



Tribhuvan University
Faculty of Humanities and Social Sciences

A Project report on:
CardsQL
- A flashcard revision/quiz app

Submitted to:
Department of Computer Application,
Himalaya College of Engineering,
Chyasal, Lalitpur

In partial fulfillment of the requirements for the Bachelors in Computer Application

Submitted by:
Sujal Gurung 6-2-378-82-2020
September 28, 2023

Under the Supervision of <Supervisor name>

Abstract

Traditional study techniques of just reading through books, course materials are **passive** and don't engage our minds too much. Students may not be able to retain studied information for long periods of time, due to the way our brains work.

Flashcards are an alternative solution for retaining information by remembering facts or answers to questions. The reasoning being that retrieving previously learned information from our memory engages our brains, strengthens the neural connections to that info and thus, we can remember it easier the next time.

This reflects real world scenarios like exams and tests how much of our learning we actually remember. Practicing flashcards from time to time can be a simple but effective form of revision. CardsQL implements this in an easy-to-use web interface with some additional quality of life features.

Acknowledgement

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Problem Statement	1
1.3	Objectives	1
1.4	Scope and Limitaiton	2
1.4.1	Scope	2
1.4.2	Limitation	2
1.5	TODO Report Organization	2
2	Background Study and Literature Review	3
2.1	Background Study	3
2.1.1	Active Recall	3
2.1.2	Spaced Repitition	3
2.1.3	SM-2 algorithm	4
2.2	Literature Review	5
2.2.1	Study of existing system	5
3	System Analysis and Design	6
3.1	System Analysis	6
3.1.1	Requirement Analysis	6
3.1.2	Feasibiliity Analysis	6
3.1.3	Data Modeling	7
3.1.4	Proces Modeling(DFD)	7
3.2	System Design	8
3.2.1	Architectural Design	8
3.2.2	Database Schema Design	8
3.2.3	Interface Design	8
3.2.4	Physical DFD	9
4	Implementation and Testing	10
4.1	Implementation	10
4.1.1	Tools Used	10
4.1.2	Implementation Details of Modules	10
4.2	Testing	11
4.2.1	Test cases for Unit Testing	11
4.2.2	Test cases for System Testing	11
5	Conclusion and Future Recommendation	12
5.1	Lesson learnt/ Outcome	12
5.2	Conclusion	12
5.3	Future recommendation	12
6	Apendix	13
7	References and Bibliography	14

List of Figures

1	Example Flashcard	1
2	Ebbinghaus' forgetting curve	3
3	Ebbinghaus' forgetting curve countered by Spaced Repetition	3
4	Use case diagram for CardsQL	6
5	Entity-Relationship diagram for CardsQL	7
6	Data Flow Diagram diagram for CardsQL	7
7	Architectural design for CardsQL	8
8	Database schema for CardsQL	8
9	Physical DFD for CardsQL	9

List of Tables

1	Meanings of flashcard rating values	4
---	---	---

1 Introduction

1.1 Introduction

Flashcards are a simple way for students to revise topics. On a piece of paper, we write a question on one side and then the answer on the next. The following is an example of what a flashcard might look like:

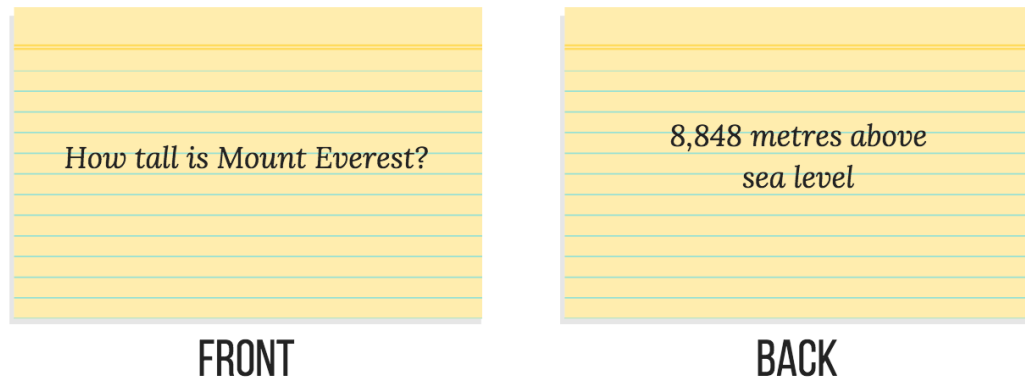


Figure 1: Example Flashcard

To utilize this flashcard, we would read the question & try to remember the answer. Then, flip the card over & see how well we remembered it. CardsQL is a web app that digitises this technique. The name is derived from a combination of Flashcards & SQL.

1.2 Problem Statement

- **inefficient traditional studying techniques**
- **inconvenient to manage physical flashcards**
 - Physical flashcards will eventually get damaged or wear out. It may also be impractical to store or carry around a large number of cards.
 - Storing cards digitally allows them to be physically secure & portable.
- **cumbersome to schedule, keep track of next repetition for cards**
 - After reviewing/ recalling a card, we need to schedule when to practice it again. We not only need to keep track of its schedule, but also remember to review it on that day. If we forget to do so, then keeping track of overdue cards adds another layer of unneeded complexity.

1.3 Objectives

- provide as easy of an entry point to flashcard programs by keeping things simple
- provide intuitive interface so that user doesn't have to understand how the system works in order to utilize it
- automate scheduling of next review. Users just have to use it & practice flashcards regularly.
- allow editing created cards' contents & review date

1.4 Scope and Limitaiton

1.4.1 Scope

CardsQL would be appropriate for the following types of users:

- Students ranging from high schooolers to Doctorates, as they have to understand and memorize a lot of concepts.
- People learning a new language. Applications like **Dulingo**, **Anki** that use question-answer based learning are popularly used for this purpose.
- People, in general, that want to retain their knowledge.

1.4.2 Limitation

- Cards are stored locally on users' computer so there are no cloud backup or multi-device synchronization features. Users can implement it manually if they wish using services like **Dropbox** or **Syncthing**.
- Initial setup process of installing dependencies and running a web server from the command line may be jarring for non-technical users.
- Databse backups haven't been implemented due to time constraints.
- Currently, all flashcards are stored in a single table in the database. This may become unorganized after a large number of cards are added.

1.5 TODO Report Organization

2 Background Study and Literature Review

2.1 Background Study

2.1.1 Active Recall

Cognitive researchers have found that trying to recall facts strengthens the relevant neural connections in our brain & thus, allows us to remember it for longer periods of time. This process is specifically called **Active Recall** & is proven to be more effective than passive studying[1].

2.1.2 Spaced Repetition

Hermann Ebbinghaus, a German psychologist, concluded after extensive research that as time passes, our ability to remember a piece of information slowly decreases[2]. He called this the forgetting curve.

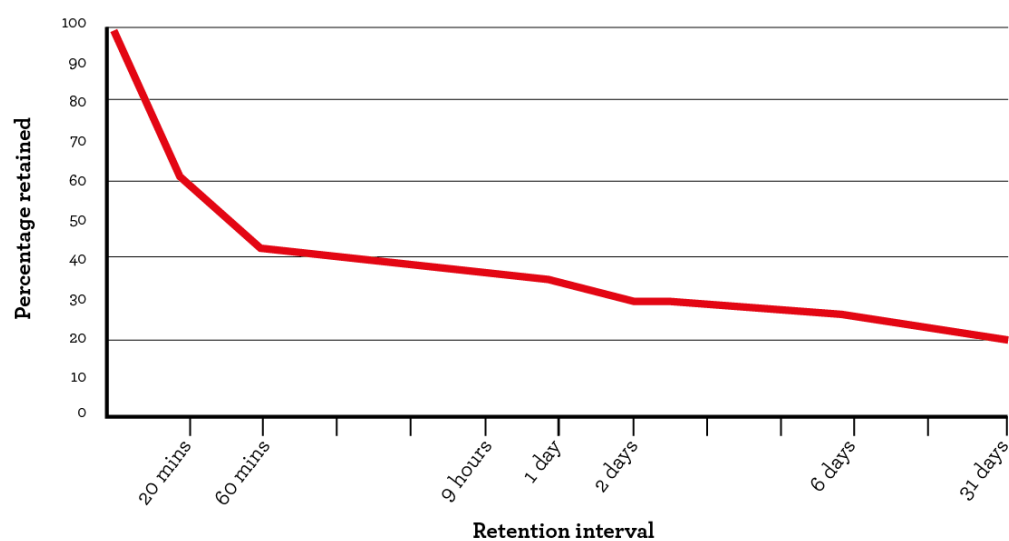


Figure 2: Ebbinghaus' forgetting curve

Ebbinghaus discovered that performing Active Recall at increasing time intervals would increase memory retention & thus, counter the forgetting curve. This is termed as **Spaced Repetition**.

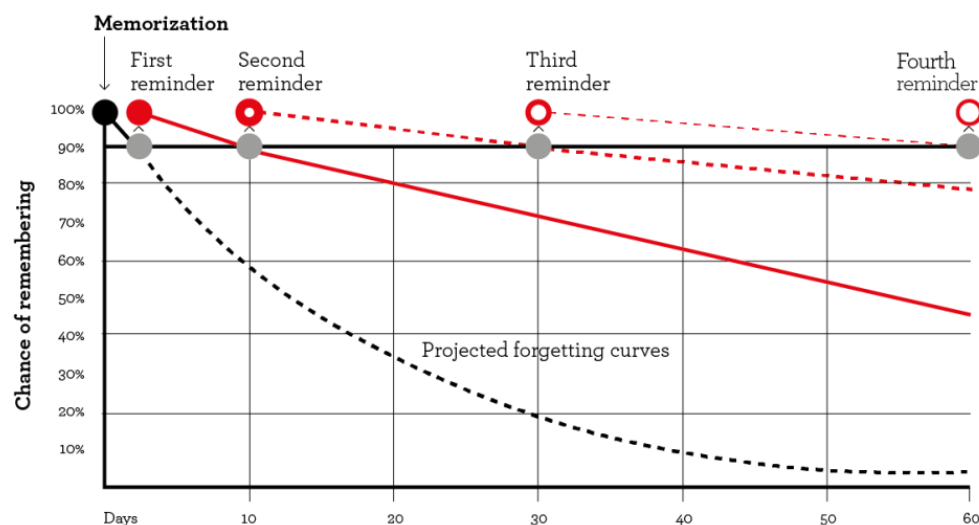


Figure 3: Ebbinghaus' forgetting curve countered by Spaced Repetition

2.1.3 SM-2 algorithm

SuperMemo is a learning software that implements the afore-mentioned methods. It uses an underlying algorithm for calculating when the next repetition/ revision date for a flashcard should be. The algorithm differs across versions but the SM-2 algorithm[3] is popular among other flashcard software. A slightly modified version of SM-2 (used in CardsQL) is described below.

While practicing flashcards, users can rate how well they were able to remember the answer. The rating/ quality of a review is represented by Q .

Table 1: Meanings of flashcard rating values

rating(Q)	meaning
0	Forgot
1	Partially remembered
2	Remembered after some effort
3	Remembered easily

SM-2 also tracks three properties for all cards:

- The repetition number N , which is the consecutive number of times the card has been successfully recalled (meaning $Q \geq 2$). Rating a card below 2 will thus reset N to 0.
- The interval I , which is the number of days after which the card should be reviewed again (to negate the forgetting curve). CardsQL sets next review date = previous review date(today) + I .
- The easiness factor EF , which loosely indicates how "easy" the card is (More precisely, it determines how quickly the interval grows). The initial value of EF for all cards is 2.5.

Due to the formula used, EF value should be ≥ 1.3 so that a card's review isn't scheduled too frequently and isn't too easy. Similarly, it should be ≤ 2.5 so that it isn't scheduled so far into the future that we've forgotten the answer completely by then.

The main algorithm is as follows:

```
input: user rating( $Q$ ), repetition number( $N$ ), easiness factor( $EF$ ), interval( $I$ )
output: updated values of  $N$ ,  $EF$ ,  $I$ 

if  $Q \geq 2$  (i.e. correct response) then
    if  $N = 0$  then  $I = 1$ 
    else if  $N = 1$  then  $I = 6$ 
    else  $I = \text{round}(I \times EF)$ 
    end if
    increment  $n$ 
else (incorrect response)
     $N = 0$ 
     $I = 1$ 
end if

 $EF = EF + (0.1 - (4 - Q) \times (0.09 + (4 - Q) \times 0.03))$ 
if  $EF < 1.3$  then  $EF = 1.3$ 
else if  $EF > 2.5$  then  $EF = 2.5$ 
end if
return ( $N$ ,  $EF$ ,  $I$ )
```

When revising cards on a particular day, CardsQL will show you cards that are scheduled for that day or older(for overdue cards).

2.2 Literature Review

2.2.1 Study of existing system

Two popular flashcard apps are:

1. Quizlet

(a) Pros

- pre-made flashcards for subjects
- emphasis on mobile version UX which allows users to revise anytime, anywhere
- utilizes machine learning from anonymous user-data to create custom study plans for users

(b) Cons

- free version has ads & lacks advanced features
- can't be used offline on free version

2. Anki

(a) Pros

- Free & Open Source Software (FOSS)
- supports sync between multiple devices
- highly customizable with user-defined card types & community-made plugins

(b) Cons

- complex from start; CardsQL can act as gateway/ introduction to flashcards. Users can transition to Anki later
- might have to spend a lot of time customizing the program, adding plugins, to get a good experience

3 System Analysis and Design

3.1 System Analysis

3.1.1 Requirement Analysis

1. Functional requirements

Note: As *CardsQL* is meant for personal use, it only has one type of user instead of admin, multiple users etc.

(a) User

- can add cards
- can revise due cards
- can edit text & review date of existing cards

2. Use Case diagram

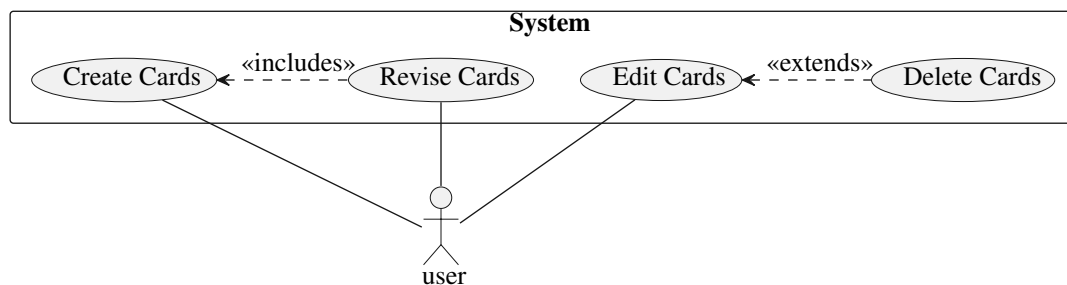


Figure 4: Use case diagram for CardsQL

3. Non-Functional requirements

- **offline access to all features**

Achieved by hosting php server & storing data both on user's computer

- **simple to use**

First thing user sees is just card creation interface

- **User shouldn't have to review too many cards in a day**

User can set daily card goal/limit

3.1.2 Feasibility Analysis

1. Technical

CardsQL is not too difficult to implement from a technical standpoint because it uses:

- plain HTML, CSS for the front-end
- basic JavaScript and PHP for the business logic
- SQLite, a lightweight RDBMS, for the database. It uses a single database file on the user's computer so it negates the need for maintaining a server for users to connect to.

2. Operational

- Because of the self-hosted architecture, the app will work offline without needing to connect to a central server. Thus, there is no need to designate manpower to ensure the app stays operational after launch.
- Users are sure to adopt the app as it is more convenient than paper flashcards, while still being easier to pick up and learn than the advanced programs discussed in **Study of existing system**

3. Economic

CardsQL is viable from an economic standpoint as:

- There are no additional costs for web hosting, server maintenance etc.
- There were no development costs as the app was built using existing hardware & freely-licensed tools.
- The app is distributed freely to help users & doesn't have any profit incentives.

3.1.3 Data Modeling

Note: The database is only used for storing cards along with their metadata. As it is meant to be used by a single-user, there is no need to implement or store login credentials.

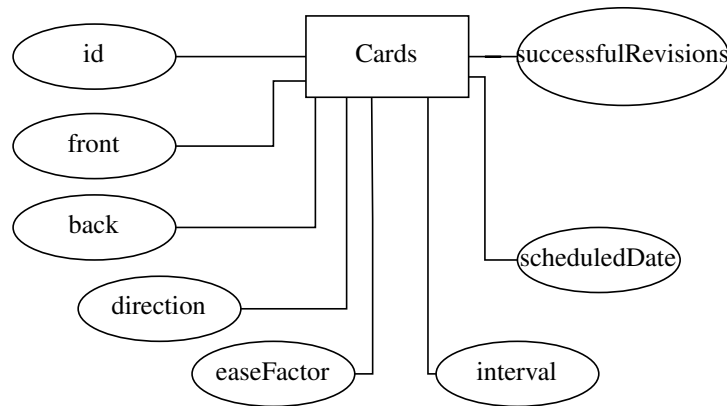


Figure 5: Entity-Relationship diagram for CardsQL

3.1.4 Proces Modeling(DFD)

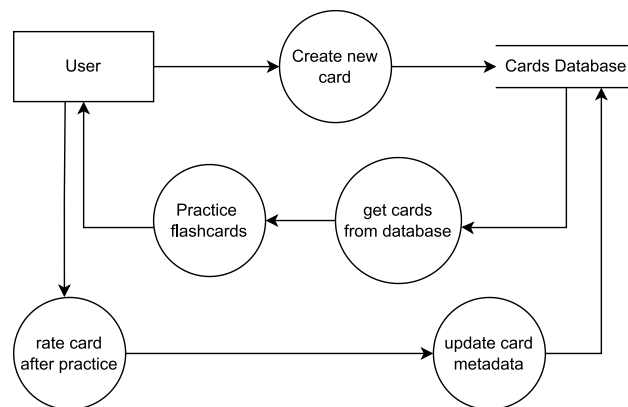


Figure 6: Data Flow Diagram diagram for CardsQL

3.2 System Design

3.2.1 Architectural Design

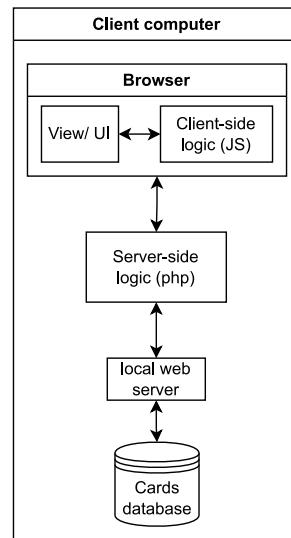


Figure 7: Architectural design for CardsQL

3.2.2 Database Schema Design

Note: In *SQLite*, *text* and *varchar* data type are the same and allocate space according to value's length. *real* data type refers to floating-point numbers

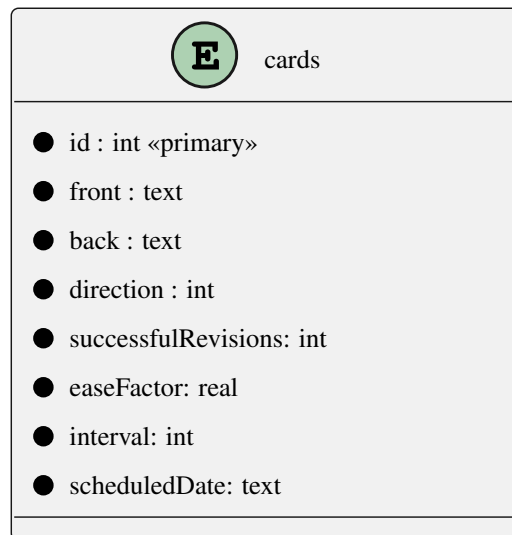


Figure 8: Database schema for CardsQL

3.2.3 Interface Design

Due to the app's simple nature, User Interface(UI) designs were not made beforehand. Styling was decided on during development, and applied after implementing correct business logic.

3.2.4 Physical DFD

The following Data Flow Diagram goes more in-depth than the previous diagram, and reflects how some processes are performed/implemented technically.

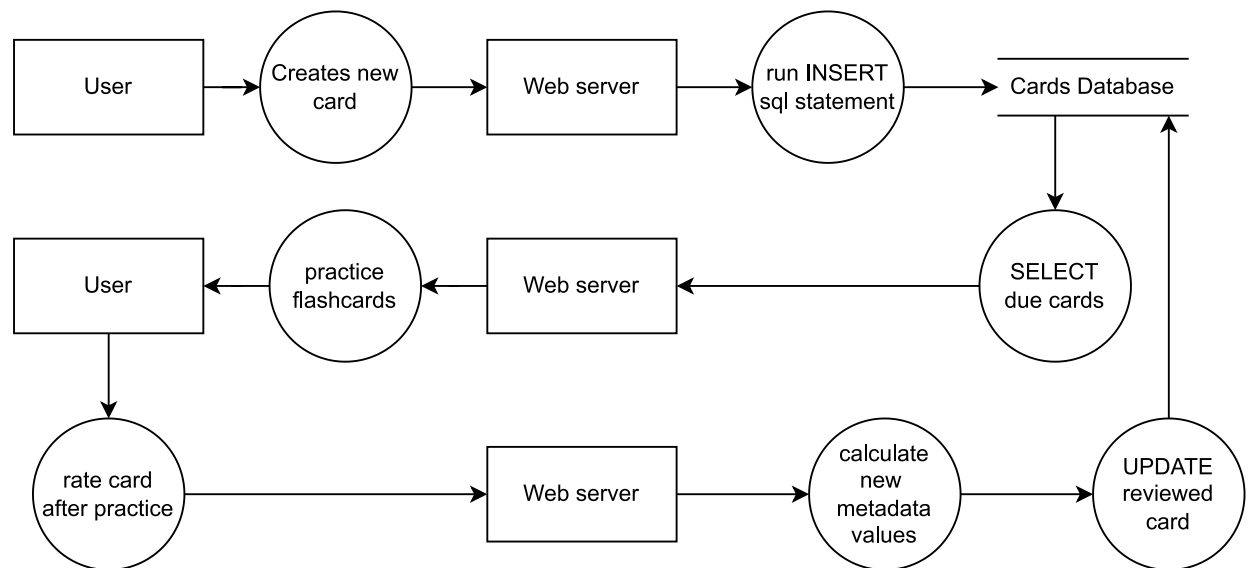


Figure 9: Physical DFD for CardsQL

4 Implementation and Testing

4.1 Implementation

4.1.1 Tools Used

The following Computer Aided Software Engineering(CASE) tools were used during the project's development:

1. Presentation layer

- HTML was used for structuring the webpage and its contents.
- Plain CSS, i.e. without any frameworks, was used for applying styling and decorations.

2. Logic layer

- JavaScript is used for client-side interactivity & DOM manipulation such as showing card answer only when button is clicked, populating edit dialog's values according to selected card etc.
- PHP is used for server-side scripting, i.e., mainly for performing Create, Read, Update, Delete (CRUD) operations on the database. Installing the language also allows us to use its basic, local web server, which is suitable for running CardsQL. This was used during development instead of dedicated server software like Apache.

3. Data layer:

- SQLite is a lightweight Relational Database Management System (RDBMS) used for our database purposes. It uses a single database file on the user's computer so it negates the need for maintaining an SQL server.

4. Miscellaneous:

- Git was used for version control, while Github served as a remote backup for the code.

4.1.2 Implementation Details of Modules

The different possible actions and how they are implemented are as follows:

1. **Add flashcards**

The homepage allows a user to add a new flashcard by specifying the text contents of its front and back side in an html form. Upon submitting the form, the contents are sent to the web server using an Hyper Text Transfer Protocol (HTTP) request. PHP code intercepts it and inserts a new row in the database table. Each row in the table represents a flashcard. Metadata values for newly created card such as the `Interval`, `EaseFactor` are set using default constraints defined in the table schema.

2. **Review flashcards**

In the review page, php code retrieves flashcards that have a due date of today or older, one at a time. Only the question(content of front side) is shown first and the user is prompted to recall the answer. They can press a button to show the correct answer and rate how well they remembered it on a scale of 0 to 3. The rating is sent to the web server and php code uses it to update the card's metadata using the SM-2 algorithm discussed previously. This process continues until no due cards are left.

3. Edit flashcards

The edit page initially shows a table of all the cards in the database. Clicking on a row will bring up a modal dialog for editing the selected card. The dialog has a form whose values are populated with the selected card using javascript. Confirming edits will submit the form and php code will execute UPDATE SQL statement on the database accordingly. The dialog also has a delete button which will tell php to execute a DELETE SQL statement instead.

4.2 Testing

4.2.1 Test cases for Unit Testing

4.2.2 Test cases for System Testing

5 Conclusion and Future Recommendation

5.1 Lesson learnt/ Outcome

5.2 Conclusion

5.3 Future recommendation

6 Apendix

7 References and Bibliography

- [1] A. Abdaal, “How to study: Active recall.” <https://aliabdaal.com/activerecallstudytechnique/>.
- [2] S. Parrish, “The spacing effect: How to improve learning and maximize retention.” <https://fs.blog/spacing-effect/>; Farnam Street Media Inc.
- [3] P. Woźniak, “Application of a computer to improve the results obtained in working with the supermemo method.” <https://www.super-memory.com/english/ol/sm2.htm>.