MUSINS-PRO

SOFTWARE REQUIREMENTS SPECIFICATION

CENG490

Yağmur ERTAŞ - 1819333
Duygu ABADAN - 1818863
Baler İLHAN - 1819853
Anıl ARPACI - 1818954
11/30/2014

Table of Contents

1. Introduction

This is a software requirement specification document for musical instrument playing system. Firstly, the purpose and scope of this document will be explained. Secondly, the overall description of the system will be given. Then specific requirements will be stated, data models and behavioral models together with their description in this document. Lastly, development plan will be introduced and the document will be concluded.

1.1 Problem Definition

Nowadays, many people are involved in creation part of the music. But most of these people do not have any musical education for playing instruments; they are just trying to do it. So, there is no any self-playing musical system, which takes a photograph of any musical notes to play, for this kind of people as a commercial product.

1.2 Purpose

The purpose of this document is to present a detailed description of musical instrument playing system. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is aimed for both the customer for its approval and a reference for developing the system for the development team.

1.3 Scope

The developed product is a system that can play musical instrument with just musical notes. System will have options that loading image file or capturing image. Any proper image file can be loaded to system for playing. With this property, the playing musical notes in a musical instrument will be done with a very simple way. Objective of the system is to make playing musical instrument easier for anybody who can interested in music.

1.4 Definitions, acronyms, and abbreviations

All the definitions, acronyms and abbreviations which are used in this document are described in the following table.

Block diagram	A diagram showing in schematic form the general arrangement of the parts or components of a complex system or process.
Class Diagram	A type of static structure diagram in UML that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes
DPI	Dots Per Inch
IDE	Integrated Development Environment
IEEE Standards	International Electric Electronic Engineering Standards 830
JRE	Java Runtime Environment
MP	Megapixel
MusicXML	An XML based file format for representing musical notation.
Octave	An interval between one musical pitch and another with half or double its frequency.
OMR	Optical Mark Recognition
PC	Personal Computer
Rest	An interval of silence in a piece of music, marked by a symbol indicating the length of the pause.
Score	A written form of a musical composition.
Score Sheet	A handwritten or printed for of music notation that uses musical symbols.
SDK	Software Development Kit
SRS	Software Requirements Specifications

StarUML	Design tool of diagrams
State Transition Diagram	A type of static structure diagram in UML that describes the transition of the system functions
USB	Universal Serial Bus
Use Case Diagram	A type of static structure diagram in UML that describes user's interaction with the system
User	Person who wants to use the system
User Interface	An interface that our system contact with the user of the system. It gets all needed information for its running, from user to our system.
XML	Extensible Markup Language
.bmp	Bitmap image file format
.mip	Developer designed file format (MusIns-Pro)
.tif, .tiff	Tagged Image File Format

Table 1: Definitions and Acronyms

1.5 References

- [1] IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998.
- [2] StarUML 5.0 User Guide. (2005). Retrieved from http://staruml.sourceforge.net/docs/user-guide(en)/toc.html

1.6 Overview

The next chapter, the Overall Description section, of this document gives an overview of the functionality of the system. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next chapter. Furthermore, the chapter also puts into words the system constraints and assumptions about the system.

The third chapter, Requirements Specification section, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product and the description of the different system interfaces.

Both second and third sections of the document describe the same system in its entirety, but are intended for different audiences and thus use different language.

The fourth chapter has diagrams, explanations and statements about classes which contain data and their relationships in this system.

In the fifth chapter of this document, how the transitions states of this system are set up and task of these states will be explained.

In the last chapters, how is our team and project planning, and conclusion are written to inform readers of this document.

2. Overall description

In this part, the general factors that affect the software and hardware part of the product and its requirements will be explained shortly. The detailed information will be given in section 3.

2.1 Product perspective

Music box is the product which reads notes and plays musical instrument. The product comprises of software part and hardware equipment. This product is developed by independent software. Hardware equipment of the product is dependent to the musical instrument organ. Software part provides reading notes from the file which is created by capturing or uploading photo, uploading musicXML file or loading .mip file by the user. ".mip" is own extension of the product. Passing from the software part to hardware part realizes reading data from the created notes file and sending data to hardware. The hardware part provides playing instrument by servo motors and solenoids which are programmed by Arduino Mega 2560.

The product with an interface and the hardware part has only one kind actor shown as the Figure 1.

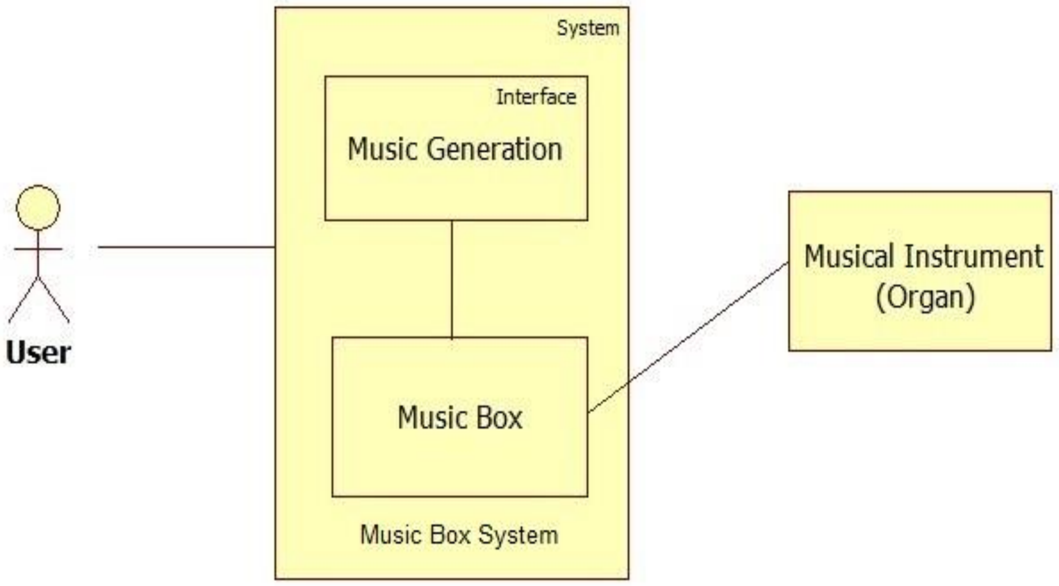


Figure 1: Block diagram of the product

2.1.1 System interfaces

The Music Box is dependent product; it needs to be integrated into musical instrument. By using the product, in the future when user gives a score to the system, they can listen to music from musical instrument even if they do not play the musical instrument. Java is used for code and design of the system and Arduino IDE is used for programming the motors which press the ivories.

2.1.2 User interfaces

The user interface of Music Box is very simple and error-susceptible. One screen will be visible to the user during the software execution. This screen contains title of the product, six buttons for the user functionalities and the field to drag and drop the file. Because of the product will be supported by the Windows, user interface will be shown in the computers with Windows operating system. Customization feature will not be available and interface elements will be locked. Interface will be resizable and it is compatible with all kind of computers with JRE.

All the users who buy the product can use the system. The user will select any functionality via the user interface. These functionalities are loading music file, capturing an image which includes the musical notes, starting to play musical instrument and stopping it.

When the user uploads the file, if the files are not proper for the system or if the system loses the connection, error messages appear.

2.1.3 Hardware interfaces

The project has hardware components. To run this component, the system needs some interfaces. Firstly, the user has to a computer Moreover, this computer can take a photo or the user must have a tool which can take a photo and the user can upload this picture to computer. Furthermore, the organ is necessary. It has at least two octaves. Hardware component of the project are portable and it can suitable for every organ.

2.1.4 Software interfaces

The software system will be designed with Java programming language. Thus java runtime environment (JRE) will be necessary to use this product. Also this system can be used with Windows 7 and higher operating system. Except that it can be used by any user who has a personal computer with Windows.

Software interface will have only one screen which achieves all the functionalities. User will be able to manage the product via this only interface, user interface. For helper browser windows of MusicBox system, Java libraries will be used.

2.1.5 Communication interfaces

The MusicBox system will communicate with computer and electronic device via USB protocol for connection, communication and power supply.

2.1.6 Memory

Memory constraint is unnecessary for this product.

2.1.7 Operations

The operations are explained in User Interfaces section (2.1.2) in detailed. Hence, it will not be mentioned again here.

2.1.8 Site adaptation requirements

There is no need for any adaptation for using this product. For using the product, it is sufficient a computer with Windows operation system.

2.2 Product functions

MusicBox system provides many menu functionalities to user for managing product. The functionalities are explained in later parts of this document in detailed version. This section provides brief summary of functionalities on the system. In the given use case diagram below provides a better understanding of the general system.

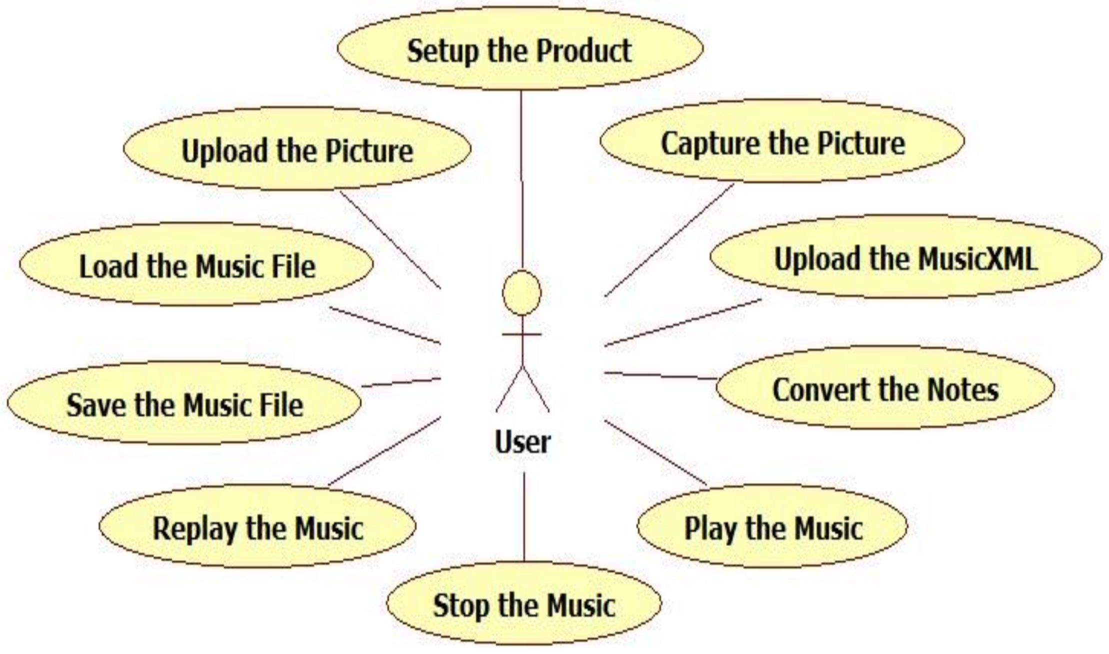


Figure 2: Use case diagram of the product

2.2.1 Setup the Product Functionality

User should use this functionality after integrating the product into organ via computer and plug in.

2.2.2 Capture the Picture Functionality

Users can capture the score sheet picture by using this functionality.

2.2.3 Upload the MusicXML Functionality

Users can upload musicXML directly to the system via this functionality.

2.2.4 Convert the Notes Functionality

This functionality provides to conversion between the picture and developer defined music file. After this step system will be ready to play the song.

2.2.5 Play the Music Functionality

This functionality makes the system play the song.

2.2.6 Stop the Music Functionality

User can stop the song during process by using this functionality.

2.2.7 Replay the Music Functionality

User can replay the song in the loaded file by using only hardware by this functionality.

2.2.8 Save the Music File Functionality

Developer created file can be saved for later usage via this functionality.

2.2.9 Load the Music File Functionality

System can be used by loading application created files (.mip files) via this functionality.

2.2.10 Upload the Picture Functionality

Users can use product by uploading score sheets via this functionality.

2.3 User Characteristics

The product generally will be used by people who interests with music. This user type can be wants to play a musical instrument. Moreover, they can be wants to learn a musical instrument. Furthermore, this user type can be wants to compose a melody using this product.

Besides, the interface of the product is user friendly. Hence, this user type is expected to have basic level of computer knowledge and experience. In addition, the user type has basic level of music knowledge.

2.4 Constraints

2.4.1 Hardware Requirement

- The computer shall have USB connection to using the product.
- The computer shall have necessary picture requirement to upload the picture.
- If the computer shall not have necessary picture requirement, the user shall have a components which can be taken a photo.
- This product shall work on with an organ which has at least two octaves.
- The product shall be fed with a power supply.

2.4.2 Software Requirements

- This product shall work on Windows 7 operating system or higher.
- JRE 7 or higher shall be necessary to run the system.
- Minimum photograph quality shall be 5 MP or 300 DPI intensity.
- The computer shall have the MusicBox system installed.

2.4.3 Other Requirements

- Safety precautions should have taken while using device.
- Safety precautions should have taken when using voltage source.

2.5 Assumptions and Dependencies

These are not strict requirements for the future releases but rather the additional functionalities that may or may not be integrated in later versions. The sole purpose is to present the points considered on extensibility and to emphasize the potential for additional features.

- Different language options for the interface.
- Slow down and speed up option can be added to the system.
- Different music formats can be used in the products.
- Different picture formats can be used in the products.
- This product can be fit to the other musical instruments.
- Different operating systems can be supported.
- Mobility can be taken into account.

3. Specific requirements

This section will describe software requirements in detail as subsections which are interface requirements, functional requirements and non-functional requirements.

3.1 Interface Requirements

3.1.1 User Interface

The user who wants to play music with MusicBox system can simply initiate the program. After initiation of system, the main interface of MusicBox welcomes the user. The main interface is the only user interface of the whole system and it includes all functionalities of the system. This user interface can be seen in Figure 3.

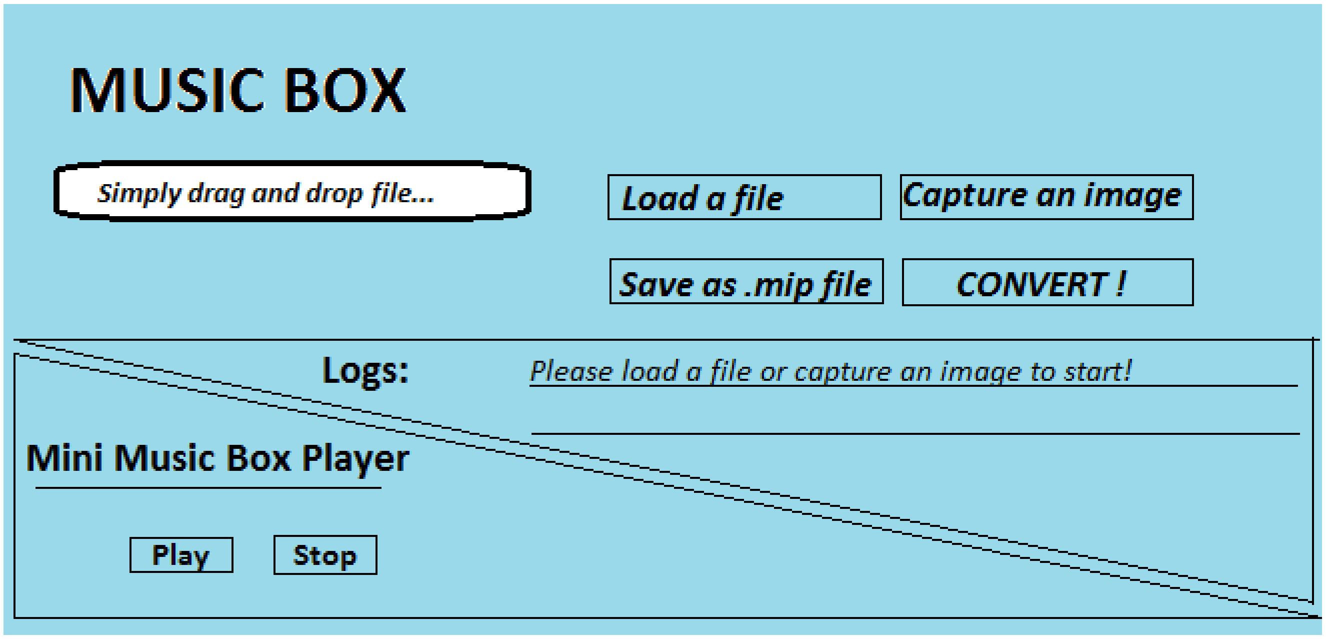


Figure 3: User Interface of Music Box

There are three way for initiating MusicBox system. Firstly, there is a ‘drag and drop file’ part at the left top of the page for initiating the system with proper files, such as .tiff, .bmp, .mip and MusicXML file; therefore user can drag these files to this ‘drag and drop file’ box. Secondly, user can initiate the system with loading a file by clicking ‘Load a file’ and browsing a file to load through browser library of Java. Lastly, there is ‘capture an image’ button at the right top of the page. By clicking that button, user can capture an image through camera connected to PC.

After initiating MusicBox system, user can save .mip MusicBox file by clicking 'Save as .mip file' button. User can browse and select saving place on PC through browser library of Java. This .mip file has all information about the image's musical notes and it is ready to play with instrument.

After initiating MusicBox system, user also directly can play instrument with clicking 'CONVERT !' button. This button converts loaded image file to .mip file and sends it to Arduino board for playing it on musical instrument. When this conversion is done, loaded notes are ready for playing. Also 'Play' and 'Stop' buttons, which are placed on the bottom of the interface window, become active after that conversion.

There are two parts on the bottom of the interface. 'Logs:' part shows all returns of system to the user for his/her information; such as error information about loading file, process information about conversion, etc. The 'Logs' part answers all interaction of user for make system easier to use. 'Mini Music Box Player' part of the interface includes 2 buttons on it and they will be active if and only if conversion is done successfully. 'Play' button starts the playing of the notes on instrument. User can stop the playing music by simply clicking 'Stop' button. If a user stops the playing music, he/she can continue playing by clicking 'Play' button again.

3.1.2 Hardware Interface

Since the system's application runs on PC and there is an Arduino board connected to PC, the only hardware interface is the Arduino's board. There is only 'Reset' button on it and by pushing it user can replay last loaded music.

3.1.3 Software Interface

The only software interface of our product is the main interface of the product which is the user interface in terms of diagrams. Users can use all aspects of product with this interface.

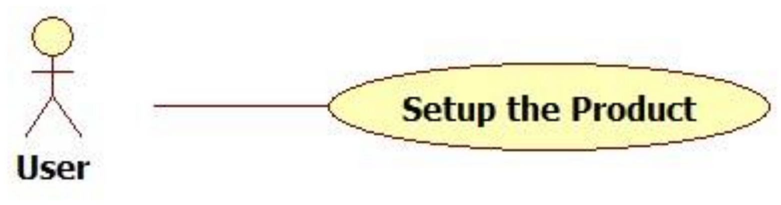
3.1.4 Communication Interface

The communication between the system and Arduino is done through USB connection. This communication is handled by the underlying operating system through our Java code. So there is no direct interface about communication.

3.2 Functional Requirements

3.2.1 Use Case 01: Setup the Product

Diagram:



Brief Description

The users who want to use the product, should integrate the product into organ and setup the system via computer and plug-in.

Initial Step-By-Step Description

1. The user should integrate the product into organ.

2. After integration, the user should make a power supply connection and USB connection to computer.

3. The user should install the program while getting started.

4. After installation, the product is ready to use with the user interface which appears on the screen.

3.2.2 Use Case 02: Upload the Picture

Diagram:

User
Upload the Picture

Brief Description

All the users, who want to play the music via the product, can upload the picture which includes musical notes.

Initial Step-By-Step Description

1. The user should browse and select the picture file which has ".bmp" or ".tiff" or ".mip" file extension by clicking the "Load a file" button or drag the file to the screen directly.

2. After selection the picture file, the user should click load button and upload it.

3.2.3 Use Case 03: Capture the Picture

Diagram:

User
Capture the Picture

Brief Description

All the users, who want to play the music via the product, can capture the picture which includes musical notes.

Initial Step-By-Step Description

1. The user should capture the picture of musical notes by clicking the "Capture an image" button to convert.

2. If the user wants to upload any captured musical notes and if the files are not in the ".bmp" or ".tiff" file format, s/he should convert the files to one of these formats and upload to the system as mentioned in the use case 02.

3.2.4 Use Case 04: Upload the MusicXML

Diagram:

User
Upload the MusicXML

Brief Description

Other choice for the users, who want to play the song, is uploading the MusicXML file to the system.

Initial Step-By-Step Description

1. In order to use this function, the user browses and selects the MusicXML file by clicking the "Load a file" button placed on the user interface or drag the file to the screen directly.

2. After selection the picture file, the user should click load button and upload it.

3.2.5 Use Case 05: Load the Music File

Diagram:

User
Load the Music File

Brief Description

All the users, who want to play the music via the product, can load the music file which was created by this application previously.

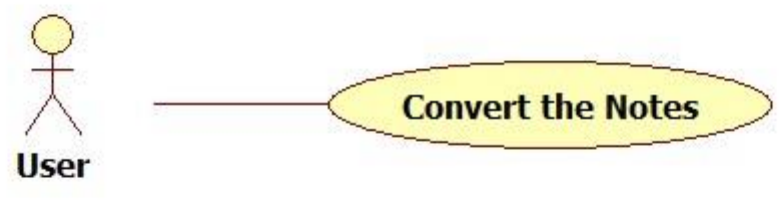
Initial Step-By-Step Description

1. In order to use this function, the user browses and selects the file with ".mip" extension by clicking the "Load a file" button placed on the user interface or drag the file to the screen directly.

2. After selection the file, the user should click load button and upload it.

3.2.6 Use Case 06: Convert the Notes

Diagram:



Brief Description

After the user uploads any file, s/he should convert the notes to make ready to play.

Initial Step-By-Step Description

1. In order to use this function, the user should upload any music file.

2. After uploading file, the user should press the "CONVERT!" button placed on the user interface to make ready to play.

3.2.7 Use Case 07: Save the Music File

Diagram:

User
Save the Music File

Brief Description

After converting the music file, the user can save the file which is created by the application and store it to use again.

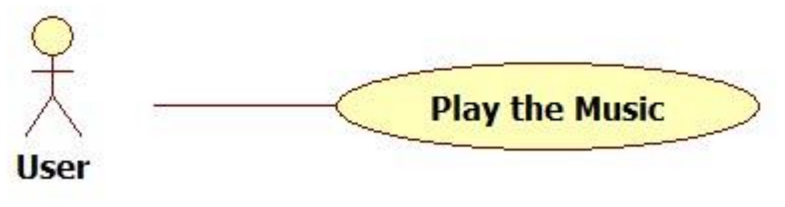
Initial Step-By-Step Description

1. In order to use this function, the user should convert uploaded music file.

2. If the user will want to play the song again in the future, s/he should click "Save as .mip file" button and save the converted file by the application with ".mip" file extension.

3.2.8 Use Case 08: Play the Music

Diagram:



Brief Description

After converting the music file, the user presses the button to run the system and make organ play the song.

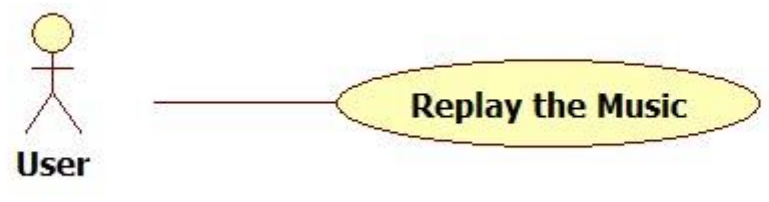
Initial Step-By-Step Description

1. In order to use this function, the user should convert uploaded music file.

2. After converting the file, the user should press the “Play” button placed on the user interface to play the song by the product.

3.2.9 Use Case 09: Replay the Music

Diagram:



Brief Description

The user should be able to replay the music by resetting the hardware part of the product.

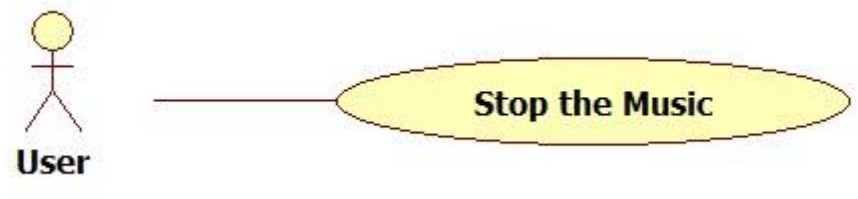
Initial Step-By-Step Description

1. In order to use this function, the user should play any music formerly.

2. When the user disconnects USB connection or plugs out the product, the last played music information stays in the product. Therefore, the user can replay the last played music by the “Reset” button of the product, even if s/he does not make a computer connection.

3.2.10 Use Case 10: Stop the Music

Diagram:



Brief Description

The user should be able to stop the music by the button placed on the user interface.

Initial Step-By-Step Description

1. In order to use this function, the user should play the music.

2. If the user wants to stop the music, s/he should press the “Stop” button.

3.3 Non-functional Requirements

3.3.1 Performance requirements

Communication between the system and the product is very important issue in this project since it is necessary to load music information in the device. If the product connects with better USB connection, the speed of product will be faster. Furthermore, loading the picture time is at most 10 seconds. On Windows operating system, the product will be started automatically and it will be directly ready to run. In addition, parsing part and reading notes for 1 page of score sheet will work very fast and the user will never wait more than 5 seconds for this stages.

3.3.2 Design constraints

- Arduino boards will be used for controlling stepper motors and solenoids.
- For the transition between image file to .mip file, MusicXML file will be used and supported for loading file.
- For board programming, Arduino Mega will be used.
- For parse the file, connection between Arduino and design of the user interface will be implemented with Java.
- For the diagrams in the project, UML standards will be used.
- For the reporting, IEEE standards will be used.

3.3.3 Software System Attributes

In this sub-section, the system attributes which makes the system get closer the perfect system will be discussed.

3.3.3.1 Reliability

The most important attribute of the system is reliability as every other product. To provide user a quality and reliable product, the system will be tested and possible errors will be minimized. The system shall provide explanatory messages when an unexpected event occurs.

3.3.3.2 Portability

The project will be developed by using common technologies and tools. Furthermore, the system which will be developed shall be portable. In other words, our product will be portable and this product can be used any model of the organ. This product is detachable for every organ which has at least two octaves. Also it can connect any kind of personal computer that has USB port and JRE.

3.3.3.2 Security

The system which will be developed can reach every people who buy our products. There is not any security system or restriction on this product.

3.3.3.3 Availability

Users shall be able to use the product every time if they have necessary hardware components and electricity. Software part of the system is also available at every time. If the system crashes, there will be no data loss but user must be start over with loading an image to the system.

3.3.3.4 Maintainability

In order to establish maintainability, all documentations about the software should be very detailed and understandable, and they should be prepared in IEEE standard 830-1998. Furthermore, it should be avoided from the complexity.

3.3.3.5 Safety

Since the project is an electrical system, it will be controlled the system electric leakage. The system shall have no electric which can thread people's lives.

3.3.3.6 Integrability

The design of the system shall be enterable as it can be maintained easily. The system will be designed in a way that allows addition of different musical instruments because of the portable and removable. Current hardware constraints are allowed these additions. As a result of this situation, this product can be used by many people among the world.

4. Data Model and Description

This part of the SRS is about classes which contains data and their relationships.

4.1 Data Description

In this section, data objects that will be managed and manipulated by the software are described.

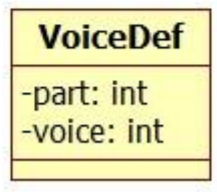
4.1.1 Data Objects

This sub-part explains the classes which contain data variables and functions which updates data variables.

4.1.1.1 VoiceDef Class

This class will be defined in the software part of the project. It will be used during parsing musicXML file from other classes like “MusicXMLParser”. Main purpose of this class is to form musicString structure which will be a combination of part and voice.

Diagram:



Description:

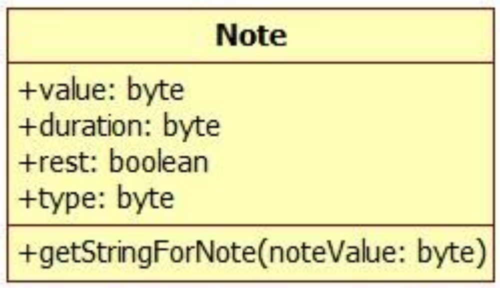
Name	Type/Return Value Type	Visibility	Definition
part	int	private	This variable refers part value of the musicString.

voice	int	private	This variable refers to voice value of the musicString.

4.1.1.2 Note Class

This class will be defined in the software part of the project. The purpose of Note class is keeping information about each note of the song which is contained by musicXML file.

Diagram:



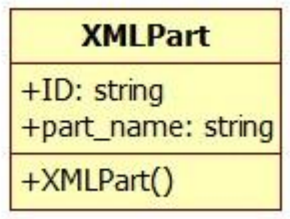
Description:

Name	**Type / Return Value Type**	**Visibility**	**Definition**
value	byte	public	This value refers the numeric value of the note.
duration	byte	public	This value refers to duration the duration of the note, as milliseconds.
rest	boolean	public	This value indicates whether this note is rest.
type	byte	public	This value indicates the type of the note.
getStringForNote	String	public static	This function returns a MusicString representation of the note value and duration which indicates a note and an octave.

4.1.1.3 XMLPart Class

This class is used as helper class for the MusicXMLParser class. It will be defined in the software part of the project. A musical notes can store multiple instruments however this project only interest with the organ. This class contains part information of the instrument.

Diagram:



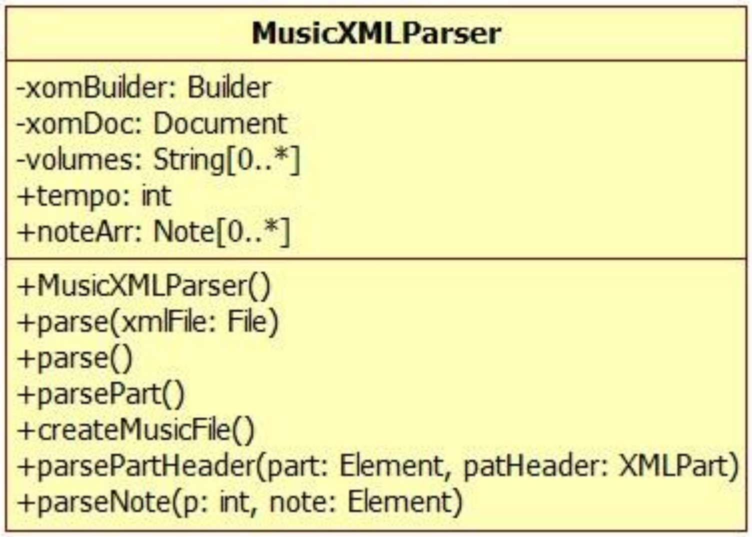
Description:

Name	Type / Return Value Type	Visibility	Definition
ID	string	public	This value refers the id of the part.
part_name	string	public	This value refers the name of the part.
XMLPart	void	public	This function is the constructor of the class which sets empty values as default.

4.1.1.4 MusicXMLParser Class

This class will be defined in the software part of the project. It can be seen as main class of the software package which uses all other classes and performs operations like parsing musicXML file and storing all notes' information in the designed class.

Diagram:



Description:

Name	**Type / Return Value Type**	**Visibility**	**Definition**
xomBuilder	Builder	private	This value responsible for creating XOM Document objects file by reading an XML document
xomDoc	Document	private	This value represents a complete XML document including its root element and others
volumes	String[0..*]	private	This value is volume of the song.
tempo	int	public	This value refers to tempo of the current song. It is measured in "pulses per quarter". The parser uses this value to convert note durations, which are relative values and not directly

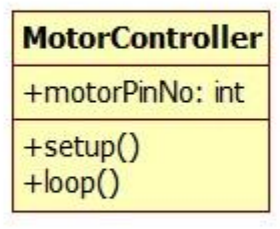
			related to time measurements, into actual times.
noteArr	Note[0..*]	public	This array stores whole notes of the current song for later usage. Note is a structure which is defined by developer.
MusicXMLParser()	void	public	This function is the constructor of the given class. It will initialize the default values.
parse()	void	public	This function will be responsible for starting parsing musicXML file. It will be called via filename.
parse()	void	public	This function will be responsible for starting parsing musicXML file. It overrides parse function with no parameters.
parsePart()	void	public	This function will be responsible for parsing one part. It can be called via three parameters which are, entire part of the music and an array of XMLpart classes that contains instrument information of the part
parsePartHeader()	void	public	This function is responsible for parsing an element in the part- list. It can be called with an element and an array of XMLpart classes which contains partlist elements
parseNote()	void	public	This function is responsible for parsing MusicXML note Element. It can be called with the note Element . Finally it creates Note object and after storing all information in it, pushes it to Note array for later step.

createMusicFile()	File	public	This function is responsible for creating a new file for Arduino.

4.1.1.5 MotorController Class

This class will be defined in the hardware part of the project. It contains a variable which name is motorPinNo. Moreover, this class provides to control servo motors and solenoids.

Diagram:



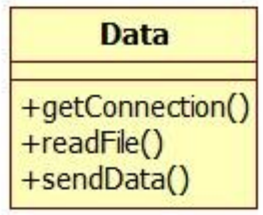
Description:

Name	**Type / Return Value Type**	**Visibility**	**Definition**
motorPinNo	int	public	This value detects which motor attaches which Arduino digital pin. Moreover, this value is essential to match the related note and the motor.
setup()	void	public	This function is responsible to set motors and solenoids. The motors and solenoids that attach which digital input can be seen in this function.
loop()	void	public	This function is responsible to run correct motors and solenoids. Furthermore, the related motors and solenoids can be stopped in this function.

4.1.1.6 Data Class

This class will be defined in the hardware part of the project. This class serves as a bridge between software part and hardware part. It provides a connection between the system and user.

Diagram:



Description:

Name	**Type / Return Value Type**	**Visibility**	**Definition**
getConnection()	void	public	This function is responsible for the connection task between software part and hardware part.
readFile()	void	public	This function is responsible for the reading data from the file.
sendData()	void	public	This function is responsible for the data sending to microcontroller which is Arduino from the file.

4.1.2 Data dictionary

This project will be formed from two parts which are software and hardware. Therefore there will be three packages to implement and connect them. As seen in Figure 4 below, the relationship between packages is;

- Hardware and software part of the project generalizes UserInterface package.
- UserInterface package will be use by the user. It will provide connection between other packages and the application.
- Developers will be implement hardware and software part separately.

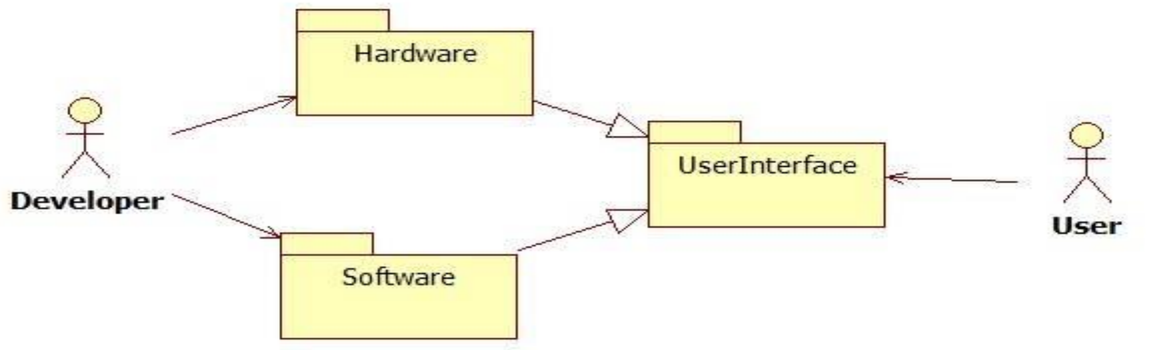


Figure 4: UserInterface Package Diagram

The relationship between classes which are inside the software package is:

- XMLPart class extends java Object class and it is used by MusicXMLParser. There is a strong relationship between them since XMLPart class will be implemented as helper class for MusicXMLParser.
- MusicXMLParser class has elements from VoiceDef and Note classes. It uses these classes during parsing. Relationship is weak since other classes have information about the score sheet and note.

It can be seen in figure 5 below.

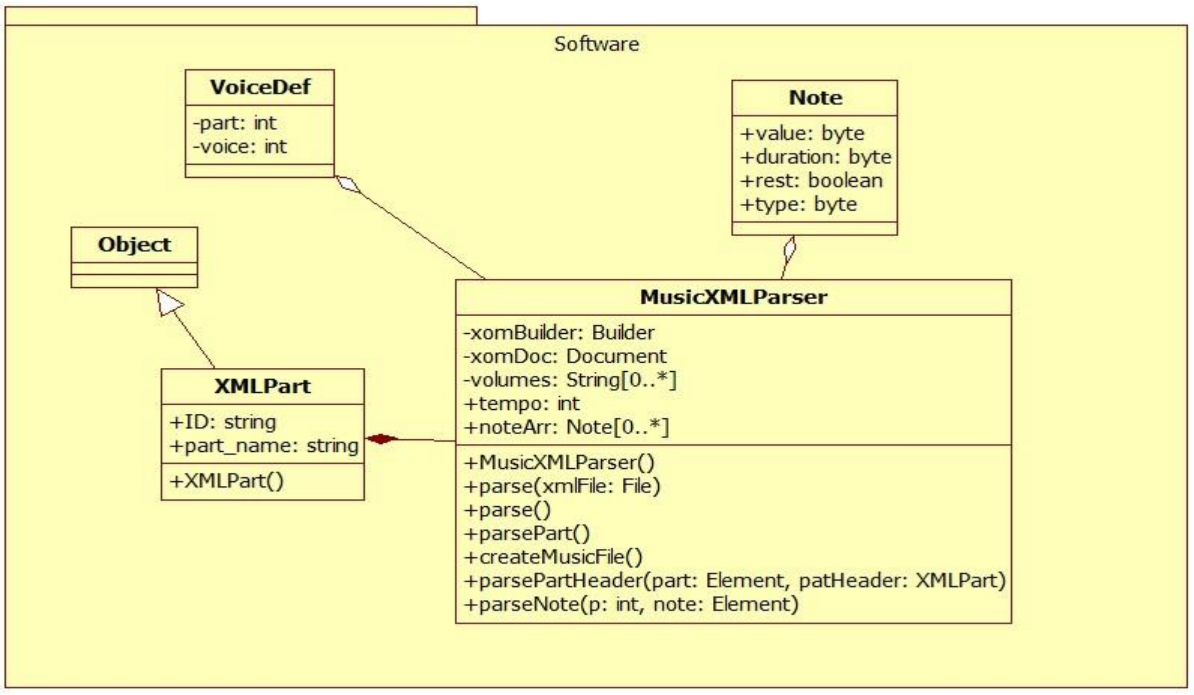


Figure 5: Software Package Diagram

The relationship between classes which are inside hardware package is:

- MotorController class has elements from Data class. This class controls motors via Arduino.
- Data class provides connection between file and the Arduino.

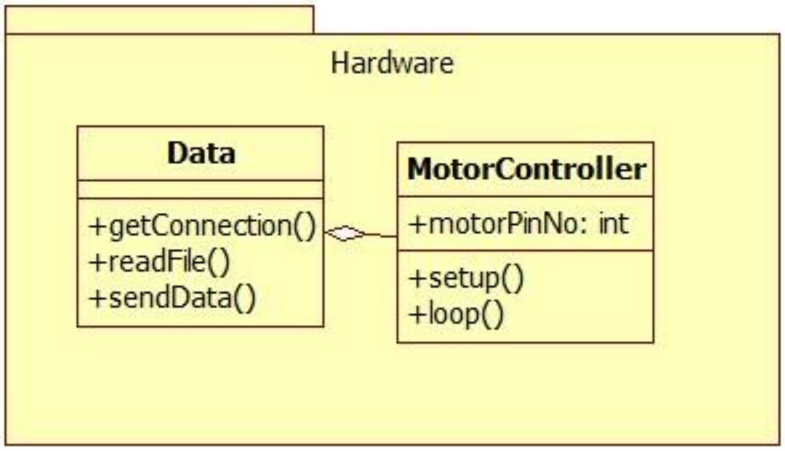


Figure 6: Hardware Package Diagram

5. Behavioral Model and Description

In this part of this document, how the transitions states are set up and task of these states will be explained.

5.1 Description for Software Behavior

To use the system, our product whose name is MusicBox, a computer which can be upload notes and a musical instruments which is specified as organ. There are 7 states of the system.

Initial State: This state is reached by providing the product.

Connection State: This state is reached from initial state when the user connects the product and the organ. Furthermore, the user must connect the product and computer and the product had to be connected with power unit. From this state, system can reach the Picture State, musicXML State and Note State. Moreover, from this state, system can reach the final state. In addition, if the user stops the music, system returns the Connection State.

Picture State: This state is reached from connection state when the user uploads or captures the picture. The picture must have some requirements. These requirements are specified in the assumptions and dependencies. If the picture satisfies these conditions, the system can reach musicXML State automatically.

MusicXML State: This state is reached from picture state when the user correct picture for the program. Moreover, this state is reached from the connection state when the user uploads musicXML directly. From this state, system can reach the Note State.

Note State: This state is reached from musicXML state when the user reads notes from musicXML. Furthermore, if the user uploads .mip file, the user can directly reach Note State from Connection State. From this state, system can reach ".mip file" State.

.mip file State: This state is reached from note state when the user plays the selected music. This music file's extension is ".mip". From this state, system can return Connection State.

Final State: This state is reached from connection state when the user disconnects the product. If the user stops the music or connects the product, system can be disconnected.

5.2 State Transition Diagrams

State transition diagram which states that which operations and conditions changes the state of the system can be shown in below Figure 7.

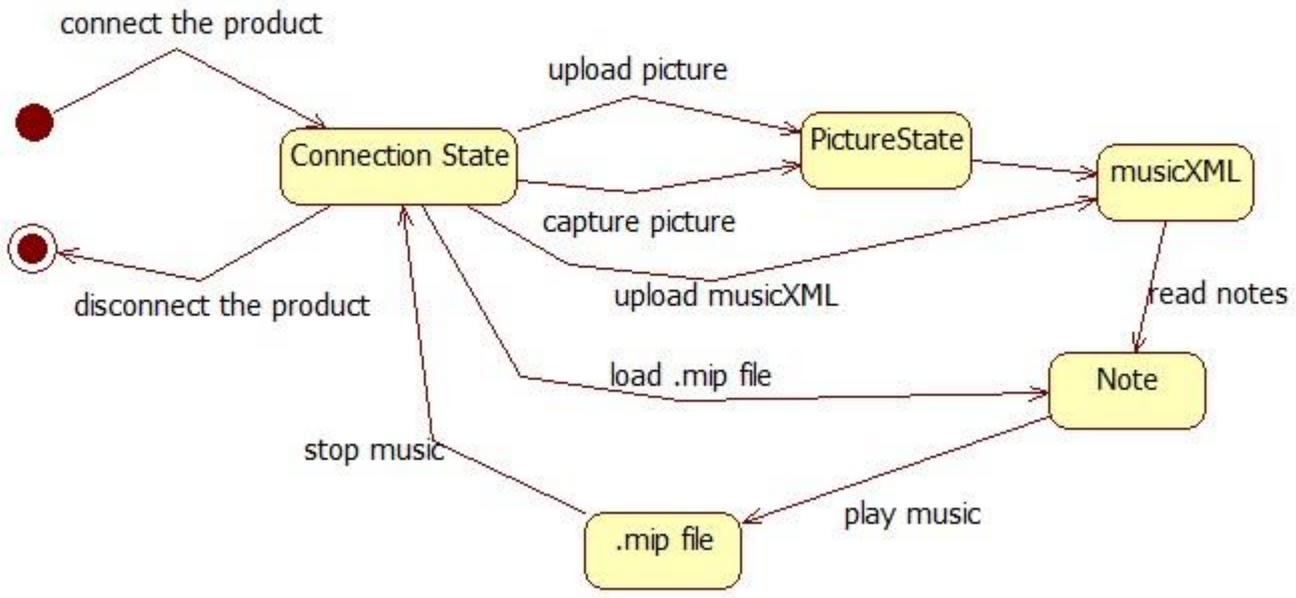


Figure 7: State Transition Diagram

6. Planning

6.1 Team Structure

Selim Temizer - Advisor

Serdar Çiftçi – Advisor

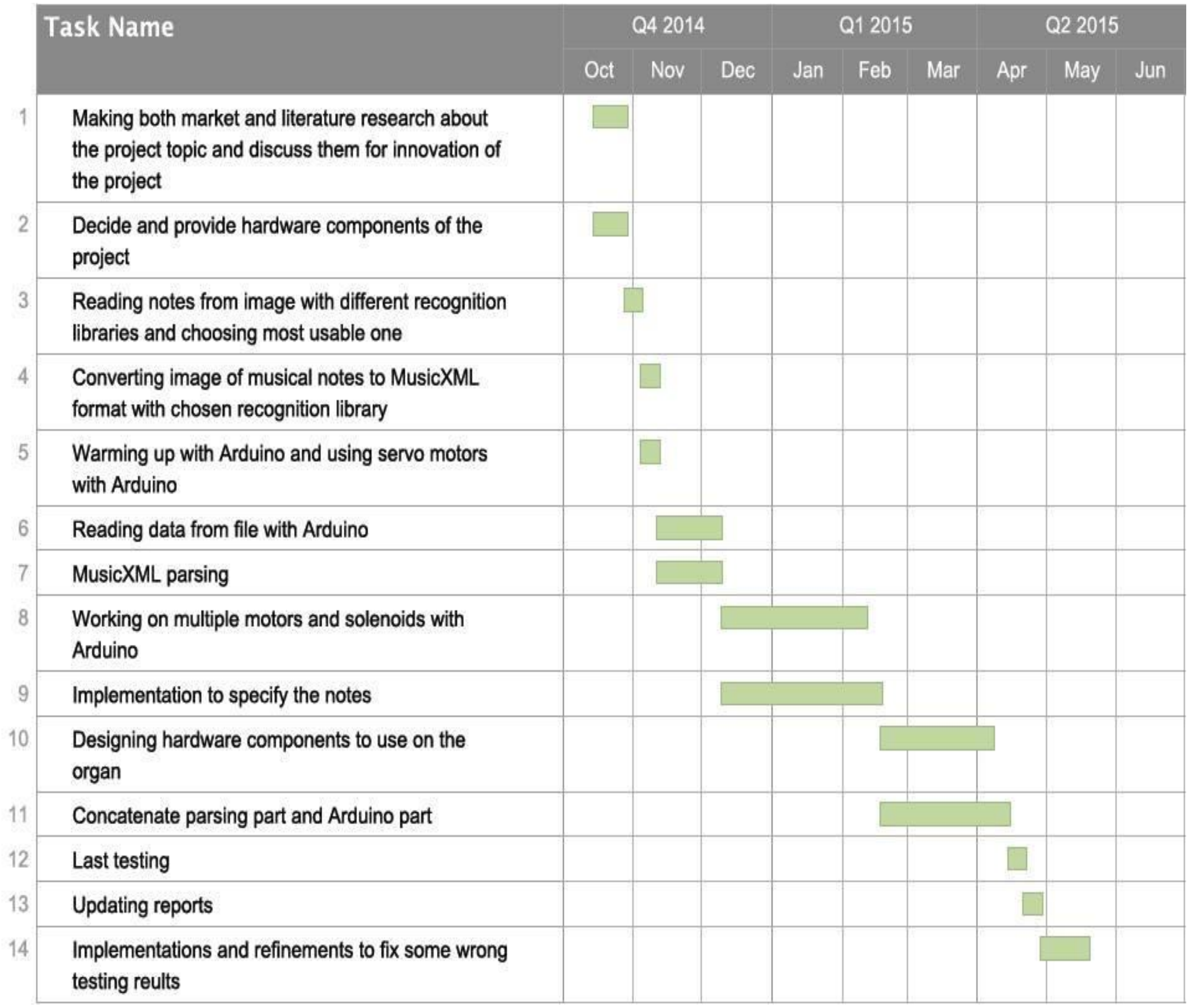
Duygu Abadan - Researcher, Developer, Hardware Designer

Anıl Arpacı - Researcher, Developer, Software Designer

Yağmur Ertaş - Researcher, Developer, Software Designer

Baler İlhan - Researcher, Developer, Hardware Designer

6.2 Estimation



	Task Name	Q4 2014			Q1 2015			Q2 2015		
		Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
1	Making both market and literature research about the project topic and discuss them for innovation of the project	🟩								
2	Decide and provide hardware components of the project	🟩								
3	Reading notes from image with different recognition libraries and choosing most usable one	🟩	🟩							
4	Converting image of musical notes to MusicXML format with chosen recognition library		🟩							
5	Warming up with Arduino and using servo motors with Arduino		🟩							
6	Reading data from file with Arduino		🟩	🟩						
7	MusicXML parsing		🟩	🟩						
8	Working on multiple motors and solenoids with Arduino			🟩	🟩	🟩				
9	Implementation to specify the notes			🟩	🟩	🟩				
10	Designing hardware components to use on the organ					🟩	🟩	🟩		
11	Concatenate parsing part and Arduino part					🟩	🟩	🟩		
12	Last testing							🟩		
13	Updating reports							🟩		
14	Implementations and refinements to fix some wrong testing reults							🟩	🟩	

Figure 8: Gantt Chart

6.3 Process Model

The project will be follow agile development model as shown in Figure xxx. Firstly, the research is made related about similar projects and methodologies. After the researching, the next step is planning the project and design. Then, implementation will come. After completing the implementation, the project's hardware part and software implementation part are combined. During the implementation and after, the project will be tested once two weeks or three weeks periodically.

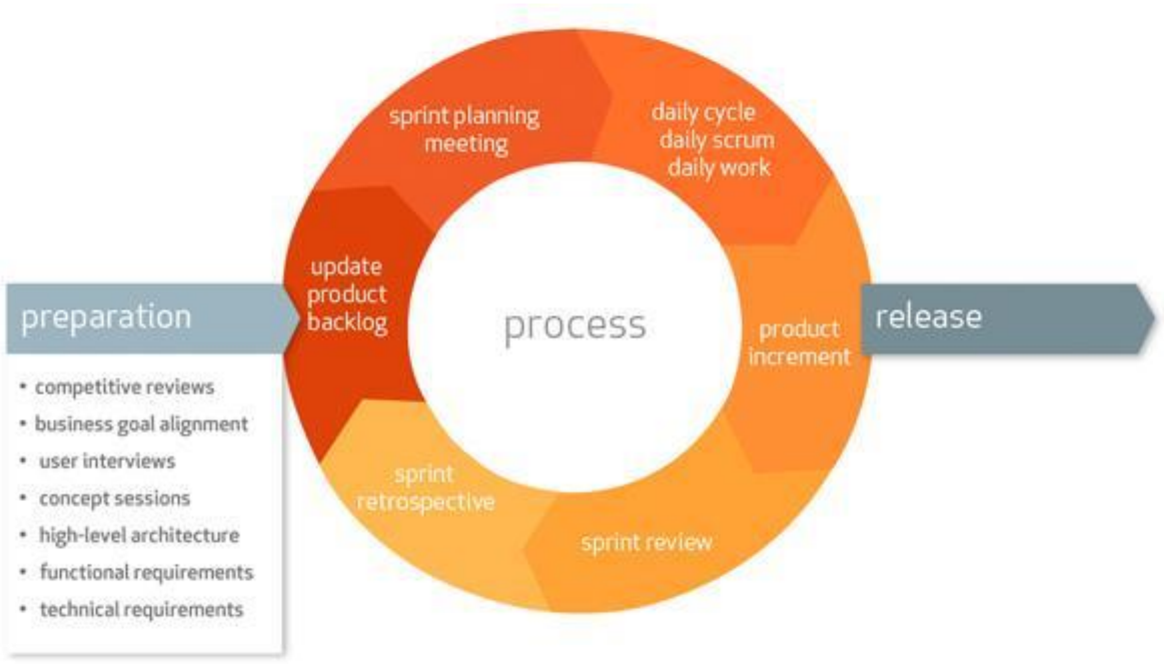


Figure 9: Agile Method Development

7. Conclusion

This Software Requirement Specification document is prepared to give requirement details of the project "Building Musical Instrument Playing Systems". Firstly, general information and definitions of the project are given. Then, all the functional, non-functional and interfaces requirements are specified detailed. Moreover, data models and behavioral models are shown in the document. Finally, planning and development stages of the product are given. This document will be helpful a basis for design and development of the project to be developed.

8. Supporting information

8.1 Index

No index is available.

8.2 Appendices

No appendix is available.