

# BIRKBECK COLLEGE

# UNIVERSITY OF LONDON

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

MSc Computer Science  
Project Report

## Vocabulary Acquisition using Spaced Repetition

Supervisor: Professor George D. Magoulas

Author: André Gordon Braithwaite

2016 / 2017

"This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. The report may be freely copied and distributed provided the source is explicitly acknowledged."

## Contents

Abstract.....	2
1. Introduction .....	3
1.1 Background.....	3
1.2 Motivation.....	3
1.3 Aim and Objective .....	4
1.4 Development Process.....	4
1.5 Development Environment.....	5
1.6 Report Structure.....	5
2. Literature Review.....	6
3. System Requirements .....	8
3.1 User Stories and Use Cases .....	9
3.1.1 Personas.....	9
3.1.2 Student stories.....	9
3.1.3 Administrator stories .....	9
3.1.4 Use Cases .....	10
4. System Design .....	24
4.1 Architecture.....	24
4.2 Usability and Visual Style .....	27
5. Implementation .....	28
5.1 Data Formatting Tool .....	28
5.2 Framework Development .....	30
5.3 Application Development.....	33
6. Testing.....	36
7. Conclusion and Further Work.....	37
References .....	38
Bibliography .....	40
Appendix A – Data Formatting Tool Code .....	41
Appendix B – Application Code.....	44
Appendix C – Application Unit Test Code .....	45

## Abstract

For many people, mastering an additional language is a rewarding and challenging experience. Throughout this project, I explore the core aspects of effective language learning and how computers can facilitate rapid vocabulary acquisition. I developed a web based PHP application for learning new words as “words are the fundamental building blocks of every language and to communicate at any level, one must have a sufficient vocabulary” (Godwin-Jones, 2010).

This report will consider my contribution in the wider context of computer assisted language learning, spaced repetition, and deliberate practice. Furthermore, I will examine how my application was built, what has been gained from the experience and areas for further research and development.

Supervisor: Professor George D. Magoulas

# 1. Introduction

It has been shown (Schmitt, 1997) that the 1000 most frequent words account for 85% of speech. Mastery of the 1000 most common words should allow users to tackle more advanced material with greater confidence. For this reason, developing an accessible computer system for learning frequently used words has value.

## 1.1 Background

Technology can play an important role in learning, offering an excellent platform for implementing effective learning paradigms. For this project, I will implement the core elements that define ‘spaced repetition’ and ‘deliberate practice’.

Spaced repetition is a technique used to specify how often material being learned is reviewed, relative to how easily it was recalled. Spaced practice facilitates a more varied learning experience and “the variable contexts that are stored in memory then serve as more effective cues for subsequent retrieval of the item” (Kang, 2016).

“Deliberate practice is a highly structured activity, the explicit goal of which is to improve performance. Specific tasks are invented to overcome weaknesses, and performance is carefully monitored to provide cues for ways to improve it further.” (Ericsson, 1993, p. 368).

## 1.2 Motivation

Rapid developments in computer hardware, software and increased access to computing devices have been a driving force in the growth of computer assisted teaching and learning. Human languages contain vast numbers of words and subtleties that can make language acquisition problematic. Word recognition is just one aspect of true language mastery but “without vocabulary nothing can be conveyed.” (Wilkins, 1972, p. 111).

WEBVOCLE, an application specifically developed to test to a web-based spaced repetition to learn foreign vocabulary, demonstrated that an accessible web-based tool, such as the one I built, can effectively complement other learning practices (Baturay, M. et al, 2009).

Many flashcard style applications implementing spaced repetition algorithms are available, notably Anki and Supermemo. However, the majority are commercial standalone products that require installation and lack the machine independent flexibility of web-based solutions. Existing web-based applications are not currently open to user extension. My application will be freely available and open to extension with an Open Source release.

## 1.3 Aim and Objective

The focus of this project will be:

1. Building a web based application accessible via a standard browser.
2. Building a tool to format my starting data.
3. Exploring the wider context of Computer Assisted Language Learning.

The primary aim of my application is developing user comprehension of written French words. I chose French due to some familiarity with the language and because “French is the second most taught and widespread second language globally, behind English.” (Lonsdale and Le Bras, 2009). This could result in greater interest and use of my application. I chose the 1000 most common French words as my dataset as mastery of these words will provide an easier transition into more challenging material. The dataset was taken from “A Frequency Dictionary of French” as this list was calculated from a combination of popular written texts and transcripts of spoken French, unlike similar lists based exclusively on written texts.

Developing a Moodle plugin for my application, as discussed in my proposal, has evolved from a primary objective to an optional feature. Further investigation into the precise modifications required to convert my PHP application into an acceptable format for Moodle, revealed significant time would be needed for learning new technologies, in addition to the time required for setting up the Moodle platform.

Although web-based and therefore somewhat compatible with mobile devices, I am restricting my development and testing to desktop browsers as many mobile devices have restrictions on JavaScript usage.

## 1.4 Development Process

I will manage the lifecycle of this software application using Agile, an iterative approach. I decided on this rather than the traditional Waterfall design process with its emphasis on planning upfront, as delayed implementation is less suited to the multiple uncertainties associated with using unfamiliar technologies.

In keeping with the Agile approach, I will focus on building usable, working, and tested software at each iteration. Functionality will be added piece by piece, which has the benefit of tangible feedback as my system is being built. An Object-Oriented style will be used throughout the implementation of this project, due to my familiarity with the paradigm.

## 1.5 Development Environment

I decided to use PHP 5.6, rather than the newer PHP 7 for the main application as version 5.6 is well supported on the university machines and on my personal webspace.

For the client side processing, I have used the latest JavaScript version 1.8.5 and jQuery 3.2.1, which is the latest version of the library hosted online. For simplicity, I chose AMPPS to host my application during development, as minimal setup is required. PHPUnit was my testing Framework, GitHub was used for version control and PhpStorm is my preferred IDE. The text formatting tool was developed in Java 8 using IntelliJ IDEA due to my familiarity with Java's text manipulation functionality.

## 1.6 Report Structure

The structure of my report is as follow:

Chapter 2 begins with a brief history and review of Computer Assisted Language Learning, Spaced Repetition, and Deliberate Practice. I will then consider how this understanding can be implemented in my own application and the contribution it makes.

Chapter 3 will detail my approach towards gathering and analysing precisely what is required of my system.

Chapter 4 considers the importance of both Architectural and Visual Design, presenting the choices I made and the reasoning behind them.

Chapter 5 describes how core areas of my system were developed and why certain methods were chosen for implementation.

Chapter 6 begins with a brief discussion of testing and its importance. This includes testing done for this project and opportunities for further testing.

Chapter 7, the final section, will evaluate the success of my project.

## 2. Literature Review

In this Literature Review, I will consider my project in the wider context of Computer Assisted Language Learning (CALL), Spaced Repetition, Deliberate Practice, and similar software applications. I will outline what is already known and understood, identify areas that would benefit from further research and examine the contribution of this project.

Documented examples of Computer Assisted Language Learning first appeared in the 1960s and 1970s at a number of academic institutions globally. These include the 'Branched Program Achievement Test' (Boyle, T. A., Smith, W. F. and Eckert, R. G., 1976) and several projects at Stanford that mathematically analysed learners past performance and material difficulty in order to schedule further instruction (Atkinson, 1972). Systems such as these did not become widely available until the 1980s, when the cost of computing technology had reduced to a level where individual schools could purchase computers that would allow them to explore the potential of this new technology in language education.

The new technology was not initially embraced, as the much of the software did not reflect the importance of "unconscious acquisition", an idea discussed at length by the popular linguist, Professor Steven Krashen, in his 1982 book 'Principles and Practice in Second Language Acquisition'. In response to this, language learning software shifted away from testing drills and towards natural language processing that attempted to correct users attempts at communicating in the target language.

'Computer-adaptive testing' was the first large movement towards analysing user responses statistically to improve the selection of future questions of an appropriate difficulty (Green, 1984). By the 1990s, with the increased popularity of the internet, many language systems shifted towards more collaborative software and more complex user input. Computational linguistics was used to process natural language in conjunction with computer assisted assessment for more advanced scoring and reporting.

A common theme throughout the research in Computer Assisted Language Learning is the importance of adjusting difficulty according to a user's proficiency. How difficulty and proficiency was measured varied throughout the research and in many existing software applications, such as Anki, the level of difficulty is self-assessed. As a step towards a fairer system, I will assess proficiency according to the amount of time taken to provide a correct response, less time being an indicator of higher proficiency.

Evidence supports the notion that computers can be beneficial in language acquisition. However, I do not believe current natural language processing technology has reached a level sufficient to replace traditional ways of learning and practice entirely.

Although there is much agreement within CALL research that computers play an important role in language learning, there is still debate on "what constitutes good material" and "how such materials can be evaluated" (Sharma, 2008)

Spaced repetition is a core aspect of my application design due to the extensive literature supporting its effectiveness. The potential of spaced repetition to enhance learning was first discussed in the 1932 book 'Psychology of Study' that stated, "Acts of revision should be spaced in gradually increasing intervals, roughly intervals of one day, two days, four days, eight days, and so on." (Mace, 1932).

This notion was revisited in 1939 with a study observing 6th grade students learning science facts (Spitzer, 1939). The findings suggest that "recall aids retention" and "tests are learning devices" when appropriately spaced. Additionally, "using substantial spacing between tests (at least when these are accompanied by feedback) is likely to provide a stronger foundation for optimizing learning than the intuitively appealing idea of arranging conditions to minimize the occurrence of errors." (Pashler, 2003).

Although there is a wealth of research supporting the effectiveness of spaced repetition, it is not without its limitations. As spacing is increased, test performance improves up to a certain point before gradually decreasing (Toppino, 1985). One explanation of this fact suggests increased spacing causes the initial memory trace to become less accessible.

To fulfil my primary objective of effective vocabulary acquisition, developing a system that facilitates what has been described as 'deliberate practice' is essential. The term 'deliberate practice' refers to "those activities that have been found most effective in improving performance" (Ericsson, 1993, p. 367) and certain conditions must be considered to meet these criteria. The primary characteristics to include in any regime attempting to nurture expert performance are:

1. It is designed specifically to improve performance.
2. It can be repeated.
3. Feedback on results is continuously available.
4. It is mentally challenging.
5. Motivation will come from progress not fun.

(Colvin, 2008)

Using the SM2 algorithm ensures the material is reviewed with timing specifically optimised to improve retention (Folksman 2013) and practice sessions are easily initiated by the user. Feedback is given after every response and time constraints will encourage users to stay focused and challenged. Improved comfort with other language materials after sustained use of my application should maintain motivation for its continued use.



### 3. System Requirements

Initially, I began coding application features without a clear structure in mind to ensure early on that the required functionality was feasible to implement with my chosen languages in the time available. Although I rapidly built working software, it quickly became apparent that some form of clear structure was imperative to build a system that is testable and maintainable. I then began researching different approaches to Software Architecture.

It has been shown that 80% of software projects do not produce expected results, go over time or budget (Standish Group, 1994). Making the effort to avoid these consequences must be a consideration in all projects. Often this starts with effective requirements analysis; collecting precise expectations of a system from users, customers and any other stakeholders.

Many approaches are used to gather requirements but there are some key issues to consider (McLaughlin, 2007). Clarity of scope is essential to define the boundaries of what exactly a project does and does not cover. Invariably, concise and highly specific objectives encourage less interpretation of functionality and fewer assumptions.

Short specific user stories will be written to capture user requirements, i.e. how my application will be used by learners and administrators and the features that are required. Once the user stories are complete, discrete use cases can be written to define the processes being modelled, how the various elements will interact to meet the stated functionality and specific technical requirements.

I will also briefly discuss why each feature is required and the consequences of its implementation. Each iteration will take one or two features through Analysis, Design, Coding and Testing. Requirements will be adjusted as needed.

User stories will be prioritised in a 'master story list' before work commences and will form the basis of my plan.

## 3.1 User Stories and Use Cases

### 3.1.1 Personas

The student and main user persona is the person who will use the application to develop their vocabulary.

The administrator persona will manage user accounts.

### 3.1.2 Student stories

As a student, I can create a new account.

I can choose my own username and password.

As a student, I can log in and out of a personalised area.

As a student, I can change my password.

As a student, I can import a set of questions to study.

I can upload this as an XML or CSV file

As a student, I can remove a set of questions from my collection.

As a student, I can view the questions and answers of a set in my collection.

As a student, I can reset my study history for a set of questions.

As a student, I can test my knowledge of new material and questions scheduled for review that day.

As a student, I can test new words after the questions scheduled for that day have been answered.

### 3.1.3 Administrator stories

As an Administrator, I can delete a user account, including the uploaded material.

As an Administrator, I can log in and out of the Administration Area.

As an Administrator, I can change the password used to access the Administration Area.

### 3.1.4 Use Cases

**Title:** Create User Account

**Description:** The Student creates a new account for the system.

**Primary Actor:** Student

**Preconditions:** The start page is visible.

**Postconditions:** The Student is registered in the system and can log into their account.

**Main Success Scenario:**

1. Student clicks “New User” from the main menu.
2. System displays a form asking the user to enter their desired username and password.
3. Student enters their desired username and password.
4. Student clicks “Create User”
5. System registers this new user in the XML file of usernames and passwords.
6. A confirmation message is displayed with a button that returns to the log in page.

**Extensions:**

1. Username entered is already taken  
Display error message and ask the student to choose another name.

**Priority:** Medium

**Notes:** Having user accounts allows a customised student experience.

**Title:** Log in to account

**Description:** The Student logs in to their account for the system.

**Primary Actor:** Student

**Preconditions:** The Student has created a user account.

**Postconditions:** The Student is logged in to the user area of the system.

**Main Success Scenario:**

1. Student enters their username into the input box provided.
2. Student enters their password into the input box provided.
3. Student clicks "Log in"
4. System verifies username exists and the password is correct.
5. System creates a session variable holding the username of the user that has logged in.

**Extensions:**

1. Log in fails  
Display error message stating why the log in has failed and present the student with an opportunity to try again.

**Priority:** Medium

**Title:** Log out of account

**Description:** The Student logs out of their account for the system.

**Primary Actor:** Student

**Preconditions:** The Student is currently logged in to the system

**Postconditions:** The Student is at the log in page and logged out of the system.

**Main Success Scenario:**

1. Student clicks the log out button on a page in the user area.
2. System unsets the session variables and returns the user to the log in page.

**Extensions:** None

**Priority:** Medium

**Title:** Change Student password

**Description:** The Student changes the password required to log in to their account.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** The Student must enter the newly chosen password to log in.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page.
2. Student clicks "Set Password"
3. System displays a form asking the user to enter a new password.
4. Student enters their new password.
5. Student clicks "Update Password"
6. System registers this new password as the logged in Student's new password in the XML file of usernames and passwords.
7. A confirmation message is displayed with a button that returns to main user page.

**Extensions:**

1. The new password entered is the same as the existing password.  
    Display error message stating the password entered is the same as the current password and no updates have taken place.

**Priority:** Medium

**Title:** Import XML Deck

**Description:** Student adds a new set of questions and answers to their collection.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** A new set of questions and answers is available for testing.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page.
2. Student clicks "Import Deck".
3. Student is presented with a choice of importing an XML deck or CSV deck.
4. Student clicks 'Import XML Deck'.
5. Student is presented with a button to browse their hard drive for an XML file.
6. Student selects a valid XML file.
7. Student clicks 'upload file'.
8. System adds the XML file to Student's data folder
9. Upload confirmation screen is presented with the option to return to the main user menu.

**Extensions:**

1. Selected file is not an XML file  
Ask user to try another file.

**Priority:** Medium

**Title:** Import CSV Deck

**Description:** Student adds a new set of questions and answers to their collection.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** A new set of questions and answers is available for testing.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page.
2. Student clicks "Import Deck".
3. Student is presented with a choice of importing an XML deck or CSV deck.
4. Student clicks 'Import CSV Deck'.
5. Student is presented with a button to browse their hard drive for an XML file.
6. Student selects a valid CSV file.
7. Student clicks 'upload file'.
8. System converts the selected CSV file to XML and adds the XML file to the Student's data folder.
9. Upload confirmation screen is presented with the option to return to the main user menu.

**Extensions:**

1. Selected file is not an XML file  
Ask user to try another file.

**Priority:** Medium



**Title:** Delete User Deck

**Description:** Student deletes a set of questions and answers from their collection.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** The Student no longer has the deck deleted available for testing.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page.
2. Student clicks "Choose Deck".
3. Student clicks on the name of the deck they wish to delete.
4. Student clicks 'Remove This Deck'.
5. System removes the deck XML from the Student's data folder.
6. A confirmation message is displayed with a button that returns to main user page.

**Extensions:** None

**Priority:** Medium

**Title:** View Questions

**Description:** Student views a table of all the questions and answers in the selected deck.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** Questions and Answers have been viewed.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page with main menu.
2. Student clicks "Choose Deck".
3. Student clicks on the name of the deck they wish to view.
4. Student clicks 'View Questions'.
5. System generates a table of questions and answers of the selected deck and a button that returns to main user page.

**Extensions:**

XML File is badly formatted  
Display error message.

**Priority:** Medium

**Notes:** Allows user to browse unknown material before testing.

**Title:** Reset Study History

**Description:** Student resets the review data of the selected deck. Studying this deck again will commence as if the first time.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** The selected deck has no review history.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page.
2. Student clicks "Choose Deck".
3. Student clicks on the name of the deck they wish to reset.
4. Student clicks 'Reset Deck'.
5. System resets all the viewing and testing history to their starting values.

**Extensions:** XML File is badly formatted  
Display error message.

**Priority:** Medium

**Title:** First Testing Session of the day

**Description:** Student tests their knowledge of 10 new questions and any questions scheduled for review today.

**Primary Actor:** Student

**Preconditions:** The Student has already created an account and is logged in.

**Postconditions:** 10 new questions have been answered correctly and review questions scheduled for today have been answered correctly.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page with main menu.
2. Student clicks "Choose Deck".
3. Student clicks on the name of the deck they wish to test.
4. Student clicks 'Start Testing'.
5. System retrieves 10 untested questions to test or as many as available.
6. For each new question retrieved, the user is presented with a word to translate and a box to enter their response.
7. System resets a ten second countdown timer when each question is asked.
8. User enters a correct response.
9. System updates the number of times this question has been viewed, verifies the answer is correct and takes the speed of response to use in calculating when this question will be asked again according to the SM2 algorithm.
10. When all the new questions have been asked, all questions marked for review today will be asked.
11. Once all the review questions have been answered correctly the user is presented with a screen that confirms the testing session is complete with an option to return to the main user menu.

**Extensions:** Student answers incorrectly

The wrongly answered question is added to the list of questions to be reviewed today and the next new question is asked. If all the new questions retrieved have been asked, the first review question will be asked.

**Priority:** High

**Notes:** Update interface using AJAX to avoid visually jarring page refreshes.

**Title:** Additional Testing Session

**Description:** Student tests their knowledge of 10 additional new.

**Primary Actor:** Student

**Preconditions:** The Student has completed their first test session of the day including all the cards marked for review today. The student has already created an account and is logged in.

**Postconditions:** 10 additional questions have been answered correctly.

**Main Success Scenario:**

1. Student is logged in and viewing the first user page with main menu.
2. Student clicks "Choose Deck"
3. Student clicks on the name of the deck they wish to test.
4. Student clicks 'Start Testing'.
5. System retrieves 10 untested questions to test or as many as available.
6. For each new question retrieved, the user is presented with a word to translate and a box to enter their response.
7. System resets a ten second countdown timer when each question is asked.
8. User enters a correct response.
9. System updates the number of times this question has been viewed, verifies the answer is correct and takes the speed of response to use in calculating when this question will be asked again according to the SM2 algorithm.
10. When all the new questions have been asked, any that were not answered correctly will be asked again as review questions until all questions have been answered correctly.
11. Once all the new questions have been answered correctly first time or as review questions, the user is presented with a screen that confirms the testing session is complete with an option to return to the main user menu.

**Extensions:** None

**Priority:** High

**Title:** Log in to Administration Area

**Description:** Administration Area.

**Primary Actor:** Administrator.

**Preconditions:** Administrator knows the required password and is at the first screen.

**Postconditions:** Administrator is logged in to the Administration Area of the system.

**Main Success Scenario:**

1. Administrator clicks "Tools".
2. Administrator enters the Administrator password into the input box provided.
3. Administrator clicks "Log in".
4. System verifies the password is correct.

**Extensions:**

1. Log in fails  
Stay on the Tools Log in page.

**Priority:** Medium

**Title:** Delete User Account

**Description:** Administrator deletes a Student account.

**Primary Actor:** Administrator

**Preconditions:** The Administrator is logged in to the Administration Area.

**Postconditions:** The selected Student is deleted from the database of Students and their deck data is removed.

**Main Success Scenario:**

1. Administrator clicks "Delete Student"
2. Administrator clicks on the name of the Student they wish to delete.
3. Administrator clicks "Remove This Student".
4. System removes the Student from the XML of users and deletes their data folder.
5. A confirmation message is displayed with a button that returns to main Administration Area.

**Extensions:** None

**Priority:** Medium

**Title:** Change Administrator password

**Description:** Administrator changes the password required to log in to the Administration Area.

**Primary Actor:** Administrator

**Preconditions:** The Administrator is logged in to the Administration Area.

**Postconditions:** Administrators must enter the newly chosen password to log in to the Administration Area.

**Main Success Scenario:**

1. Administrator clicks "Set Password".
2. System displays a form asking the Administrator to enter a new password.
3. Administrator enters their new password.
4. Administrator clicks "Update Password".
5. System sets this new password as the password in the XML file with the Administrator Password.
6. A confirmation message is displayed with a button that returns to the Administration Area.

**Extensions:**

1. The new password entered is the same as the existing password  
Display error message stating the password entered is the same as the current password and no updates have taken place.

**Priority:** Medium



## 4. System Design

In this chapter, I consider the importance of both Architectural and Visual Design, explaining the choices I made and the thinking behind them. Due to the limited time available for this project, I decided to focus my efforts on developing a stronger application and an underlying framework, keeping an open source release as a future goal.

### 4.1 Architecture

Software Architecture embodies the first set of decisions in a systems design. Making these decisions well is vital because changing them later in the development process becomes increasingly hard as more elements are built on top.

Formal descriptions for a number of architectural patterns are readily available and a well-designed system will align with the elements and interactions presented in the chosen pattern. Good architecture allows easier modification and extension with components that take discrete responsibilities, limiting the reach of changes made.

A clear architecture allows a system to be built incrementally with the potential of reusing elements, in keeping with my chosen Agile paradigm. Additionally, communication between elements can be managed more easily as a step towards greater security.

Although architecture has far reaching consequences, it cannot guarantee good software and that good decisions will be made throughout the software development lifecycle. “Good architecture is necessary, but not sufficient” (Bass, 2012).

By consciously limiting my programs elements and their interactions, the complexity of implementation is reduced. Although the types of elements will be limited, I maintain the flexibility of unique implementations for individual elements.

A number of well documented architectural patterns could be used to organise my code. However, for this project, I will consider the five types that are most widely used (Richards, 2015). These are:

1. Layered Architecture
2. Event-Driven Architecture
3. Microkernel Architecture
4. Microservice Architecture
5. Space-based Architecture

Of the five pattern types being outlined, layered architecture is probably the most common. Layered architectures are built with databases in mind and many of the most popular frameworks share this design. Data, such as user input, typically arrives at the top layer and works its way down to some form of database. Each layer will have a clear responsibility and could be developed in parallel.

Model View Controller (MVC) is one popular implementation of this and can be summarised as follows.

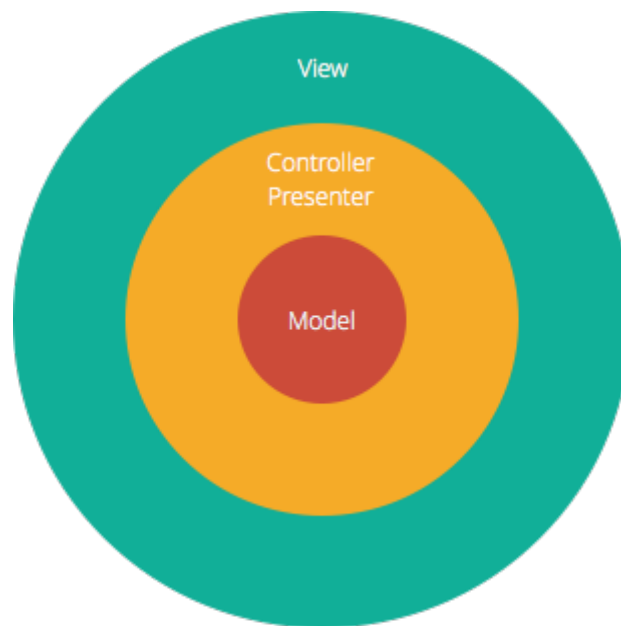


Figure N

The outermost or top layer called the View typically contains code managing what the user interacts with. In the context of web applications, this may include HTML with dynamic code customising the precise output, JavaScript and CSS.

Underneath the View is the Controller layer. This layer receives data from the View and decides what action needs to be taken. This may include updating the View and Model layers with modified data.

At the bottom is the Model layer which contains methods handling the interaction with the database below.

The major advantage of layered architectures is the clear separation of responsibilities, meaning updates to system logic don't need to interfere with display concerns, a change of database or vice versa. This makes for a system that is more easily built, tested, and maintained.

Although a strong and popular approach, it is not without disadvantages. Isolating layers means that each module must be clearly understood in order to understand the architecture as a whole.

Event-driven architecture is based around a central module receiving data before passing it on to a sub module built to handle that particular type of data. The process of passing the data on to a dedicated module can be thought of as an 'event'. With this in place, only directly relevant code ever sees an event unlike a layered approach where inputs have contact with

each layer. Scalability is a major advantage of an event-driven approach as new event types only require a new module to build and a minor update to the receiving module. One thing that must be considered, especially when modules interact with one another, is how interactions will be tested before a fully working system is available and how the main module will respond if a submodule fails.

The Microkernel pattern is based around a simple core module (Microkernel) and surrounding modules that 'plug-in' to it. The core contains key functionality that will be used frequently with additional modules handling specialist tasks.

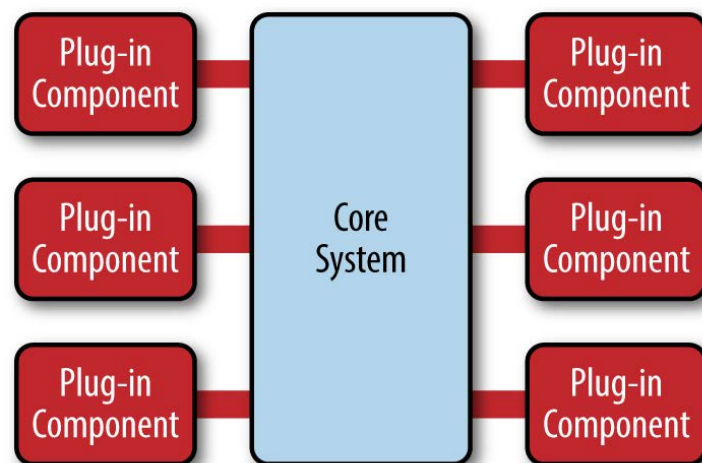


Figure N

Deciding precisely which functionality should be placed in the Core System is not always clear and modifying the Microkernel becomes increasingly difficult as more and more plugins rely on it. For this reason, this architecture is best reserved for implementations that clearly divide basic processes and more complex routines.

Microservice architecture divides a problem into multiple small programs each performing a single task. This approach is particularly useful for cloud based systems where some tasks are required far more often than others. This separation and independence between services allows for better scaling of resources to where they are required.

After considering the approaches outlined, I decided that a layered approach was most suited to my needs. It provides enough structure to facilitate a rapid iterative development process, good support for managing data access and is simple enough to learn quickly as a new paradigm. Specifically, I will use the well-established and well documented MVC pattern with multiple classes at each level to handle different data types e.g. a 'Log in Model', 'Log in View' and 'Log in Controller' will handle functionality related to the user log in process.

A number of established MVC frameworks are already available for PHP such as Zend, Laravel and CakePHP. However, I felt this project presented a unique opportunity to learn this popular design approach more deeply by building my own framework. With my proposal objective of

developing a Moodle plugin reprioritised to a future extension, more time had become available. Additionally, my own framework could be customised to my specific needs and unnecessary complexity could be more easily avoided.

## 4.2 Usability and Visual Style

Design aesthetics play an important role in making a system accessible. If people don't know how to interact with a system they will not come back to it. For this reason, I have prioritised making my application as self-explanatory as possible, with headings and interactive buttons clearly indicating what users can and are most likely to want to do. A specific interface and colour scheme have been chosen. Green indicates the most likely action, Blue indicates important information on what to do and red shows that an action needs consideration, such as deleting. The option to log out will be present on pages where the user is logged in. All the style information in the application has been defined using CSS.

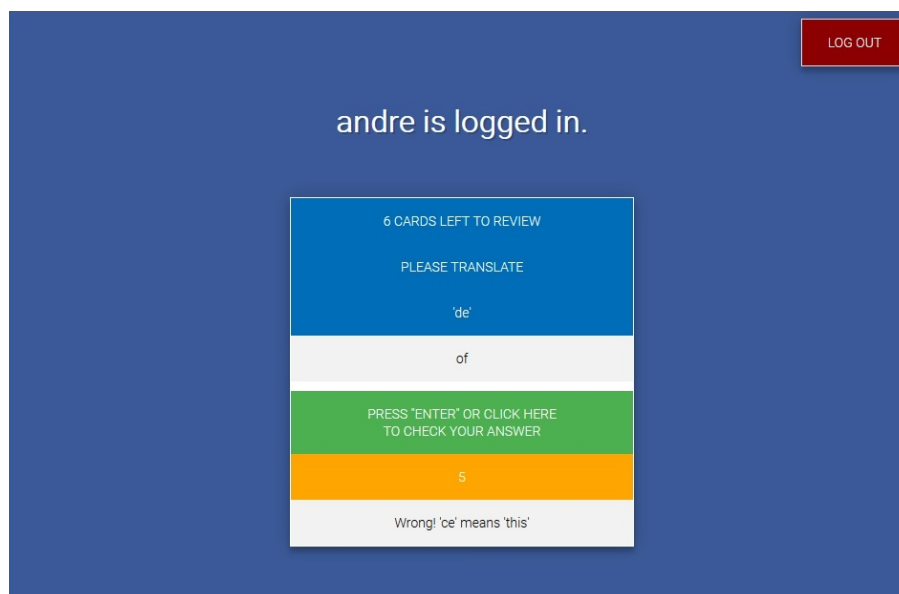


Figure N

## 5. Implementation

This chapter presents how the main system components were developed.

### 5.1 Data Formatting Tool

As mentioned previously, I decided to use the 1000 most frequent words in French as my initial data set. This list is freely available from multiple sources and I have some familiarity with language. The specific text selected was taken from the book 'A Frequency Dictionary of French'. This version has the words ordered from most common to least common with each word followed by its definitions on a single line. It also has the advantage of being in a digital format that can be copied and pasted in to a text file.

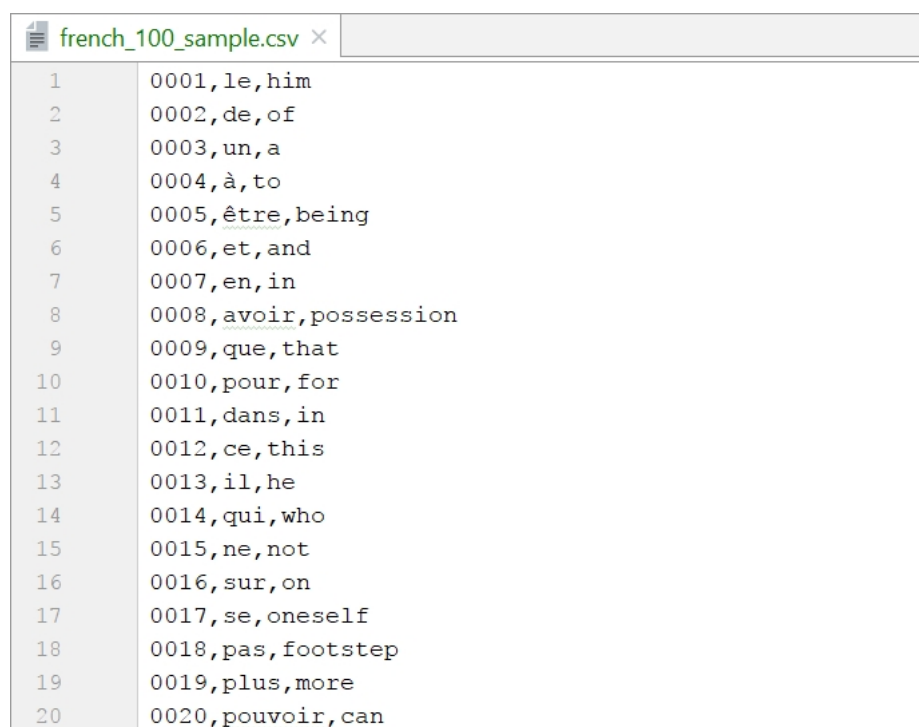
#### Frequency index

**rank frequency (501, 502...), headword, part of speech, English equivalent**  
• sample sentence  
range count | raw frequency total, indication of major register variation  
**1 le** *det, pro* the; him, her, it, them  
• vive la politique, vive l'amour – *long live politics, long live love*  
89 | 2359662  
**2 de** *det, prep* of, from, some, any  
• il ne rêve que d'argent et de plaisirs – *he only dreams of money and pleasure*  
88 | 1665907  
**3 un** *adj, det, nm, pro* a, an, one  
• je me suis cassé un ongle – *I broke one of my fingernails*  
95 | 421500  
**4 à** *prep* to, at, in  
• ils restent à l'école le plus longtemps possible – *they remain at school as long as possible*  
93 | 557546  
**5 être** *nm, v* to be; being  
• tout le monde veut être beau – *everybody wants to be beautiful*

Figure N

I wanted to store the question and answer data in XML files as PHP has good tools for managing XML content with SimpleXML. However, this tool will output a CSV file from my starting content that was copied and pasted into a text file. The main application will then import and do the conversion of the CSV file into an XML file. Having the functionality to import CSV files as part of the main application also means that users can easily make their own CSV files in a text editor or spreadsheet. CSV is probably the simplest file format used in representing table data. Essentially, each line in a text file can be considered a row and columns are defined by placing commas between values.

My program begins by loading the source text file and skipping past any UTF BOM characters present. To preserve the accented French characters (à, é, ï, ô, etc) my text file was saved with UTF-8 encoding and files with this encoding contain hidden characters at the start of the file to signal the encoding used. Once this has been done the program traverses through the file line by line, stopping on lines that begin with a digit to extract the frequency rank, French word, and first English definition. With these elements extracted, a line for the future CSV file is created and added to an array of future CSV lines. Once all the lines have been inspected, the array of future CSV lines is sorted by rank with the most common word first in the list. Sorting the list ensures the most useful words will be tested first in the main application. This now sorted list of lines is then written out line by line to a new CSV file.



french_100_sample.csv ×		
1	0001,	le, him
2	0002,	de, of
3	0003,	un, a
4	0004,	à, to
5	0005,	être, being
6	0006,	et, and
7	0007,	en, in
8	0008,	avoir, possession
9	0009,	que, that
10	0010,	pour, for
11	0011,	dans, in
12	0012,	ce, this
13	0013,	il, he
14	0014,	qui, who
15	0015,	ne, not
16	0016,	sur, on
17	0017,	se, oneself
18	0018,	pas, footstep
19	0019,	plus, more
20	0020,	pouvoir, can

Figure N. CSV Output

## 5.2 Framework Development

The first task in developing my framework was clearly defining the functionality needed to develop my application. The framework requirements were as follows:

1. A way to organise code in Model, View and Controller classes and a clear separation between logic and presentation.

‘Model’ meaning anything modifying saved data.

‘View’ meaning code related to user input and display.

‘Controller’ meaning code defining the systems logic.

2. A way to pass data between elements.

=

3. Added security.

These considerations will facilitate more organised code that is easier to develop, understand and maintain.

I began my framework by creating the folder structure

The ‘app’ folder will contain folders to house the model, view and controller classes as well as the XML data stored. The core functionality of the framework will be in the ‘core’ folder and the ‘public’ folder will hold any publicly available non-dynamic files. Browser access to folders other than ‘public’ will be restricted using ‘.htaccess’ files, adding a layer of security to sensitive information like passwords.

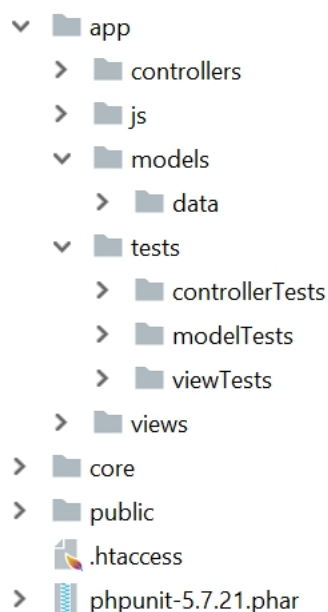


Figure N. Folder Structure

After the folders were created, statements were added to the '.htaccess' file in the root directory to redirect users to the 'index.php' file in the public folder, the new root of the application.

The first element created for my framework is the main entry point of the system, often known as a 'front controller'. This allows PHP code to run indirectly without needing to access files directly through the file system. Using my front controller ensures that all URLs are processed through 'index.php' in the public folder. Essentially, the front controller will decide which class and method will be used and the associated parameters, all from the query string. The query string is the text following 'index.php?' in the browser address and can be considered a 'route' to the correct method to execute, including its arguments. To simplify the routes further, the rewrite rules of the server were modified in the root '.htaccess' file to allow what are often called 'pretty urls'



```
.htaccess x
1 # Start at index.php in public directory
2 DirectoryIndex public/index.php
3
4 # Allow 'pretty' URLs
5 RewriteEngine On
6 RewriteBase /
7 RewriteCond %{REQUEST_FILENAME} !-f
8 RewriteCond %{REQUEST_FILENAME} !-d
9 RewriteCond %{REQUEST_FILENAME} !-l
10 RewriteRule ^(.*)$ mvc-shuffle/public/index.php?$1 [L,QSA]
```

Figure N. Rewrite Rules

Once the entry point of the framework was established, it was necessary to define how the query string from the browser would define which class and method was required, including the arguments to be passed in. This is the most fundamental ability of the framework and has been defined in the 'Router' class inside my 'core' folder. I decided to use the format '/controller-name/method-name' for my query strings and in keeping with popular PHP naming convention, classes will be defined in 'Studly caps' such as 'UserModel' where the first letter of each word is capitalised and methods will be defined in 'camel case' such as 'resetStats' where names begin lower case and new words begin with a capital letter. For these reasons, some name translation will be required.

The responsibilities of the Router class are as follows:

1. Extract the class and method name from the query string.
2. Extract the arguments to pass into the method.
3. Execute the correct method with the correct parameters (routing).
4. Convert hyphenated name to camel case or Studly caps.



Much of the Router logic was defined using pattern matching and regular expressions. All of the URL requests will pass through the root index.php and will be processed by a Router object instantiated here. Additionally, the 'index.php' loads the classes in the system so their methods are available.



```
1 <?php
2
3 // Front Controller, every request goes through here.
4 // Route comes in through the url query string and is matched with the correct method
5 // paths are relative to this index.php
6
7
8 // Require all classes in a given folder
9 function requireClasses($src) {
10
11     foreach (glob($src) as $filename) {
12         require $filename;
13     }
14 }
15
16 // Load Classes
17 requireClasses( $src: "../core/*.php");
18 requireClasses( $src: "../app/controllers/*.php");
19 requireClasses( $src: "../app/models/*.php");
20
21 $router = new Router();
22
23 $url = $_SERVER['QUERY_STRING'];
24 $router->dispatch($url);
25
```

Figure N

With this in place, several Model, View and Controller classes were created according to the type of tasks they were responsible for. All the data modification is handled by methods in Model classes, system logic is detailed in Controller classes and the user input and interface is defined in View classes. Methods can pass values between each other by sending an appropriate URL to the browser.

## 5.3 Application Development

In this section, I will describe how the core aspects of my application were implemented, starting with spaced repetition. In this discussion, the term 'card' refers to a single question and answer pairing and the term 'deck' refers to a collection of 'cards'. I decided to schedule questions for review using the SM2 spaced repetition algorithm as it is open source and well documented. Although few studies have occurred, the effectiveness of the SM2 algorithm has been tested at length by P.A.Wozniak (1998) and was addressed in a study by D. Folksman in 2013. Spaced Repetition was the first and perhaps most important feature to implement once my framework was in place. To use the SM2 algorithm, one must track the number of times a card has been reviewed and an 'easiness factor' for each card. The easiness or 'E-Factor' is a floating-point value that represents how easy a given question is to answer with values starting at 2.5 and dropping no lower than 1.3 for the most difficult questions. The E-Factor will gradually decrease for cards the student is struggling to recall correctly.

The SM2 algorithm will be used as follows:

1. Ensure all cards start with an E-Factor of 2.5.

2. 'n' is the number of times a card has been viewed:

if  $n = 1$ ; review card tomorrow.

if  $n = 2$ ; review card in 6 days.

if  $n > 2$ ; review card in  $((n - 1) \times (\text{E-Factor}))$  days.

If the number of days calculated is fractional, round it up to the nearest integer.

3. After each answer is received, grade the response with an integer between 0 and 5.

For my implementation, I will grade as follows.

5 – correctly responded within 4 seconds.

4 – correctly responded in 5 to 7 seconds.

3 – correctly responded in 8 or more seconds.

0 – Gave an incorrect answer.

4. After each response, update the E-Factor of the card with the formula:

$$\text{New E-Factor} = (\text{Current E-Factor}) + (0.1 - (5 - \text{grade}) \times (0.08 + (5 - \text{grade}) \times 0.02))$$

If the calculated value is less than 1.3, set the New E-Factor to 1.3

5. Repeat cards graded 0 until they are answered correctly.

6. Increment the number of times a card has been viewed every time it is tested.

(P.A.Wozniak, May 10, 1998)

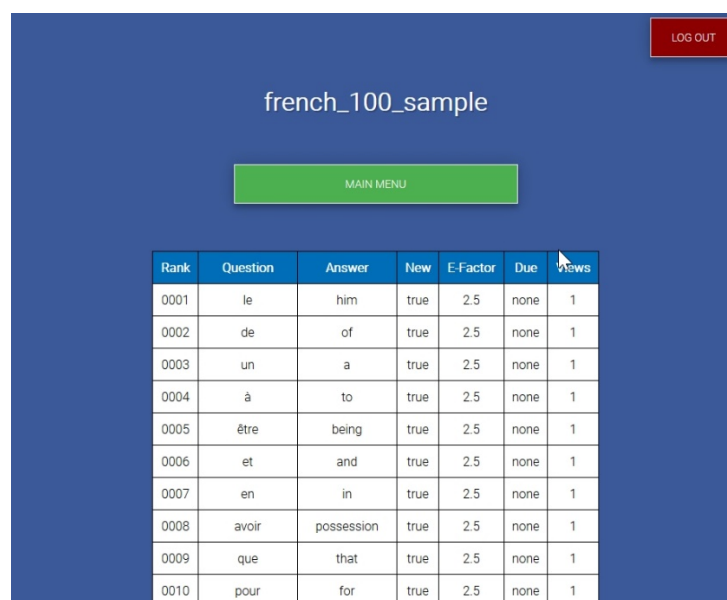
In order for a response to be considered correct, the user must input the expected answer followed by the enter key or a click on the button marked 'check your answer'. Although one

could argue that multiple translations of a word are correct, for the scope of this project, only a single answer is accepted.

The testing page is a combination of AJAX, JavaScript and PHP. AJAX was a key component in implementing the testing mechanic as it allowed me to update the question and feedback areas of the screen without visually jarring refreshes of the webpage. When the user submits a response, the input data is sent using AJAX to a PHP Controller that performs the required updates to the deck before returning appropriate feedback to the user using AJAX. This includes how many cards new or review are remaining, how the user did on the last question and so on. Questions answered incorrectly will have the expected answer revealed in the grey feedback area while the following question is displayed. Correct responses will display positive feedback after submission including how quickly the user responded.

JavaScript was used to decrement the 10 second counter every second and to change the colour as time runs out. This design of counting from green to red was chosen to create a sense of urgency and to keep the user engaged, this supports the notion of deliberate practice.

Importing a CSV file and converting it to a formatted XML file was implemented in PHP using SimpleXML. After the file has been loaded and the encoding signal has been bypassed, each line of the CSV file is inspected in order and the three values are extracted. A new XML element is then created and these three values populate the first three fields of the element namely, 'rank', 'question' and 'answer'. Four additional fields are then added to the element to allow the SM2 algorithm to be used, with their default values, before the next line of CSV values is used to create the next XML element and so on. All the methods modifying the XML files throughout the system used SimpleXML, while the display of XML data used JavaScript and in the case of the "View Deck" Feature an XSLT processor and stylesheet.



Rank	Question	Answer	New	E-Factor	Due	Views
0001	le	him	true	2.5	none	1
0002	de	of	true	2.5	none	1
0003	un	a	true	2.5	none	1
0004	à	to	true	2.5	none	1
0005	être	being	true	2.5	none	1
0006	et	and	true	2.5	none	1
0007	en	in	true	2.5	none	1
0008	avoir	possession	true	2.5	none	1
0009	que	that	true	2.5	none	1
0010	pour	for	true	2.5	none	1

Figure N. View Deck Feature

The user management implemented, as it was not a major focus of this project, has been developed enough to separate students and a single administrator but no further. Passwords

have been stored without encryption and although they are not accessible by casual malicious intent, they are on the server in a folder with '.htaccess' restrictions, storing them as plain text does have security implications. Who is logged in is being tracked with PHP session variables rather than cookies.

## 6. Testing

Testing has been limited to unit testing core class methods as features were developed due to time constraints. Initially, implementation began in a 'Test Driven Development' (TDD) style, writing tests before the related methods. This proved a slower approach and as my time was limited it was decided that each feature would be tested after it had been built. User testing has been performed by myself, family and peers and valuable feedback on usability was a key factor in making the system more self-explanatory.

## 7. Conclusion and Further Work

In conclusion, I feel my project was a worthwhile and rewarding experience. I achieved my primary objective of building a usable and useful application. Working on this project gave me the opportunity to gain a deeper understanding of how applications are built, Agile methodology and how frameworks can be implemented. Additionally, it gave me the opportunity to make a contribution to the development of Computer Assisted Language Learning. By releasing it as an open source project, my hope is that it can be further developed into a more robust tool with a Moodle plugin, serving the educational community. Improved security will be a major factor in future releases, starting with storing passwords as md5 encrypted strings in the next release.

## References

- Ericsson, K. Anders et al. (1993) The role of deliberate practice in the acquisition of expert performance. *Psychological Review*. Vol 100(3). pp. 363-406.
- Kang, Sean H. K. (2016) Spaced Repetition Promotes Efficient and Effective Learning: Policy Implications for Instruction. *Policy Insights from the Behavioral and Brain Sciences* 2016, Vol. 3(1) 12 –19.
- Godwin-Jones, Robert. (2010) Emerging Technologies, From Memory Palaces to Spacing Algorithms: Approaches to second-language vocabulary learning. *Language Learning & Technology*. June 2010, Volume 14, Number 2, pp. 4 – 11.
- Schmitt N. and McCarthy M.(1997) *Vocabulary: Description, acquisition, and pedagogy*. Cambridge: Cambridge University Press.
- Lonsdale D. and Le Bras Y. (2009) *A Frequency Dictionary of French*. Routledge.
- Boyle, T. A., Smith, W. F. and Eckert, R. G. (1976), Computer Mediated Testing: A Branched Program Achievement Test. *The Modern Language Journal*, 60: 428–440. doi:10.1111/j.1540-4781.1976.tb03666.x
- Atkinson, R. C. (1975). Mnemotechnics in second-language learning. *American Psychologist*, 30(8), 821-828.
- GREEN, B. F., BOCK, R. D., HUMPHREYS, L. G., LINN, R. L. and RECKASE, M. D. (1984), Technical Guidelines for assessing computerized Adaptive Tests. *Journal of Educational Measurement*, 21: 347–360.
- Mace C. A. (1932) *Psychology of Study*
- Spitzer, H. F. (1939). Studies in retention. *Journal of Educational Psychology*, 30(9), 641-656.
- Sharma, Pete. "CALL Dimensions: Options and Issues in Computer-Assisted Language Learning A Practical Guide to Using Computers in Language Teaching." (2008): 102-105.
- Pashler, H., Zarow, G., & Triplett, B. (2003). Is Temporal Spacing of Tests Helpful Even When It Inflates Error Rates? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 29(6), 1051-1057.
- Toppino, T. C., & Gracen, T. F. (1985). The lag effect and differential organization theory: Nine failures to replicate. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 11(1), 185-191.
- Colvin, G. (2008). *Talent is overrated: what really separates world-class performers from everybody else*. Finland: Nicholas Brealey Publishing.

Folksman, D. (2013). A Mobile Multimedia Application Inspired by a Spaced Repetition Algorithm for Assistance with Speech and Language Therapy, *Developments in eSystems Engineering (DeSE)*

Baturay, M. et al (2009). Effects of Web-Based Spaced Repetition on Vocabulary Retention of Foreign Language Learners, *Eurasian Journal of Educational Research (EJER)* . Jan2009, Issue 34, p17-36.

Standish Group (2004) The Chaos Report.

Available at: [standishgroup.com/sample\\_research\\_files/chaos\\_report\\_1994.pdf](http://standishgroup.com/sample_research_files/chaos_report_1994.pdf)  
(Accessed 11<sup>th</sup> July 2017)

Bass, Len, Paul Clements, and Rick Kazman (2012). *Software Architecture in Practice*. 3rd ed. Addison-Wesley

Richards, M. (2015), *Software Architecture Patterns*, O'Reilly Media, Inc.

Wozniak P.A. (1999) *Effective learning: Twenty rules of formulating knowledge*. Available at: [supermemo.com/en/articles/20rules](http://supermemo.com/en/articles/20rules) (Accessed 28th March 2016)

McLaughlin, Brett, Gary Pollice, and David West. *Head First Object-Oriented Analysis and Design*. Head First Series. Sebastopol, p 55.



## Bibliography

Chapelle, C. (2007). Computer applications in second language acquisition. Cambridge: Cambridge University Press.

## Appendix A – Data Formatting Tool Code

```
import java.io.*;
import java.util.*;

public class Format {

    public static void main(String[] args) {

        String myFile = "french_full_v01_UTF-8.txt";

        String line = null;
        Set<String> wordSet = new HashSet<String>();

        try {
            File file = new File(myFile);
            FileReader fileReader = new FileReader(file);

            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
                new FileInputStream(myFile), "UTF-8"));

            StringBuffer outputStringBuffer = new StringBuffer();
            outputStringBuffer.delete(0, outputStringBuffer.length());

            while ((line = bufferedReader.readLine()) != null) {

                // Remove UTF BOM Characters
                if (line.startsWith("\uFEFF")) {
                    line = line.substring(1);
                }

                boolean startsWithDigit = Character.isDigit(line.charAt(0));

                if (startsWithDigit) {

                    int spaceIndex = line.indexOf(" ");

                    // Append Rank
                    String rank = line.substring(0, spaceIndex);

                    if (rank.length() == 1) {
                        rank = "000" + rank;
                    } else if (rank.length() == 2) {
                        rank = "00" + rank;
                    } else if (rank.length() == 3) {
                        rank = "0" + rank;
                    }

                    outputStringBuffer.append(rank + ",");

                    // Append French Word
                    outputStringBuffer.append(line.split(" ")[1] + ",");
                }
            }
        }
    }
}
```

```

// Append English Definition
if (line.split(" ")[2].equals("to")) {

    String verb = line.split(" ")[3];

    if (verb.substring(verb.length() - 1).equals(",")) {
        verb = verb.substring(0, verb.length() - 1);
    }
    outputStringBuffer.append(line.split(" ")[2] + " " + verb);
} else {
    String definition = line.split(" ")[2];

    if (definition.substring(definition.length() - 1).equals(",")) {
        definition = definition.substring(0, definition.length() - 1);
    }

    if (definition.substring(definition.length() - 1).equals(";")) {
        definition = definition.substring(0, definition.length() - 1);
    }
    outputStringBuffer.append(definition);
}

wordSet.add(outputStringBuffer.toString());
outputStringBuffer.delete(0, outputStringBuffer.length());
}

}

fileReader.close();

List<String> CSVRows = new ArrayList<String>(wordSet);

// Order the words by rank
Collections.sort(CSVRows);

ListIterator<String> itr = CSVRows.listIterator();

boolean firstIteration = true;
String previousRank = "";

while (itr.hasNext()) {

    if (!firstIteration) {
        previousRank = itr.previous().substring(0,5);
        itr.next();
    } else {
        previousRank = "No previous rank";
    }

    boolean extraDef = false;
    extraDef = itr.next().startsWith(previousRank);

    if (extraDef) {
        itr.remove();
    }
    firstIteration = false;
}

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(
    new FileOutputStream("french_full_v01_formatted.txt"), "UTF-8"));

```

```

        // Write output
        for (String item : CSVRows) {
            System.out.println(item);
            writer.write(item);
            writer.newLine();
        }
        writer.close();

        System.out.println();
        System.out.println("ArrayList Size: " + CSVRows.size());

    } catch (IOException e) {
        e.printStackTrace();
    } catch (NoSuchElementException e1) {
        e1.printStackTrace();
    }
}
}

```

## Appendix B – Application Code

## Appendix C – Application Unit Test Code