTEXT (context), FALSE);

    priv = TOMOE_CONTEXT_GET_PRIVATE (context);
    if (!priv->user_dict) {
        g_warning ("user dictionary doesn't exist");
        return FALSE;
    }

    return tomoe_dict_unregister_char (priv->user_dict, utf8);
}

TomoeChar *
tomoe_context_get_char (TomoeContext *context, const gchar
*utf8)
{
    TomoeContextPrivate *priv;
    TomoeShelf *shelf;
    TomoeChar *chr = NULL;
    GList *names, *node;

    g_return_val_if_fail (TOMOE_IS_CONTEXT (context), chr);

    priv = TOMOE_CONTEXT_GET_PRIVATE (context);

    shelf = priv->shelf;
    if (!shelf) return chr;

    names = tomoe_shelf_get_dict_names (shelf);
    if (!names) return chr;

    for (node = names; node; node = g_list_next (node)) {
        const gchar *name = node->data;
        TomoeDict *dict;

        dict = tomoe_shelf_get_dict (shelf, name);
        chr = tomoe_dict_get_char (dict, utf8);
        if (chr)
            break;
    }

    return chr;
}
```

Appendix 3: tomoe-recognizer-simple.c

```
#include <stdlib.h>
#include <gmodule.h>

#include <tomoe-module-impl.h>
#include <tomoe-recognizer.h>
#include "tomoe-recognizer-simple-logic.h"

#define TOMOE_TYPE_RECOGNIZER_SIMPLE
tomoe_type_recognizer_simple
#define TOMOE_RECOGNIZER_SIMPLE(obj)
(G_TYPE_CHECK_INSTANCE_CAST ((obj),
TOMOE_TYPE_RECOGNIZER_SIMPLE, TomoeRecognizerSimple))
#define TOMOE_RECOGNIZER_SIMPLE_CLASS(klass)
(G_TYPE_CHECK_CLASS_CAST ((klass), TOMOE_TYPE_RECOGNIZER_SIMPLE,
TomoeRecognizerSimpleClass))
#define TOMOE_IS_RECOGNIZER_SIMPLE(obj)
(G_TYPE_CHECK_INSTANCE_TYPE ((obj),
TOMOE_TYPE_RECOGNIZER_SIMPLE))
#define TOMOE_IS_RECOGNIZER_SIMPLE_CLASS(klass)
(G_TYPE_CHECK_CLASS_TYPE ((klass),
TOMOE_TYPE_RECOGNIZER_SIMPLE))
#define TOMOE_RECOGNIZER_SIMPLE_GET_CLASS(obj)
(G_TYPE_INSTANCE_GET_CLASS((obj), TOMOE_TYPE_RECOGNIZER_SIMPLE,
TomoeRecognizerSimpleClass))

enum {
    PROP_0,
    PROP_DICTIONARY
};

typedef struct _TomoeRecognizerSimple TomoeRecognizerSimple;
typedef struct _TomoeRecognizerSimpleClass
TomoeRecognizerSimpleClass;

struct _TomoeRecognizerSimple
{
    TomoeRecognizer  object;
    TomoeDict       *dict;
};

struct _TomoeRecognizerSimpleClass
{
    TomoeRecognizerClass parent_class;
```

```
};

static GType tomoe_type_recognizer_simple = 0;
static GObjectClass *parent_class;

static GObject     *constructor               (GType
type,
                                               guint
n_props,

GObjectConstructParam *props);
static void         dispose                   (GObject
*object);
static void         set_property              (GObject
*object,
                                               guint
prop_id,
                                               const GValue
*value,
                                               GParamSpec
*pspec);
static void         get_property              (GObject
*object,
                                               guint
prop_id,
                                               GValue
*value,
                                               GParamSpec
*pspec);
static GList       *search                    (TomoeRecognizer
*recognizer,
                                               TomoeWriting
*input);

static void
class_init (TomoeRecognizerSimpleClass *klass)
{
    GObjectClass *gobject_class;
    TomoeRecognizerClass *recognizer_class;

    parent_class = g_type_class_peek_parent (klass);

    gobject_class = G_OBJECT_CLASS (klass);
    gobject_class->constructor  = constructor;
    gobject_class->dispose      = dispose;
```


```
    gobject_class->set_property = set_property;
    gobject_class->get_property = get_property;

    recognizer_class = TOMOE_RECOGNIZER_CLASS (klass);
    recognizer_class->search = search;

    g_object_class_install_property (
        gobject_class,
        PROP_DICTIONARY,
        g_param_spec_object (
            "dictionary",
            "Dictionary",
            "The dictionary of the recognizer",
            TOMOE_TYPE_DICT,
            G_PARAM_READWRITE | G_PARAM_CONSTRUCT_ONLY));
}

static void
init (TomoeRecognizerSimple *recognizer)
{
}

static void
register_type (GTypeModule *type_module)
{
    static const GTypeInfo info =
        {
            sizeof (TomoeRecognizerSimpleClass),
            (GBaseInitFunc) NULL,
            (GBaseFinalizeFunc) NULL,
            (GClassInitFunc) class_init,
            NULL,           /* class_finalize */
            NULL,           /* class_data */
            sizeof (TomoeRecognizerSimple),
            0,
            (GInstanceInitFunc) init,
        };

    tomoe_type_recognizer_simple =
        g_type_module_register_type (type_module,
                                     TOMOE_TYPE_RECOGNIZER,
                                     "TomoeRecognizerSimple",
                                     &info, 0);
}
```


```
G_MODULE_EXPORT GList *
TOMOE_MODULE_IMPL_INIT (GTypeModule *type_module)
{
    GList *registered_types = NULL;

    register_type (type_module);
    if (tomoe_type_recognizer_simple) {
        gchar *name = (gchar *) g_type_name
(tomoe_type_recognizer_simple);
        registered_types = g_list_prepend (registered_types,
name);
    }

    return registered_types;
}

G_MODULE_EXPORT void
TOMOE_MODULE_IMPL_EXIT (void)
{
}

G_MODULE_EXPORT GObject *
TOMOE_MODULE_IMPL_INSTANTIATE (const gchar *first_property,
va_list var_args)
{
    return g_object_new_valist (TOMOE_TYPE_RECOGNIZER_SIMPLE,
                                first_property, var_args);
}

G_MODULE_EXPORT gchar *
TOMOE_MODULE_IMPL_GET_LOG_DOMAIN (void)
{
    return g_strdup (G_LOG_DOMAIN);
}


static GObject *
constructor (GType type, guint n_props,
             GObjectConstructParam *props)
{
    GObject *object;
    GObjectClass *klass = G_OBJECT_CLASS (parent_class);
    TomoeRecognizerSimple *recognizer;

    object = klass->constructor (type, n_props, props);
```

```
    recognizer = TOMOE_RECOGNIZER_SIMPLE (object);

    if (!recognizer->dict) {
        gchar *filename = g_build_filename (RECOGNIZER_DATADIR,
                                            "Devanagari.xml",
                                            NULL);
        recognizer->dict =
            tomoe_dict_new ("xml",
                            "filename", filename,
                            NULL);
        g_free(filename);
    }

    if (!recognizer->dict) {
        g_warning ("dictionary isn't set for
TomoeRecognizerSimple.");
    }

    return object;
}

static void
set_property (GObject *object,
              guint prop_id,
              const GValue *value,
              GParamSpec *pspec)
{
    TomoeRecognizerSimple *recognizer = TOMOE_RECOGNIZER_SIMPLE
(object);

    switch (prop_id) {
      case PROP_DICTIONARY:
        if (recognizer->dict)
            g_object_unref (recognizer->dict);
        recognizer->dict = g_value_get_object (value);
        if (recognizer->dict)
            g_object_ref (recognizer->dict);
        break;
      default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
pspec);
        break;
    }
}
```

```
static void
get_property (GObject *object,
              guint prop_id,
              GValue *value,
              GParamSpec *pspec)
{
    TomoeRecognizerSimple *recognizer = TOMOE_RECOGNIZER_SIMPLE
(object);

    switch (prop_id) {
      case PROP_DICTIONARY:
        g_value_set_object (value, recognizer->dict);
        break;
      default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
pspec);
        break;
    }
}

static void
dispose (GObject *object)
{
    TomoeRecognizerSimple *recognizer;

    recognizer = TOMOE_RECOGNIZER_SIMPLE (object);

    if (recognizer->dict)
        g_object_unref (recognizer->dict);

    recognizer->dict = NULL;

    G_OBJECT_CLASS (parent_class)->dispose (object);
}
```

```
static GList *
search (TomoeRecognizer *_recognizer, TomoeWriting *input)
{
    TomoeRecognizerSimple *recognizer;

    recognizer = TOMOE_RECOGNIZER_SIMPLE (_recognizer);
    return _tomoe_recognizer_simple_get_candidates (_recognizer,
                                                    recognizer-
>dict,
                                                    input);
}
```

CHAPTER 6

BIBLIOGRAPHY / REFERENCES

[1] GNOME Documentation Library. [Online]. Available: http://library.gnome.org/devel/

[2] CellWriter: Open source handwriting recognition for Linux – By Nathan Willis
. [Online Blog]. Available http://www.linux.com/archive/feature/120867 /

[3] Mathieu's Log. [Online Blog]. Available: http://www.mblondel.org/journal/category/google-soc/

[4] e-mail: info@anderssonj.com. [Online Blog]. Available http://anderssonj.com/

[5]Tomoe- A handwriting Recognition Engine. [Online Website]. Available:
http://tomoe.sourceforge.jp/cgi-bin/en/blog/index.rb/

[6] Hidden Markov Model. [Online]. Available: http://en.wikipedia.org/wiki/Hidden_Markov_model/

[7] Roger Pressman, *Software Engineering-A Practitioner's Approach*. New Delhi: Tata McGraw Hill, 2006.