MAINTENANCE MANUAL

Table of Contents

1. Introduction

1.1 Purpose and scope

The purpose of this maintenance plan is to provide information about the project Tambic in order to maintain, modify and update the project in the future. This document will cover the overview of the system, software and hardware environment introduction, maintenance hints, further development plan, etc.

1.2 Product and environment

The TamBiC system is an online application that provides users with an easy-to-use, on-line access to the Tampere Bilingual Corpus database that contains a large number of Finnish and English texts which are accompanied by their translations into the other language.

The main users will be staff and students of English Philology, who will use the system to gather research material for e.g. their thesis works. Using the provided graphical search tool, the users will be able to, for example, look up all English phrases, where some certain word is used, and study, what the corresponding Finnish translation phrases look like.

The application will be hosted in a server from the Computer Center at University of Tampere. The application will be online all year long and can be access via browser from any computer with internet access.

1.3 Definitions, acronyms and abbreviations

CSS	=	Cascading Style Sheet.
Corpus	=	collection of sample texts
DB	=	Database
Eclipse	=	Programming environment
FTP	=	File Transfer Protocol.
HTML	=	Hyper Text Markup Language.
MS-DOS	=	Microsoft Disk Operating System; command line based operating system, released in 1981.
MVC	=	Model-View-Controller; recommended model to be used when planning how to divide software into individual components.
OS	=	Operating system.
PHP	=	Hypertext Preprocessor.
PostgreSQL	=	Type of relational databases
SRS	=	Software Requirement Specification
SVN	=	Subversion: a version control tool
TamBiC	=	Tampere Bilingual Corpus; the name of the project and the new system to be implemented during the project.
UI	=	User Interface
GWT	=	Google Web Toolkit

1.4 References

[Cooper, 1998] Robert W. Cooper, The Tampere bilingual corpus of Finnish and English: development and applications." In: *Compare or Contrast? Current Issues in Cross-Language Research,* Robert W. Cooper (ed.), University of Tampere, 1998.

[Cooper, 2006] Robert W. Cooper (2006). Short description of Robert Cooper's personal corpus work: "The Tampere Bilingual Corpus of Finnish and English". Can be retrieved from http://www.uta.fi/laitokset/kielet/engf/staff/cooper_projects.html.

[Cooper, 2009] Robert W. Cooper, Proposal for software project.
[Wikipedia, 2009] Wikipedia's estimation about usage share of web browsers (September 2009). Can be retrieved from http://en.wikipedia.org/wiki/Usage_share_of_web_browsers.

1.5 Overview

Chapter 1 describes the project and the product and the purpose of the document. The chapter also includes a list of references used in the document and the terminologies.

Chapter 2 explores the purpose, user, functions and general restrictions and licenses used.

Chapter 3 deals with the environment: hardware environment, software environment, communications environment and compatibility.

Chapter 4 describes the overview of implementation.

Chapter 5 gives a quick sight to files and components.

Chapter 6 one of the most important chapters in this document gives all the maintenance hints that will be used when the maintenance is performed.

Chapter 7 an introduction to the reports and documentation.

Chapter 8 gives an explanation about the known problems and errors.

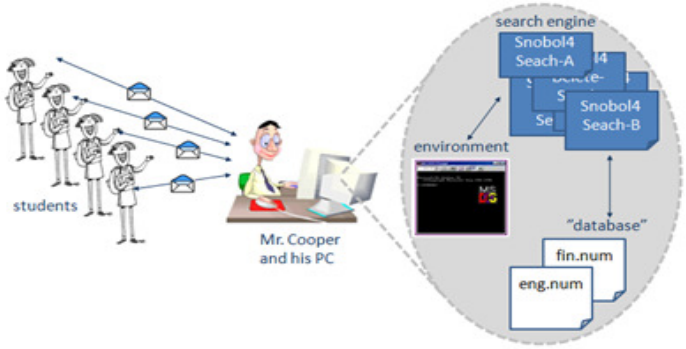
Chapter 9 describes the rejected alternatives.

Chapter 10 contains new ideas that may be employed in the project for further development.

2. Overview

2.1 purpose

The client and the primus motor of the TamBiC project was Mr. Robert Cooper from the Tampere University School of Modern Languages and Translations Studies. Over the years, Mr. Cooper has carefully selected and gathered a collection of texts in both English and Finnish language. The old system worked so that students contact Mr. Cooper personally, asking for material on certain words or phrases. In order to perform different kind of searches in the 2 text files, Mr. Cooper has used his own Snobol4-programs in MS-DOS environment. After Mr. Cooper has made the searches, the then e-mails the findings back to the students for them to analyze.



Picture 1: old TamBiC system

As all the searches in the text files are done on command line basis, and the programs input and output .txt files, the risk of accidentally over-writing the actual corpora files is always present. Also, the Snobol4-programs are a bit difficult to use for modern students, who are more used to using graphical interfaces than MS-DOS environment. As all the searches in the text files are done on command line basis, and the programs input and output .txt files, the risk of accidentally over-writing the actual corpora files is always present. Also, the Snobol4-programs are a bit difficult to use for modern students, who are more used to using graphical interfaces than MS-DOS environment.

As Mr. Cooper will be retiring in a few years, he would like to leave the corpora he has collected as his legacy to his department. However he wanted the TamBiC corpus system to be updated so that it would be enticing, exiting and easy to use for the future students - as he would not be there to guide them using it. For that purpose, an on-line search engine was be needed.

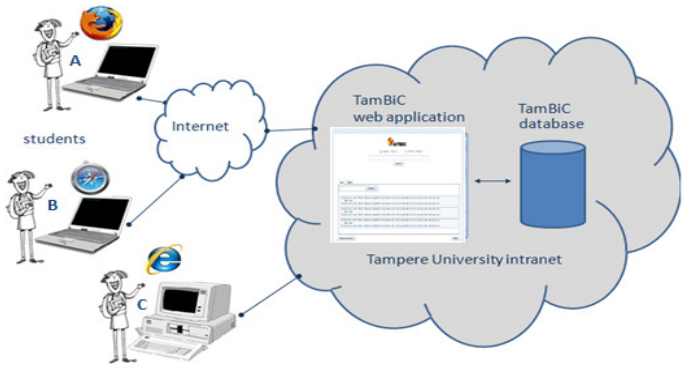
- TamBiC project mission: **transform the existing corpus system into a web application**
- TamBiC project goals:
  - ✓ Make the bi-lingual corpora **easily approachable** by reducing the complexity in the way sophisticated searches are done.
  - ✓ Provide users with a **modern looking**, **user friendly and intuitive** GUI.
  - ✓ Allow a **large number of users to access** the database simultaneously for searching material, while securing the database content to be editable by one key user (i.e. the administrator) only.
  - ✓ **Empower** students and staff by providing them with means to **independently use the corpus on-line** to perform the searches and language analysis in their personal work.
  - ✓ Make **operations** in the corpus database **fast and reliable**.

The new TamBiC system empowers the students to use the corpus independently on-line - to perform the searches and language analysis needed for their thesis works in their own – thus saving Mr. Coopers valuable time from being squandered on some trivial, routine work. It will also provides the teachers, researchers and students in the School of Languages and Translation Studies with a possibility to use this TamBiC application in the language/translation courses, or maybe, for their own, common research purposes. Principally, the new system will allow a large number of users to access the database simultaneously for reading, while securing the database content to be editable by one key user (i.e. the administrator) only.

The new TamBiC system makes the corpus more easily approachable and appealing by reducing the complexity of the way searches are done. The visuals of the graphical UI make the process of performing the searches easier to track, which further enhances the users understanding on how to best extract a certain kind of information from the corpus. Printing of the search results, or saving or pasting the results into a new file (e.g. Microsoft Word) is possible.

Adding, editing and deleting to the corpus database is now reliable. With the data store securely in a separate database, and protected with access right levels, there is no more risk of accidentally erasing the database contents while performing subsequent searches. As the whole TamBiC system is updated, the future of the TamBiC corpus is now longer dependent on Mr. Cooper's personal efforts.

In the new TamBiC system, the students can access the corpus database via the Tampere University intranet. The corpus search engine is a web application, available via any web browser.



Picture 2: new TamBiC system

2.2 Users

- **Core Users (Administrator)**
  The most important user of the system is Mr. Cooper, as the only administrator now for the expecting system. Mr. Cooper will frequently utilize all the features the system contains. He is a very experienced user of this system as he has been working on the old version system for over 20 years.

  The administrator characteristic will allow him to add more data to the database, modify and delete existing data. Change his password as admin, and also to create and modify other users that have access to the admin tool.

- **Basic Users**

Every typical period of time, there would be certain students of English Philology who will or are taking Mr. Cooper's courses. And those students will become the main users of the system. During the certain period of time, such as one study period or semester, the main users will often access to the system. The main users probably have little experiences in using corpus systems; however, they are without doubt fast learners.

Also other people interested in linguistics will be able to access the corpus and perform the searches.

2.3 Functions

1. **Word matching with or without wildcard**- Allows the user to search for a word or a word with any letters before or after. This gives total new possibilities as the user will be able to find inflexions of the words in a very easy and fast way.
   (e.g. play, play*, *play, play*, *play*, …)

2. **Phrase matching "…" with or without wildcards-** Allows the user to search for an specific phrase, the use of quotations define the phrases. It is possible to use the wildcard inside a phrase also. The use of a wildcard between two words including spaces allows searching for phrases containing the first word anything in between and the last word. Making the feature very powerful.
   (e.g. "the one", "the *ones", "the * one", …)

3. **Distinguish upper and lower case letters-** This function can be turn On/Off , depending on the kind of search we want to do. The default setting is Off, meaning that when the search is performed no attention is paid to the upper cases (e.g. searching for cat will find cat, Cat, CAT, …) When the option is On the matching will be done considering exactly the way the string was given (e.g. searching for Cat will find Cat only)

4. **OR syntax** – This allows you to find *any one* of several words. (e.g. **go OR went)** will find sentences containing either one of these words.
   go OR went (finds: they often **go** there / we often **went** there, etc)

5. **AND syntax** – If you place the **AND** operand between two words, this will find sentences containing *both* words. The words can appear be in any order.
   he AND man (finds: **he** was a **man** / **he** saw a **man**, etc)

6. **THEN syntax** – If you place **THEN** immediately before a word, TamBiC will find sentences containing both words *in that order*:
   talk THEN about (finds: I shall **talk** to him **about** it)

7. **NOT syntax** – If you place **NOT** immediately before a word, this indicates that you want to exclude sentences containing this word. The **NOT** operand only affects the word which follows it.
   depart* NOT department (only finds: **depart, departs, departed, departing**)

8. **Showing more context-** This option allows you to displays the next and previous sentence in the text.

9. **Show all more context**- This option works as the showing more context, but it shows the context of all the search results.

10. **More than one operand** – It is possible to use the same operand twice in the same search (e.g. **dog OR cat OR horse**; **sing* NOT singer NOT single**; etc).

    Different operands, on the other hand, should not be used in the same search query. Instead, perform two separate searches.

11. **"..."** Phrases in quotation marks may be combined with the other operands: **"in the evening" OR "at night"**; **"as * as" NOT "as well as" NOT "as soon as"**

12. **Print-** For printing (using the on-screen format of the website).

13. **Save-** This enables you to save the currently displayed search results on your computer. The saved file will be in .rtf format, and can be renamed if you so wish.

14. **Help-** Provides information on the functions that the corpus can perform.

15. **About-** Gives an overview of what the corpus is about and people related.

16. **Login-** Access to update the corpus is restricted so that the administrator system is only available for group users who are accredited to modify the corpus.

17. **Logout-** The administrator system provides the user a way to explicitly log out from the administration system. If the browser is closed, the user session will terminate automatically.

18. **Password change-** If needed, the user can change the password from the administrator system.The admin can change the password of any user.

19. **Add new user-**It must be possible to add new users to the administrator system (the user and password are defined by the admin), this will only be done by the main admin .

20. **Remove user-**in the situation that a user account is created accidentally or that some users lose his rights to administrate the corpus, the admin tool provides means for deleting user accounts. The purpose is to delete a user account.

21. **Add new entry-** The administrator will be able to update the corpus,it must be possible to add new texts to the corpus.

22. **Delete existing entry-** It is desirable that the admin tool provides means for deleting sources. The purpose is to delete the entry from the system in case is not needed any more.

23. **Edit existing entry-** It is possible to modify an existing entry and change the text the code or the acceptance permission.

24. **Admin tool manual-** contains information about the functions of the admin tool and how to use it.

2.4 General restrictions

There are limitations for using the administration tool, as there are different access right levels. The users with basic access rights are able to add edit and delete texts in the database, while the users

with full administrative rights are able to edit the details of the registered users and managing texts in the database.

For the corpus itself there are no restrictions and searches can be performed without further limitations. However, making searches that produces thousands of results (such as looking for English word “he”) will take some time to render on the screen. Also, saving and printing search result files with hundreds of results may take several tens of seconds to complete.

2.5 Assumptions and dependencies

1) This system assumes that all the users have at least some basic computer utilizing and web browsing experience.

2) This corpus system assumes that all the users have at least relatively high Finnish and English language proficiency.

The display format should be always:

Original language text -	The group immigrated to Western Canada in the late 1800s to escape brutal persecution.
ID code -	(PRESS 1:1:11)
Translated text -	Ryhmä muutti Länsi Kanadaan 1800 luvun lopulla paetakseen julmaa vainoa.

3. Environment

3.1 Hardware environment

Tambic runs on the server of Department of Computer Science in University of Tampere. So the hardware environment will be according to the hardware environment of the server of Department of computer science. There are no special hardware requirements to use the corpus. The user just needs a computer and the Internet connection. When the user wants to print the results a printer is needed.

3.2 Software environment

3.2.1 Development

Tools used for development were:

- PostgreSQL 8.4
- PHP version 5.2.0
- Eclipse 3.5 Galileo for PHP Developers
- Google Web Toolkit 2.0

3.2.2 Users

The access to the application can be done by:

- Mozilla Firefox 2.0 or higher
- Internet Explorer 6 or higher
- Any other web browser

See “SW_Project_Structure.pdf” in project Wiki/CD “Implementation” section for more information about the SW project directory structure and how to set up an Eclipse working environment.

See “Corpus_database_design.pdf” in project Wiki/CD “Implementation” section for more information about the structure of the Postgres database tables used in the corpus database.

In this document, section “How to” describes actions required to handle the corpus database.

3.3 Communications environment

Detailed information about the communications environment is not available.

3.4 Compatibility

The system is compatible with all the already version software.

4. Overview of implementation

4.1 Architecture and design solutions

4.1.1 Software architecture

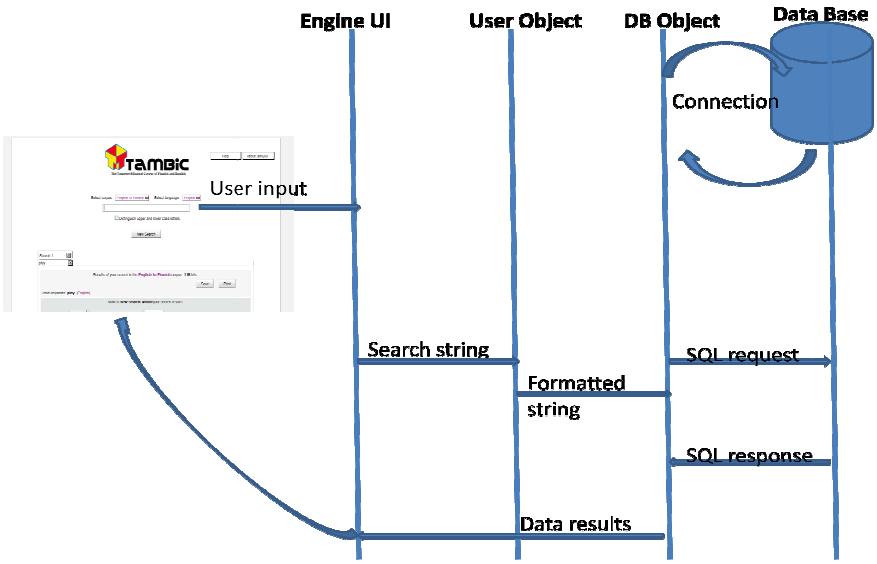
The TamBiC search engine and admin tool were created using JavaScript and PHP scripts upon web pages implemented using HTML and CSS. The search engine UI and its dynamics are implemented in JavaScript - or actually, they were written in Java, taking advantage of the Google Web Toolkit, which renders the written Java code into executable JavaScript.

While GWT is versatile, using it in a complicated UI project requires a structured development environment, like the Eclipse we used. Working with GWT produces results quickly in running code, but due to the lack of general "framework" in JavaScript, maintaining and editing the code files proved a bit tedious.

PHP scripting was used for creating dynamic web page content in the Admin tool web page forms. For that usage, PHP code was embedded into an HTML source document and then interpreted by a web server, which generates the web page document. PHP was also used to execute SQL queries in the database and to create printable/savable result files. As PHP can be used for command-line scripting and client-side, it also worked for us as the primarily filter in the search engine, taking input from a file or DB and outputs another stream of data. PHP is free to use, and it's released under the PHP License.

The actual database was created in PosgreSQL, an open-source Object-Relational DBMS supporting almost all SQL constructs.

The following explains the PHP interaction.



Picture 3: System Interaction Diagram

To see more information about the classes see Section 5.2 of this document.

See also "Search_engine_design.pdf" in project Wiki/CD "Implementation" section for more information about the structure and functionality of the search engine.

4.1.2 Hardware architecture

The TamBiC web-based application is hosted by the University of Tampere, on a Linux server. The Corpus system can be accessed from any computer with the Internet connection as long as the server is running.

No real hardware was designed for the project. Hardware architecture can change depending on the user.

4.2 Modules

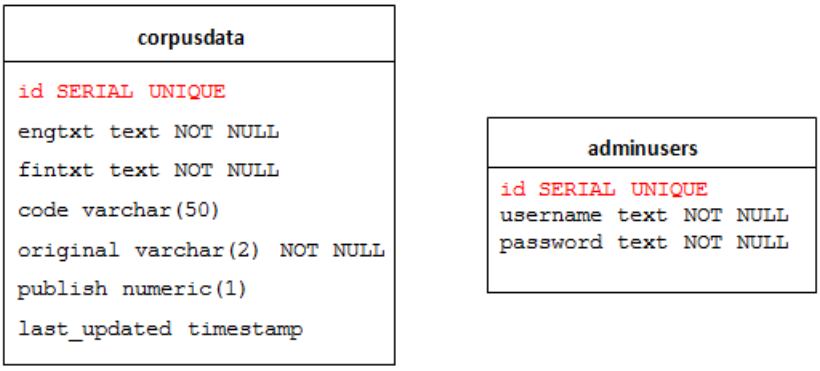
The provided new TamBiC system consists of 3 parts:

1. **SQL database holding the contents of the two corpora.** Features:
   - Provides efficient methods for searches based on string pattern matching
   - Provides easy methods for inserting and editing the corpora contents

2. **Administrative web application for developing the database**. Features:
   - Web form for making searches on rows (by text code)
   - Web form for editing and deleting rows
   - Web form for adding new data to the database

3. **Web application for making the searches in corpora.** Features:
   - Basic searching, where user needs to select corpus and language (English/Finnish)
   - Making a new search within your current search results (i.e. refining results)
   - Versatile search options:
     - Word matching w/o wildcard (e.g. play, play*, *play, play*, *play*, …)
     - Phrase matching w/o wildcards (e.g. “the one”, “the *ones”, “the * one”, …)
     - Case in/sensitivity: turn On/Off (e.g. cat, Cat, CAT, …)
     - AND, OR and NOT syntax (e.g. cat AND dog, cat NOT dog, cat OR dog, …)
     - Showing more context, which displays the next and previous sentence in the text
     - THEN syntax , which finds sentences where words are in the given order
   - Graphical UI, where the results are shown with hit counting and matching words being bolded in the search results to provide an easy verification of the results
   - Having several searches open at the same time, since every results is put in a new tabs.
   - Saving results (in a .rtf file) w/o sentence context being saves, as well
   - Printing results (from a new .html page) w/o sentence context being printed, as well
   - “Help” page, which contains the user manual on how to use the system
   - “About TamBiC” page, where user can find more information on e.g. the text sources.

4.3 Database

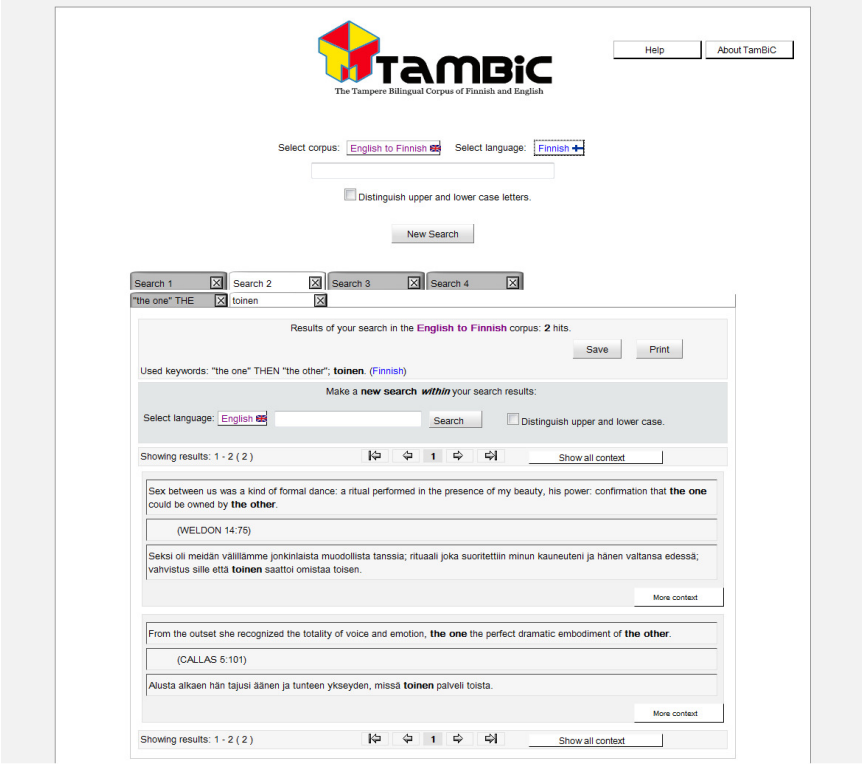


Picture 4: TamBiC database tables

The DB contains only one table in which all the data is stored, is a very easy table that contains the English, finish text, the code and the most important part will be the origin as it defines in which language was the original text. The table with data should look something like this:

Id	Entxt	Fintxt	Code	Original	Publish	Last_update
1	There is no other area on the earth that resembles Finland in this respect, lying on a suitable north-south axis within the Aurora Borealis zone, but also in an area which *is* adequately populated and provided with electricity.	Maapallolla ei ole toista Suomen kaltaista aluetta, joka *sijaitsisi* revontulivyöhykkeellä ja *olisi* lisäksi sopivasti pohjois-etelä suuntainen sekä tarpeeksi asutettu ja sähköistetty.	TRANS D6:5	Fintxt	1	29.11.2009 15:02:11
2	Especially during the 1990s the climate *was* considerably warmer.	Eritysesti 1990-luvulla on ollut huomattavasti lämpimämpää.	SUN J4:5	Entext	0	29.11.2009 18:05:11
…	…	…	…	…	…	…

4.4 User interface



Some description of some parts of the interface and their function:

1. **Scope of the corpus:** the TamBiC corpus is a bilingual translation corpus and is in two parts: original English texts together with their Finnish translations (“English to Finnish”), and original Finnish texts with their English translations (“Finnish to English”). You can select the part of the corpus you want to search by clicking on the Select Corpus button.
2. **Language selection:** it is possible to search in either the original language or in the translations. Just click on the Select Language button.
3. **Search box:** This is where you type the word(s) you are searching for.
4. **Distinguish upper and lower case letters:**normally the search engine will look for words beginning with either upper or lower case letters. Click here if you want the search to include (or exclude) words with initial capitals (i.e. so that searching for **dog** is NOT the same as searching for **Dog**).
5. **Help button:** This is a link to the help file. Click here for detailed information about the search engine and the different features that are available.
6. **About TamBiC:** This is a link to the information file. It contains information about the corpus, the authors and texts that are included, and the development of the search engine.

5. Files and components

5.1 General information

The following table gives the results of the lines on the project as well as the number of functions:

Language	Java (/JavaScript), PHP, HTML, CSS
LOC	Total: **6200 LOC** ○ Search engine: ~ 5200 LOC ○ CSS: 920 ○ HTML: 790 ○ Java: 2650 ○ PHP: 880 ○ Admin tool: ~ 960 LOC (*) ○ HTML: 90 ○ PHP: 870
SLOC	TOTAL: **4800 SLOC** ○ Search engine: ~3900 SLOC ○ CSS: 760 ○ HTML: 600 ○ Java: 2000 ○ PHP: 530 ○ Admin tool: ~ 900 SLOC (*) ○ HTML: 90 ○ PHP: 810
Reused code	0
Reused and modified	0
Files and classes	TOTAL: **46 files** Search engine: 24 files • CSS: 5 files • HTML: 6 files • Java: 8 files, 8 classes • PHP: 5 files, 5 scripts Admin tool: 22 files • HTML: 3 files • PHP: 19 files
Functions	Search engine: • Java: 57 • PHP: 22 Admin tool: • PHP: 26
Code revisions	several

5.2 Functions and classes

The following are some PHP classes and with their functions.

DBObjectInterface.php

// Called from constructor, must return true if succeeded, false if not
// - init tables for queries and results
// - setup sql database connection

```
        //   - store connection so that if
becomes persistent
        public function init();

        // Called when solution is about to be
closed ( aka user logging off )
        // - deinits tables
        // - removes the sql connection
        public function deinit();

        // Called from UIObject, performs
search. queryString is the keyword(s) to
search for
        // queryID is a id of search in question
( aka determinates if new tab needs to be
opened )
        // and refinedSearch is to know if this
search should be performed within previous
search( queryID)
        // - search database or result for given
keyword(s)
        // - store result into table associated
with queryID
        // - return true if query is performed
with somesort of results, or false if there
weren't results at all
        //   or if the keyword(s) were invalid in
someway ( too short etc )
        public function performQuery(
$queryString, $queryID, $refinedSearch );

        // Returns results as string from table.
Separator for each search results might be '/'
        // for example 'search string 1/search
string2/search string 3'. queryID is table
location of results
        // we want. pageNumber indicates
from which point to which point we want
results from..
        // - get result from table from index
queryID
        //
        public function getResults( $queryID,
$pageNumber );

        // Called when user tries to log in.
User gives username and password in 'login'-
screen.
        // - check if username and password is
a match in database.
        // - return true if user is
authenticated, false if not.
        public function authenticate(
$username, $password );
```

UIObjectInterface.php

```
        // Loads javascripts, css-files and basic
        html tags.
        public function init();

        // Puts end tags..
        public function deinit();

        // Sets Desired view
        public function setView( $view );

        public function render();

        // Renders the skeleton of loginpage.
        public function renderLoginPage();

        // Renders the skeleton of mainpage
        public function renderMainPage();

        // Renders tabs ( which are located
within mainpage )
        public function renderTab();

        // Renders the menu
        public function renderMenu();

        // Renders the results( which are
located within the tabs )
        public function renderResults();
```

UserObjectInterface.php

```
        // Called from constructor, sets
        variables to default state..
        public function init();

        // Returns true if user is
        authenticated, false if not
        public function isAuthenticated();
```

```
// Called when we want this user
object to be authenticated
public function setAuthenticated();


// Sets username and password (
maybe crypted )  for this user object
public function setCredentials(
$username, $password );

// Returns username
public function getUsername();
```

5.3 Refining a search

"Refining" is basically always an AND-operation; when you refine your search, you want to LIMIT somehow the search results you've already got...

So, each result tab in the UI stores the search key string that was used to get there, and when a new refined search is made, the search keyword will then contain (all previous keywords)AND(new keywords). And, again, the latest search string is stored to the latest tab in the UI.

Like this:

Basic level search: cat OR kitten*
1st result tab will maintain this:

EN KEY: (|NOT=0|CASE=0|KEY=_cat_|OR|NOT=0|CASE=0|KEY=_kitten*|).
FI KEY: .
Sql query: (engtxt ~* '_cat_' OR engtxt ~* '_kitten*') .

Next refined search (using 1st tab results): kissa
2nd result tab will maintain:

EN KEY: (|NOT=0|CASE=0|KEY=_cat_|OR|NOT=0|CASE=0|KEY=_kitten*|).
FI KEY: (|NOT=0|CASE=0|KEY=_kissa_|).
Sql query: ((engtxt ~* '_cat_' OR engtxt ~* '_kitten*') AND (fintxt ~* '_kissa_')).

Next refined search (using 2nd tab results): NOT pappi
3rd result tab will maintain:

EN KEY: (|NOT=0|CASE=0|KEY=_cat_|OR|NOT=0|CASE=0|KEY=_kitten*|).
FI KEY: (|NOT=0|CASE=0|KEY=_kissa_|)AND(|NOT=1|CASE=0|KEY=_pappi_|).
Sql query: ((engtxt ~* '_cat_' OR engtxt ~* '_kitten*') AND (fintxt ~* '_kissa_') AND (fintxt !~* '_pappi_')).
---
etc.

(As you can notice, in the generated description string, "NOT" keyword means if its a "not this word" search (1) or a normal "please find this word" (0), CASE key means case sensitivity is ON (1) or OFF (0), and KEY has the actual keyword... AND and OR are used to combine those NOT+CASE+KEY patterns.

The primitive SQL search string - with regular expression pattern matching - is then generated from those keys.

6. Maintenance hints

6.1 Maintenance principles

Maintenance of the system should ensure the existing features not to be broken. All maintenance should be recorded into this maintenance manual.

If new features are added the Requirement Specification document should be also updated.

We strongly recommend performing the tests from the Test Plan document at the end of each maintenance to ensure the total functionality and reliability from the application.

6.2 Compiling

To recompile the project you need to install Eclipse 3.5 Galileo for PHP Developers with PHP version 5.2.0, and then add the plug in of Google Web Toolkit 2.0.

For the search engine UI as it is dynamic the program is written in Java and once its compiled by the GTW you will obtain the JavaScript files.

6.3 Installation

There are no special installations tools created or used. The requirements for the system are described in the Implementation Plan.

The system environment installation can be performed using the following steps:

1. Create a database in PostgreSQL 8.4 instance using the file which includes the information in the defined forma and follow section 6.2
2. Upload the folder WAR in to the web server location that can be found in the CD.
3. The corpus should be ready to use!!
4. For the admin, first start by logging in to the Admin Tool. Use predefined user account with username: admin and password: admin.
5. Create your personal account for Administration and delete the predefined user account to protect system from abuse.
6. Start using the Admin Tool!

6.4 Data Base

See section 7.

6.5 User interface

The UI buttons and pictures “look” can be updated from the CSS files. For changing components positions (i.e. panel structure) or functionality you’ll need to re-compile the whole code in Eclipse environment and load at least “jtambic” and “WEB-INFO” directories to the used server.

6.6 Portability (/transferability)

No special attention has to be paid as long as sufficient PHP and PosgreSQL environments are provided.

6.7 Testing

See the Test Plan document for the test cases.

7. Modifying database - how to…

This section describes how to perform actions needed for maintaining and/or updating the corpus database. The current system is located in paprika.uta.fi, so you'll need access rights there. Also, you need to be able to access the "tambic" PostgreSQL database. Contact the Tampere University Computer Center for necessary privileges.

If you have a whole new version of the corpus text files, and you want to replace the entire existing corpus with the new version, see sections 7.1 and 7.3 for help.

If you have lost your admin username and/or password, see section 7.4 for help.

7.1 Turning your corpus .txt files into .cvs files

The corpus files are stored by Mr. Cooper in simple .txt files, which are encoded as "extended ascii". Since .txt files can not be uploaded to the Postgres DB as such, you'll first need to create 2 .cvs file out of those text files. This is how to do that:

1. Create a new text file with correct encoding in Unix environment:

```
iconv -f 437 -t UTF-8 ENG.nm3 > ENG_utf8.nm3
iconv -f 437 -t UTF-8 FIN.nm3 > FIN_utf8.nm3
```

2. Make a csv-file:

Open the new txt file in OpenOffice Writer (MS Word crashes!) and use "Search and Replace":

a) Search: " "
Replace: ""

b) Search: "$"
Replace: "#"

c) Save as .csv File

d) Edit csv-File with Calc or Excel program

Open Calc or Excel
Import data from text file
Separated
Delimiter = #; no textqualifier
format of all 3 columns = text
Add in column D for every line en or fi, for the corpus
Add in column E for every line 1 for accept

e) If Excel or Calc makes # for every column, you have to search and replace them all

7.2 Inserting new data to the tables

Log into server paprika.uta.fi (using e.g. SSH) and in a new terminal window, start a new PosgreSQL sessions (e.g. with command "psql tambic tambic").

Command: **INSERT INTO <table> (<column1>, <column2>, ...) VALUES ('<value1>', '<value2>', ...);**
Example:

```
INSERT INTO corpus.adminusers ( user, password ) VALUES ( 'tempUser', 'salasana' );
```

7.3 Creating new tables

Log into server paprika.uta.fi (using e.g. SSH) and in a new terminal window, start a new PosgreSQL sessions (e.g. with command "psql tambic tambic").

If there's an old table in the database (which is the case if the search engine and admin tool are functioning), first you'll need to either rename the old table so it won't be used anymore. Remember that the search engine/admin tool only access DB tables which are named correctly:

- If you're doing something for the search engine, you'll need to modify table "**corpus.corpusdata**", which contains the corpus texts.
- If you're doing something for the admin tool, you'll need to modify table "**corpus.adminusers**", which contains the usernames and passwords for all the users that can edit the database.

If you're creating tables to an EMPTY database - for instance, when transferring the system to a new environment - run the following command, and continue then from step 2:

```
CREATE SCHEMA corpus;
```

1. Rename the existing table(s) (if there are any):

Command: **ALTER TABLE <current_table_name> RENAME TO <new_name>;**
Example:

```
ALTER TABLE corpus.corpusdata RENAME TO old_corpusdata;
```

2. Create new table(s):

For corpus texts:

```
CREATE TABLE corpus.corpusdata (
 id SERIAL UNIQUE,
 engtxt text NOT NULL,
 fintxt text NOT NULL,
 code varchar(50) UNIQUE,
 original varchar(2) NOT NULL,
 publish numeric(1),
 last_updated timestamp
);
```

For these timetamp setting commands, you'll need 'super user' privileges. Running these commands is NOT absolutely necessary, but having this kind of timestamp system is a sign of good DB design...

```
CREATE LANGUAGE 'plpgsql' HANDLER plpgsql_call_handler LANCOMPILER 'PL/pgSQL';

CREATE OR REPLACE FUNCTION corp_stamp() RETURNS TRIGGER AS $corp_stamp$
    BEGIN
        NEW.last_updated := current_timestamp;
        RETURN NEW;
    END;
```

```
$corp_stamp$ LANGUAGE plpgsql;
CREATE TRIGGER corp_stamp BEFORE INSERT OR UPDATE ON corpus.corpusdata FOR EACH ROW
EXECUTE PROCEDURE corp_stamp();
```

For administrative users:

```
CREATE TABLE corpus.adminusers (
      id SERIAL UNIQUE,
      username text NOT NULL,
      password text NOT NULL
);
```

3. Upload new data to the new table

For corpus texts:

```
\copy corpus.corpusdata (engtxt, code, fintxt, original, publish) FROM
'/home/my_dir/engfin.csv' WITH DELIMITER AS '#'

\copy corpus.corpusdata (fintxt, code, engtxt, original, publish) FROM
'/home/my_dir/fineng.csv' WITH DELIMITER AS '#'
```

Note that the path (here: */home/my_dir/*) for the .cvs file must point to where the .cvs file is located in Unix. Use the correct .cvs names, too. Pay attention to the parameter order; its different in different .csv files.

For administrative users: see section 7.2 on how to insert new information to table, one row at a time.

4. …test that the search engine and/or admin tool work with the new tables…

5. Delete old table(s) (if there are any):

Command: **DROP TABLE <table_name>;**
Example:

```
DROP TABLE corpus.old_corpusdata;
```

7.4 Changing user name and/or password for Admin tool

If you've forgotten your username to the admin tool, you can check the details from the database. Log into server paprika.uta.fi (using e.g. SSH) and in a new terminal window, start a new PosgreSQL sessions (e.g. with command "psql tambic tambic").

Command: `SELECT * FROM corpus.adminusers;`

Should print out something like this:

```
id |   username    |             password
---+---------------+----------------------------------
 16 | user1admin    | a851308387ed93a0548c48b6150429aa
 17 | tempuser      | 0522302bfd3a69d0f260c01a794c4f86
(2 rows)
```

1. How to change username

Command: **UPDATE ONLY <tablename> SET username='<new_value'> WHERE id=<row_id>;**
Example:

```
UPDATE ONLY corpus.adminusers SET username='DonaldDuck' WHERE id=17;
```

You can verify the result with this: `SELECT * FROM corpus.adminusers;`
It should print out something like this:

```
id |   username    |             password
----+---------------+----------------------------------
 16 | user1admin    | a851308387ed93a0548c48b6150429aa
 17 | DonaldDuck    | 0522302bfd3a69d0f260c01a794c4f86
(2 rows)
```

2. How to change password

The admin tool expects that all passwords are md5 protected, i.e. the password in the adminusers table are NOT readable, but they're MD5 encrypted hash codes produced from the original password. There is no way of determining the original password from the stored hash code, so in case you've forgotten your password totally, you'll just need to change it.

When you insert a new password to the table, insert it as MD5 encrypted. Here's an online website that can count the hash code for you: http://www.onlinefunctions.com/ -> open Online MD5 Hash calculator

E.g. word "test_password" produces hash code "16ec1ebb01fe02ded9b7d5447d3dfc65", so

- insert the code "16ec1ebb01fe02ded9b7d5447d3dfc65" to the table as your password like this:
- use password "test_password" to log into admin tool.

Command: **UPDATE ONLY <tablename> SET password='<new_value'> WHERE id=<row_id>;**
Example: `UPDATE ONLY corpus.adminusers SET password='16ec1ebb01fe02ded9b7d5447d3dfc65' WHERE id=17;`

You can verify the result with this: `SELECT * FROM corpus.adminusers;`
It should print out something like this:

```
id |   username    |             password
----+---------------+----------------------------------
 16 | user1admin    | a851308387ed93a0548c48b6150429aa
 17 | DonaldDuck    | 16ec1ebb01fe02ded9b7d5447d3dfc65
(2 rows)
```

3. How to add new username+password: see section 7.2. Make sure the new username is md5 encrypted; otherwise it will not work.

7.5 Other practical commands

===== How to remove all rows in a table =====

DELETE FROM TABLE <tablename>

8. Reports and documentation

8.1 Bug report

The bug report was done in the wiki site and is an appendix of the Test Plan. The format use was:
PROBLEM: Description of the problem
REPORTED BY: Person who found the bug
STATUS: Corrected or Open

See the complete Test Plan for test cases.

8.2 System documentation and location

(1) SW (installed and on a back-up CD):

* Search engine
* Admin tool
* Corpus DB

(2) Info pages (Online and in Wiki):

- Help
- About
- User Manual

(3) Project documentation (in Wiki and as .pdf in the CD)

* README.txt (for CD only)
* Project plan
* Weekly reports
* Requirements specification (or corresponding product backlog)
* Test plan
* Final report
* Project's story (to the dept's report series)

(4) SW documentation (in Wiki and as .pdf in the CD)

* Implementation plan
* Test Report
* Usability analysis
* UI specification
* User's guide
* Online helps
* Maintenance document

+ Code files (.java, .PHP, .CSS, .HTML)
+ Executable code files

8.3 Documents to be updated

The following documents should be updated if more features or major changes are done to the application:

* Requirements specification (or corresponding product backlog)
* Maintenance document
* UI specification
* User's guide
* Online helps

8.4 Documents not to be updated

* Project plan and Project's story
* Weekly reports
* Final report

9. Known problems and errors

All bugs found during the implementation and testing were corrected and reported in the bug list.

10. Rejected ideas

The following ideas where rejected as we found that they were already implemented or not needed any more:

- Punctuation sensitivity to be considered or disregarded: this functionality was already built in the logic of the search scripts: the punctuation between words (i.e. letter characters) was already ignored, while looking for punctuation characters inside words was possible (like in word “didn’t”). Therefore there was no point on making this any kind of an extra option in a menu.
- An editing/sorting program which allows the user to automatically sort the output files into further (smaller) files: this was implemented as a refined search inside the main results, which was a much better solution usability wise that using some external file merging/appending program.
- Searching in both corpuses at the same time: this was rejected by the client as he consider very unlikely situation where you will need that, is also possible to do it in 2 separate searches.

11. Ideas for further development

We believe it would be very worthy to continue with this project next fall with another project team (2010) and add the functionality we were not able to provide:

For the admin tool:

- A web form, where you could upload a whole .txt document (in restricted format) and have that uploaded to the database as new entries (i.e. rows), would be very useful. Now user has to insert new data one entry at a time.
- A web form, where you could open a load of entries (based on e.g. giving to codes, “to” and “from”), and edit/delete them in a single view, would be useful. Now user can browse the database only one entry at a time.
- A possibility to delete the whole database and set up a new one - easily - would be needed, too.

For search engine:

- A totally different kind of UI approach might be more practical that the one that’s currently implemented. The current UI does not scale properly, and the amount the open tabs is limited to five (due to lack of horizontal space). New project should start with usability testing!
- Taking advantage of the dictionaries in PostgreSQL database would be beneficial, too. One could implement the lemmatized searching (i.e. you look for word “go”, and it finds you also “went”) as an alternative search method.
- Better handling of error situations (e.g. server does not respond/responds with an SQL error) would also be practical.
- Add pipe-syntax: instead of writing “dog THEN brown OR dog THEN black OR cat THEN brown OR cat THEN black” one could use simpler syntax “dog|cat THEN brown|black”.
- To help building more sophisticated queries, also support for using brackets would be needed. Brackets would be practical for indicating what parts the syntax should be evaluated first, e.g. “cat OR dog AND brown” could now be interpreted as both “cat OR (dog AND brown)” or “(cat OR dog) AND brown”.

- Add categorization feature, where you can define categories and then use the categories for each hit. You should be able to select from a drop down menu the category for that hit.
- Open this categorization in new tabs with a button, once you have select a category for each hit, they should open in different searches.
- Discontinued searches (e.g. as....as, have....ed)
- It should be possible to add the number of words that you want to search before the next word appear. (e.g. play 3 words *ed) Syntax: word1 [0-4] word2
- Combine results from search 1, search 2 and search 3 in a new file.