**Software Engineering Group Projects –**

**Design Specification Standards**

|  |  |
| --- | --- |
| Author: | Rishabh Kumar & Juraj Borka |
| Config Ref: | SE.QA.05 (for CS22120) |
| Date: | 25th January 2020 |
| Version: | 1.0 |
| Status: | Release |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2016

**CONTENTS**

[1 INTRODUCTION 4](#_Toc34326531)

[1.1 Purpose of this Document 4](#_Toc34326532)

[1.2 Scope 4](#_Toc34326533)

[1.3 Objectives 4](#_Toc34326534)

[2 DECOMPOSITION DESCRIPTION 4](#_Toc34326535)

[2.1 Programs in system 4](#_Toc34326536)

[2.2 Significant classes in each program 5](#_Toc34326537)

[2.3 Modules shared between programs – N/A 6](#_Toc34326538)

[2.3 Mapping from requirements to classes 6](#_Toc34326539)

[3 DEPENDENCY DESCRIPTION 6](#_Toc34326540)

[3.1 Component Diagrams 6](#_Toc34326541)

[3.2 Component Diagram for Program 1 6](#_Toc34326542)

[3.3 Component Diagram for Program 2 6](#_Toc34326543)

[4 DECOMPOSITION DESCRIPTION 8](#_Toc34326544)

[4.1 Programs in system 8](#_Toc34326545)

[4.2 Significant classes 9](#_Toc34326546)

[4.3 Table mapping requirements onto classes 9](#_Toc34326547)

[5 DEPENDENCY DESCRIPTION 10](#_Toc34326548)

[6 INTERFACE DESCRIPTION 10](#_Toc34326549)

[7 DETAILED DESIGN 11](#_Toc34326550)

# INTRODUCTION

## Purpose of this Document

The purpose of this document is to describe the implementation of the software Welsh Learning App. This document tracks the necessary information required to effectively define architecture and system design of the Welsh Learning App.

## Scope

This document describes the functions of the software “Welsh Learning App” and the implementation of classes, interfaces, controllers and relationship used in the back end. It also provides an explanation of the algorithms used in the implementation of the software’s logic and examples of the algorithms.

## Objectives

The main objective is to provide overall structure of the Welsh Learning App, in terms of classes, interfaces and controllers. Explaining how each feature of the application has been implemented, and how these features match the requirement specification.

# DECOMPOSITION DESCRIPTION

## Programs in system

There is only one program build using Java. The packages have been separated according to their use and role within the application.

Json package, as the name suggests, is responsible for holding all the text inputs and outputs. This package deals with loading data into the program and saving data into multiple files.

Mains package, the most crucial package, is responsible for running the user interface and enables functionality of the game.

Assets package is responsible for all the labels, buttons and text readers.

## Significant classes in each program

There is only one program present in the project, therefore all the classes are associated to the Welsh Learning App. There will be 5 classes present in the Welsh Learning App.

### 2.2.1 Main

|  |  |
| --- | --- |
| Class | Main |
| Purpose | The purpose of this class is to primarily connect the backend and frontend together. |

### 2.2.2 WordList

|  |  |
| --- | --- |
| Class | WordList |
| Purpose | WordList is responsible for initiating all the communication between the classes. |

### 2.2.3 Word

|  |  |
| --- | --- |
| Class | Word |
| Purpose | Word class is responsible for loading and saving data. |

### 2.2.4 Question

|  |  |
| --- | --- |
| Class | Question |
| Purpose | This class creates generates a multiple-choice question for the user to answer. |

### 2.2.5 MyWord

|  |  |
| --- | --- |
| Class | MyWord |
| Purpose |  |

## 2.3 Modules shared between programs – N/A

## Mapping from requirements to classes

|  |  |
| --- | --- |
| *Requirement* | *Classes providing requirement* |
| FR1 | WordList, Word |
| FR2 | WordList |
| FR3 |  |
| FR4 | WordList, PracticeList |
| FR5 | WordList, Word |
| FR6 | WordList |
| FR7 | WordList, Word |
| FR8 |  |
| FR9 |  |
| FR10 |  |

# DEPENDENCY DESCRIPTION

# 

## Component Diagrams

## Component Diagram for Program 1

## 

## Component Diagram for Program 2

A description of each element mentioned in the outline structure is presented in the following sections.

# INTERFACE DESCRIPTION

## Main

This class is the main class that is used to run the program.

Main() – this is a constructor that creates an empty Main object.

Public Void run() – this method runs the program and calls all the methods based on the users input.

Public Void displayWordList(bool favourite) – this method displays the words in one of the 2 Wordlists saved in the Main object depending on the input.

Public Void createWord(String English, String welsh, String type) – this method is used to create a new word and puts it in both WordLists in the object.

Public Void addToFavourite(Word word) – this method is used to add a word from the main WordList to the favourite WordList.

Public Void removeFromFavourite(Word word) – this method removes a word from the favourite WordList.

Public ArrayList search(String word, bool list) – this method displays all the words that fit a search term depending on the input.

Public Void practice(bool myWords) – this method runs the practice quiz and depending on the input, it either uses the favourite or the main WordList. It then displays the results and all the words the user got incorrect.

Public Word Question() – this method creates a multiple choice question and gets the user to answer it. If the user answers wrong, it returns the word.

Public Void flashcard(Word word) – this method allows the user to look at the word in a selected language and lets the user then check if it was the word they thought it was. (apparently this is in the specification).

Public Void options() - this method runs the options for the program.

## WordList

This class is used to store lists of words and also sort and search these lists.

WordList() – this is a constructor that creates an empty WordList object.

Public ArrayList search(String term) – this method searches the arraylist and outputs all the words that fit the search condition in alphabetical order.

Public Void addWord(Word newWord) – this method is used to add a word to the ArrayList in the WordList object.

Public Void removeWord(Word word) – this method is used to remove a word to the ArrayList in the WordList object.

Public Void save(String file) – this method is used to put words from the ArrayList in the object into a json file.

Public Void load(String file) – this method is used to load words from an json file into the ArrayList in the object.

## Word

This class is used for storing information for each word.

Word(String English, String welsh, String type) – this creates a word object.

Public String getEnglish() – this method gets the English translation of the word.

Public String getWelsh() – this method gets the Welsh translation of the word.

Public String getType() – this method gets the word type of the word.

## Question

This class is used to create and display multiple choice questions

Question(String wordToAnswer, String option1, String option2, String option3, String option4, String answer) – this is a constructor that creates a question with the input of a set of strings.

Public String getWordToAnswer() – this method returns the question

Public String getOption1() – this method returns option 1 of the multiple choice

Public String getOption2() – this method returns option 2 of the multiple choice

Public String getOption3() – this method returns option 3 of the multiple choice

Public String getOption4() – this method returns option 4 of the multiple choice

Public Void shuffle() – this method shuffles all the options so they are in random order

Public String getAnswer – this method returns the correct answer

## MyWords

This class is used to display all the words in the program onto the screen

## BinaryTree

This class is used to store the main node and methods that are then used on the existing tree.

Public Node getRoot() – this method returns the root node of the binary tree

Public

## Node

This class is used with the BinaryTree class to store words

Node(Word word) – this is the constructor for this class. This adds a word as a node must have a full word variable but the other variables can be empty

Public Node getLeft() – this method returns the node that is on the left

Public Node getRight() – this method returns the node that is on the right

Public Void setLeft(Node node) – this method sets the left node

Public Void setRight(Node node) – this method sets the right node

Public Word getWord() – this method returns the word stored in the node

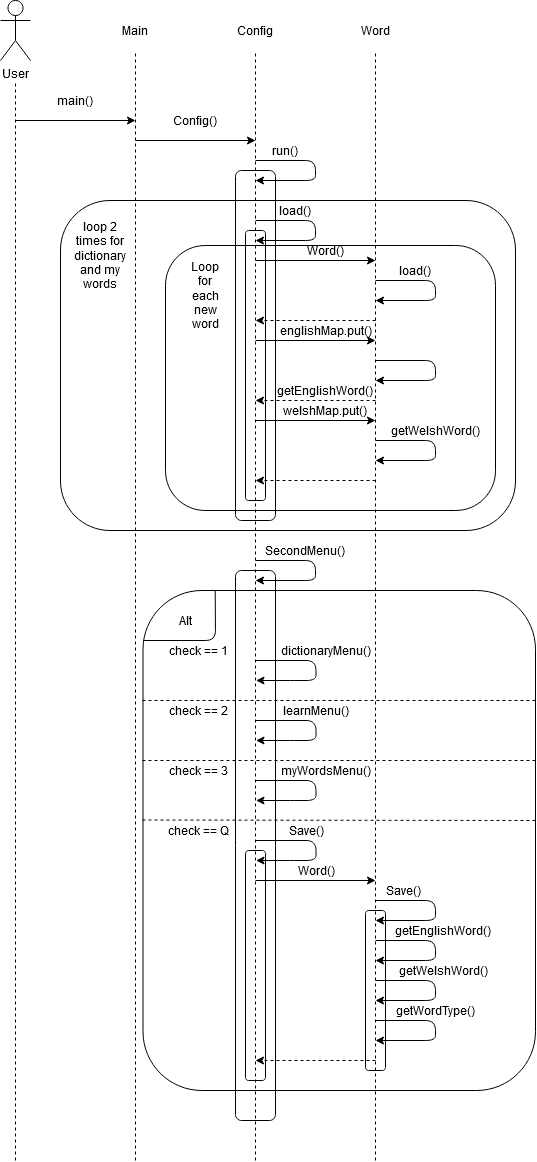
# DETAILED DESIGN

1. DETAILED DESIGN
   1. Sequence diagrams
   2. Significant algorithms
   3. Significant data structures

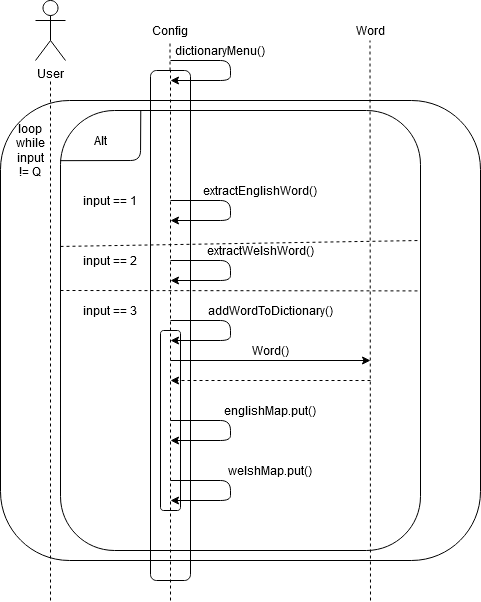
*Extract from document structure*

## Sequence diagram

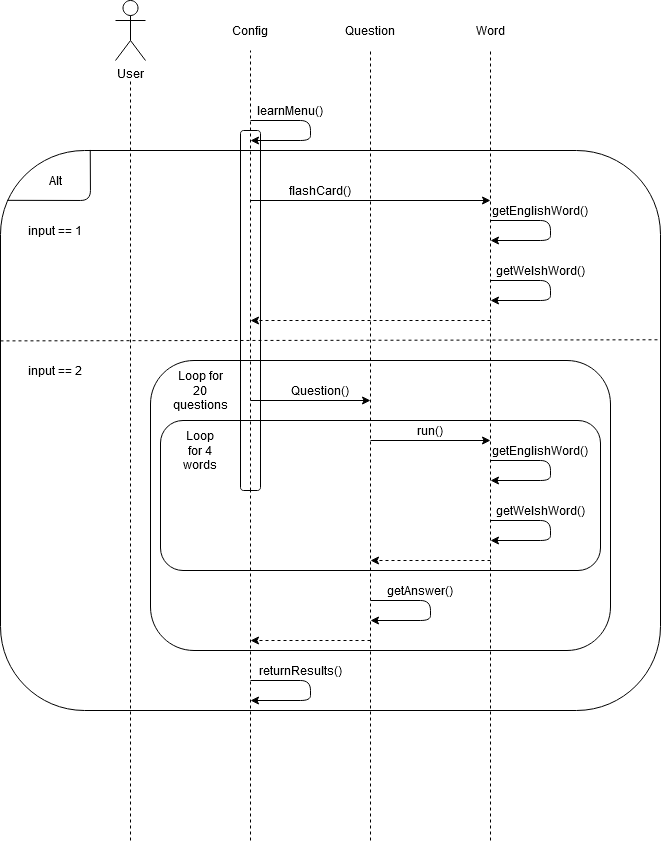
### Start, load, and save



### dictionary



### Flash Cards and Multiple Choice



It is inappropriate in the group project to provide all of the internal details of each module. However, you should have considered the difficult parts of the design, and the way in which all of the classes work together. For any item whose internal workings are not self-evident, enough details should be provided to demonstrate the feasibility and coherence of the overall design.

UML *Sequence diagrams* are a good way of documenting how the classes work together for the major operations of the program (identified by use cases, described in the user manual for the group project).

During design, the team will have performed experimental programming and theoretical investigation to reduce the risk associated with difficult parts of the implementation. Decisions on difficult parts of the system should be documented in this section. The most appropriate way of doing this for algorithms is likely to be as a textual description of what is to be done. Where code exists, it might be referred to, but probably should not be included.

Significant data structures occur when a number of objects are linked together in a more complex structure. Class diagrams showing the entity relationships between classes should be drawn where appropriate, along with object diagrams which show how the static relationships in class diagrams work out in real examples. The mechanism to support any persistent data should be described. If data is transmitted from one program in the system to another, or is exported from the system, it would be appropriate to specify the format. This might involve re-use of an existing format, the design of a schema for JSON or XML, or design of of a bespoke format.

**REFERENCES**

1. QA Document SE.QA.01 - Quality Assurance Plan.

1. QA Document SE.QA.03 - Project Management Standards.

1. QA Document SE.QA.08 - Operating Procedures and Configuration Management Standards.

1. QA Document SE.QA.02 - General Documentation Standards.

1. QA Document SE.QA.09 - Java Coding Standards.

**DOCUMENT HISTORY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| 1.0 | N/A | 20/02/20 | Document Created | Rik7 |
| 1.1 | N/A | 06/03/20 | Updated Interface Description | Jub27 |
| 1.2 | N/A | 17/03/20 | Added Sequence Diagrams | Jub27 |