|  |
| --- |
| University of michigan |
| CIS 376 Term Project |
| Harry Potter and the Secret of the Chords |
|  |
| **By Corey Maylone and Scott Smereka** |
| **12/21/2010** |

|  |
| --- |
|  |

Contents

[1.0 Overview/Introduction 1](#_Toc280700173)

[2.0 Architectural Design of the Software 2](#_Toc280700174)

[2.1 Class Diagram 2](#_Toc280700175)

[3.0 CMMI Assessment of the Project Development 3](#_Toc280700176)

[3.1 Level 1 3](#_Toc280700177)

[3.2 Level 2 3](#_Toc280700178)

[4.0 V&V, SQA, and Testing Processes used in your Project 4](#_Toc280700179)

[4.1 Interface Prototype V&V 4](#_Toc280700180)

[4.2 Design & Testing 4](#_Toc280700181)

[4.3 Final Refinement & Testing 4](#_Toc280700182)

[5.0 Measurements and Metrics 5](#_Toc280700183)

[5.1 Intermediate COCOMO 5](#_Toc280700184)

[5.2 Function Point Analysis 5](#_Toc280700185)

[6.0 Documentation and Development Tools 7](#_Toc280700186)

[6.1 Visual Studio 7](#_Toc280700187)

[6.2 Microsoft Word 7](#_Toc280700188)

[7.0 General Principles used for your User Interface Design 8](#_Toc280700189)

[8.0 KLM of Major F Scale Generation 10](#_Toc280700190)

[9.0 Software Lifecycle 11](#_Toc280700191)

[10. Appendices (code listing, example screens, and other applicable info) 12](#_Toc280700192)

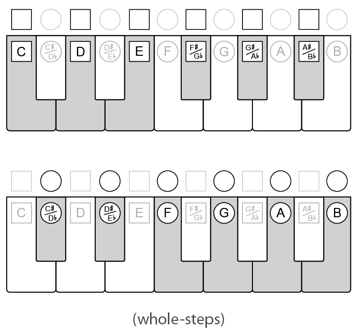
[10.1 Function Point Cost Drivers 12](#_Toc280700193)

[10.2 Auto-Generated Metrics (Visual Studio) 12](#_Toc280700194)

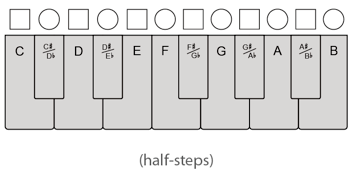
# 1.0 Overview/Introduction

Harry Potter and the Secret of the Chords is a program implemented in C# that outputs the notes in a scale based on the chosen root note and scale type. It is currently configured to output the two most common and fundamental scales in western music: The major scale and the minor scale. There are many other scales in music but they are beyond the scope of this program.

In music theory the major scale and the minor scale are constructed from variations of whole steps and half steps. On the piano a whole-step would be moving two keys over (including the black keys). A half-step would be moving one key over (including the black keys).



Figure



Figure

A major scale is constructed using the form:

* Whole, Whole, Half, Whole, Whole, Whole, Half
* Example C Major: C D E F G A B

While a minor scale is constructed using the form:

* Whole, Half, Whole, Whole, Half, Whole, Whole
* Example F♯ Minor: F♯ G♯ A B C♯ D E

Our program uses these formulas to construct the scale correctly.

2.0 Architectural Design of the Software

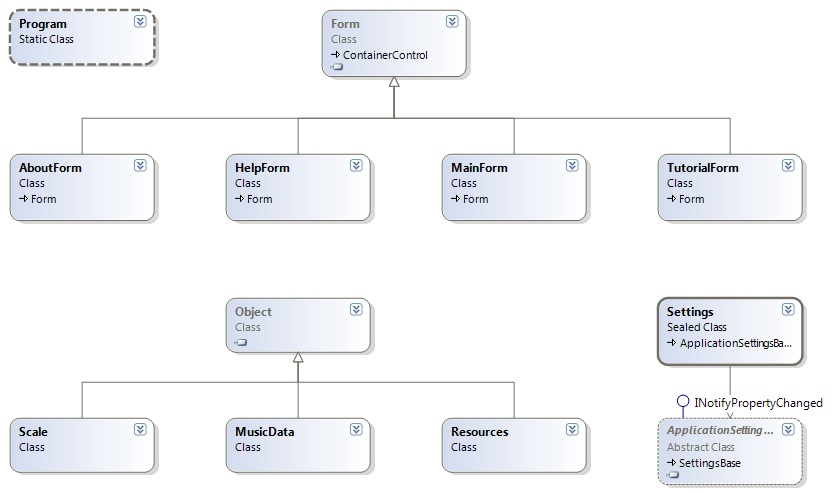
This program uses an Object Oriented Architecture to achieve its goals. The classes are broken down into two categories: Forms and Objects. There are four major form classes for the user interface:

* **MainForm:** Displays the main user interface. This is where the user should spend most of his or her time. All other forms are accessed from this form.
* **HelpForm:** Displays help documentation.
* **TutorialForm:** Displays a video of how to operate the software.
* **AboutForm:** Displays general information about this software.

The program also contains two major object classes:

* **Scale:** This class is where all the work is done by the program. It generates the scale based on user input.
* **MusicData:** This class is used to generate the inputs that the user can select.

## 2.1 Class Diagram



Figure

# 3.0 CMMI Assessment of the Project Development

Our project is still in level 1 of CMMI but is moving towards level 2. Below you can see a breakdown of our CMMI Assessment.

## 3.1 Level 1

*Ad Hoc, Chaotic, Individual Effort*

* No Key process areas
  + - In the beginning our project was unstructured and Chaotic. It is starting to move towards less individual effort and more towards a team like structure.

## 3.2 Level 2

*Focus is on basic project management*

* Software configuration management
  + Our project did not use any software configuration management. This would have been useful for keeping the software updated while working remotely. For the most part this was not an issue because we worked on tasks that could operate independently from one another.
* Software quality assurance
  + Our project’s SQA plan went from testing our initial user interface to testing other major software functions. When project was further refined it went through this same testing again to verify the software is of high quality.
* Software subcontract management
  + This project was not large enough to warrant the need for subcontracts.
* Software project tracking and oversight
  + This KPA was chaotic and not utilized in project management. We just gauge our progress by what we have completed and where we need to be. There was no use of project tracking.
* Software project planning
  + For project planning we used a simplified Rapid Prototype approach. We started with a conceptual prototype then specification, design, implementation, integration then release. The specification and design phase were molded into one phase in our project. Also, the implementation and integration phases were condensed into one phase of our project development planning[[1]](#footnote-1).
* Requirements management
  + Our requirements management tasks were at a very basic level. Seeing that we are both customer and developer it was not difficult to keep track of what our project requirements were. This key process area could use more work.

# 4.0 V&V, SQA, and Testing Processes used in your Project

Our program uses a Rapid Prototyping and was dynamic verified and validated as such. The testing process was continuous throughout the program.

## 4.1 Interface Prototype V&V

Our first goal was to quickly create the user interface prototype before we did the underlying calculations. We insured that our prototype would conform to the requirements of the program. So it would have to have a scale type (major or minor), the various root notes that can be selected and some type of output. Our initial prototype looks like this:

Figure

## 4.2 Design & Testing

At this point we had the basis for our program. We used this prototype interface to help implement our scale class. If you recall the scale class is where the program does its calculations and returns the scale so that it can be outputted.

Our testing at this point can be broken down into two stages:

* Interface testing
  + Testing interface functionality
  + Insure arguments are being passed properly after user selection
  + User interface is properly outputting data
* Scale Calculations
  + Testing that the scale class is returning the correct scale as defined by user input
    - Each scale was selected and tested individually for proper output based on our knowledge of music theory and on an expert source[[2]](#footnote-2)

## 4.3 Final Refinement & Testing

In this stage we reworked the interface to be more users friendly and accessible while still meeting our requirements. So it had to be tested against the pervious criteria again. We made our final assessment of our program to insure it was meeting our original requirements and is in a release worthy condition.

# 5.0 Measurements and Metrics

Metrics were estimated using COCOMO and function point analysis and then compared to the actual final metrics obtained through Visual Studio's auto metric generations. As you can see the estimates were very close to the actual values.

|  |  |  |  |
| --- | --- | --- | --- |
| Metrics | Obtained Using | Estimates | Actual |
| Person Months (PM) | COCOMO | 1.31 PM | 1.0 PM |
| Lines of Code (LOC) | FP to LOC Table | 648 LOC | 591 LOC |
| Function Point (FP) | FP Analysis | 21.6 FP | 19.7 FP |

## 5.1 Intermediate COCOMO

This program runs by itself and therefore is classified as an Organic program and we will use the following equations.

|  |  |  |
| --- | --- | --- |
| Cost Drivers (Non-Nominal) | Rating Value | Reasoning |
| Complexity | 0.85 (Low) | Very few classes and functions will be needed to accomplish the goals of this program. |
| Execution Time Constraint | 1.11 (High) | Main functions must be able to generate the desired output very quickly. This is due to the design of the user interface calculating values interactively. |
| Use of Software Tools | 0.91 (High) | The high use of advanced tools allow for faster development. [[3]](#footnote-3) |

## 5.2 Function Point Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Equations and Results |  |  |  |  |  |
| Value Adjustment Factor (VAF) |  | = |  | = | 0.80 |
| Unadjusted Function Point (T) |  | = | See table above | = |  |
| Adjusted Function Point (VFP) |  | = |  | = |  |
| Lines of Code (LOC) |  | = | 30 \* 21.6 | = | 648 |

|  |  |  |
| --- | --- | --- |
| Major Functions |  |  |
| Function | **Domain Value** | **Details** |
| Get Major or Get Minor Roots | ILF | Functions used to access files containing musical data. |
| Get Scale or Generate Scale | EI | Given user input, generates a scale to be outputted to user. |
| Get Half Step or Get Whole Step | EO | Calculates a half or whole step between notes. |
| Save Scale | EI | Allows user to save a generated scale to a desired location. |
| View Help or View Tutorial | EO | Prompts user with helpful information or tutorials. |
| Scale Type or Root Changed | EI | User inputs or changes the selected root or scale type. |
| Update Application | EQ | Check against an online server for any new updates. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Domain Values | Count |  | Simple | Final |
| External Inputs (EI) | **3** | X | **3** | **9** |
| External Outputs (EO) | **2** | X | **4** | **8** |
| External Inquiries (EQ) | **1** | X | **3** | **3** |
| Internal Logical Files (ILF) | **1** | X | **7** | **7** |
| External Interface Files (EIF) | **0** | X | **5** | **0** |
| Count Total (T) |  |  |  | **27** |

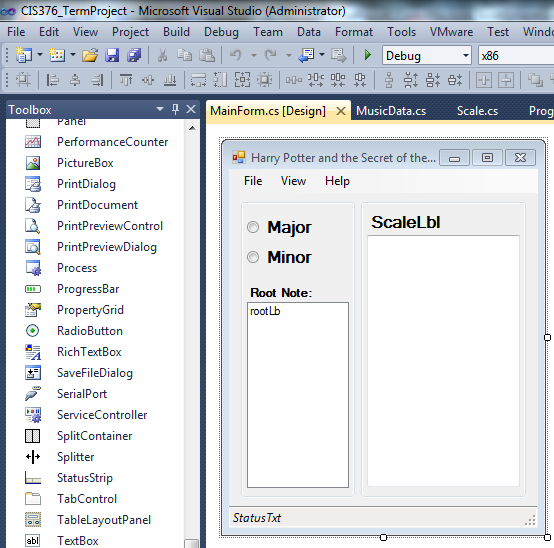
|  |  |
| --- | --- |
| Factor | Value |
| Does the system require reliable backup and recovery? | 0 |
| Are data communications required | 0 |
| Are there distributed processing functions | 0 |
| Is performance critical | 2 |
| Will the system run in an existing heavily utilized operational environment? | 2 |
| Does the system require on-line data entry | 0 |
| Does the on-line data entry require input transaction to be built over multiple screens or operations? | 0 |
| Are the master files updated on-line | 2 |
| Are the inputs, outputs, files, or inquiries complex | 1 |
| Multiple User-Site\Is the internal processing complex? | 0 |
| Is the code designed to be reusable? | 2 |
| Are conversion and installation included in the design? | 1 |
| Is the system designed for multiple installations in different organizations? | 0 |
| Is the application designed to facilitate change and ease of use by the user? | 5 |
| Total | 15 |
| Value Adjustment Factor (VAF) | =0.8 |

# 6.0 Documentation and Development Tools

## 6.1 Visual Studio

Visual Studio has several features used for development and documentation of a program. It was chosen as our main development tool because of its high compatibility with windows-based GUI programs.

**GUI Development** - Design of the GUI using Visual Studios GUI builder and tools.



Figure

**C# Coding** - Classes and interface coding done from using Visual Studio's compiler.

**Metrics** - Auto generation of metrics (such as LOC) using ultimate edition of visual studio

**Diagrams** - Creation of class diagrams using Visual Studio's diagram creation.

**Debug and Testing** - Errors and variable tracking done using the built-in debugger.

## 6.2 Microsoft Word

Microsoft word was used to document our entire project due to its flexibility to work with many different formats of information (such as pictures, formatting, and tables). Microsoft word also can auto generate table of contents, cover pages, and other useful documentation.

# 7.0 General Principles used for your User Interface Design

**User Familiarity -** Use of common windows components allows for the user to find familiarity within the designed GUI. Things such as the file menu, status bar, and windows controls are found on almost any windows program and need little explanation to the common user.

**File menu**

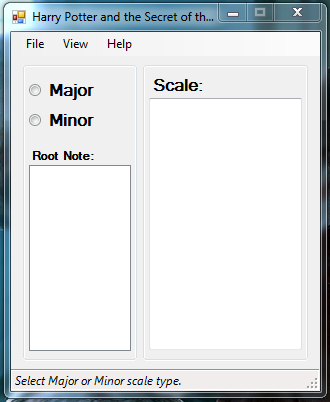
**Status Bar**

**Window Controls**

**Radio Button: Scale Type Selection**

**List Box: Root Note Selection**

**List Box: Scale Output**



Figure

**User in Control** - The GUI was designed to be as easy to use as possible for the user. Some examples:

* Large unambiguous input and output controls.
* Menu items are not nested and easy to find.
* Easy access to tasks with right click (Such as export scale to file).
* Instructions always displayed in status bar.

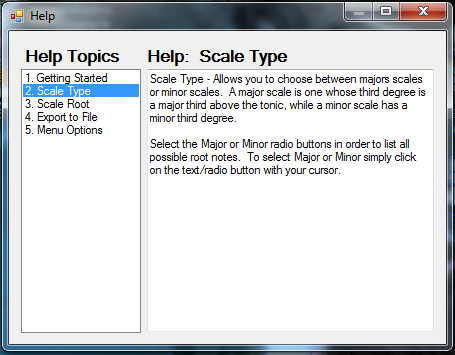
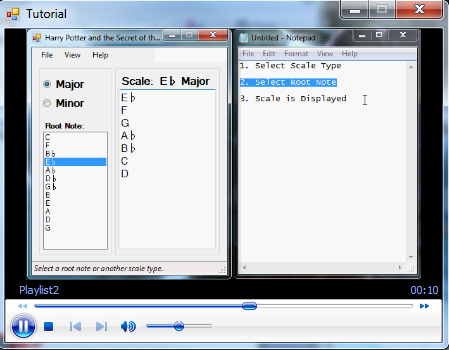
**Consistency** - All forms contain a consistent windows based theme and controls. Selecting different scales displays the information in a consistent manner and different ways to access the same features have the same results. All of these help keep the program consistent and familiar to the general user.

**Minimal User Memory** - All user input options are displayed in the GUI so there is never a need to memorize input. When selecting Major or Minor a list of possible root notes will be automatically generated for the user to choose from. Also since all menu items are a single layer deep the user can always find what he/she is looking for instead of memorizing an option's location.

**Minimal Surprise** - The GUI is designed to have no surprises which may leave the user confused. When comparing the GUI to a standard test it is much like multiple choice rather than fill in the blank. Fill in the blank, where the user must think about what to input, can leave the user feeling confused or surprised by the results. Multiple choice has all the possible answers in front of you and therefore the user will not be as surprised by the outcome of the selected action. Also help is available to help the user understand anything he/she may be surprised by or confused about.

**User Guidance** - Several techniques have been used to provide help to the user.

* A Status Bar - Displays instructions on the next step to generating a scale.
* Help Menu - Displays information in text form about each function of the program and how to use it.
* Tutorial - A video tutorial designed to show the user graphically exactly how to use the program.

Figure

**User Feedback** - If an error occurs a message box will be displayed with an explanation of what kind of error or possibly what may have caused the error to occur.

**Recoverability** -In addition to error messages (from user feedback) a next best action is recommended to the user. For example if an error occurred selecting Major or Minor scale types the error message "An Error has occurred while selecting scale type, please reselect a Scale Type." would be displayed.

**User Diversity** - The GUI was built using windows default controls, therefore default language and window settings (such as size) should automatically be accounted for when the program is installed on the host machine. Root notes, sharps, flats, and Major/Minor should be universally known (much like numbers) and therefore do not need to be changed with the language. Only the help menu text will not be dynamically changed with the operating system, but since there is a graphical tutorial available the user should have no trouble finding help. This graphic tutorial can also help those who may be illiterate or have trouble reading text.

**Confirmation of Destructive Request** - There are no destructive actions possible by this program and therefore no reason for a confirmation prompt.

# 8.0 KLM of Major F Scale Generation

|  |  |  |
| --- | --- | --- |
| Description | Operation | Time (Seconds) |
| Reach for mouse | H[mouse] | 0.40 |
| Move pointer to Major Radio Button | P[Major RB] | 1.10 |
| Click on Major Radio Button | K[mouse] | 0.20 |
| Move pointer to F Root Note | P[F Root Note] | 1.10 |
| Click on F Root Note | K[mouse] | 0.20 |
|  | **Total** | **3 seconds** |

# 9.0 Software Lifecycle

Figure

# 10. Appendices (code listing, example screens, and other applicable info)

## 10.1 Function Point Cost Drivers

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cost Drivers | Very Low | Low | Nominal | High | Very High | Extra High |
| Required Software Reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 |  |
| Database Size |  | 0.94 | 1.00 | 1.08 | 1.16 |  |
| Product Complexity | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| Execution Time Constraint |  |  | 1.00 | 1.11 | 1.30 | 1.66 |
| Main Storage Constraint |  |  | 1.00 | 1.06 | 1.21 | 1.56 |
| Virtual Machine Volatility |  | 0.87 | 1.00 | 1.15 | 1.30 |  |
| Computer Turnaround Time |  | 0.87 | 1.00 | 1.07 | 1.15 |  |
| Analyst Capabilities | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 |  |
| Applications Experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 |  |
| Programmer Capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 |  |
| Virtual Machine Experience | 1.21 | 1.10 | 1.00 | 0.90 |  |  |
| Programming Language Experience | 1.14 | 1.07 | 1.00 | 0.95 |  |  |
| Use of Modern Programming Practices | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 |  |
| Use of Software Tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 |  |
| Required Development Schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 |  |

## 10.2 Auto-Generated Metrics (Visual Studio)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Form  Member | Maintainability Index | Cyclomatic Complexity | Depth of Inheritance | Class Coupling | LOC |
| Project | **63** | **117** | **7** | **62** | **591** |
| AboutForm | **60** | **6** | **7** | **19** | **47** |
| HelpForm | **57** | **16** | **7** | **23** | **96** |
| MainForm | **52** | **28** | **7** | **50** | **284** |
| MusicData | **73** | **8** | **1** | **1** | **36** |
| Program | **81** | **1** | **1** | **3** | **3** |
| Scale | **49** | **51** | **1** | **3** | **95** |
| Tutorial | **66** | **7** | **7** | **18** | **30** |

1. See section 9.0 for a diagram of our software life-cycle [↑](#footnote-ref-1)
2. <http://www.studybass.com> This website is tailored towards learning bass guitar but also contains detailed information on music theory. [↑](#footnote-ref-2)
3. See Appendices for a list of all Intermediate COCOMO cost drivers. [↑](#footnote-ref-3)