SE 3XA3: Software Requirements Specification
Title of Project

Team #, Team Name
Student 1 name and macid
Student 2 name and macid
Student 3 name and macid

November 9, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

1 Introduction

1.1 Overview

XPYCharts is a python based graphing library that is a recreation of the Jcharts graphing library available for Java developers. XPYCharts supports the production of scatter plots, line graphs and pie charts that can be used for statistical analysis and in a learning environment for data visualization.

1.2 Context

This document will serve as the Module Guide (MG) for the SRS document of XPYCharts. The MG shows a decomposition of the modules of the software system and how these modules accomplish the functional/non-functional requirements previously stated within the SRS. The MG will also show how the software adheres to the principals of information hiding and data independence in order to create a modular structure. The MG document is followed by the Module Interface Specification (MIS) that describes from an external viewpoint the intended behavior of the modules functions.

1.3 Design Principals

The software system of XPYCharts is built with consideration to several design principals including Modular decomposition, data independence, encapsulation and information hiding. In addition the decomposed modules of the system interact with each other through a structure that ensures low coupling and high cohesion so that modules stay as independent of each other as possible. This type of design will allow for easier software maintenance and modification in the future releases of the product. Similarly information hiding allows secrecy to exist over the module implementation making it easier to use the modules at a higher level without having to worry about the way they were implemented. This is especially important to XPYCharts as it is a library and therefore developers using the library should not have to know how the library modules are implemented to be able to use them.

1.4 Document Structure

The document is divided into 7 major sections including the Introduction above. Listed below is each of the major sections (excluding Introduction) with a brief description of the information that they contain:

- Anticipated and Unlikely Changes (Section 2): This section outlines the changes that are likely and unlikely to happen. This section is split into 2 subsections covering all likely and unlikely changes.

- Module Hierarchy (Section 3): This section provides an outline of the module design by detailing the module hierarchy and providing a table describing the secrets of the modules.
- Connection between Requirements and Design (Section 4): This section maps out the modules that implement each of the requirements as presented in the SRS document.
- Module Decomposition (Section 5): This section decomposes the modules by their secrets and states the design decisions that are satisfied by these secrets. The section focuses on the purpose of the modules existence rather than how it is implemented.
- Traceability Matrix (Section 6): This section provides two tractability matrices that allow tracking of which requirements were satisfied by which modules and which modules will take care of which anticipated changes.
- Uses Hierarchy Between Modules (Section 7): This section shows the uses relationships between the modules of the system.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section **??**, and unlikely changes are listed in Section **??**.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format of the initial input data.

...

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

...

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table **??**. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

...

Level 1	**Level 2**
Hardware-Hiding Module	Output Window Module
Behaviour-Hiding Module	Input Format Module Exception Raising Module Output Format Module Polynomial Equations Module
Software Decision Module	Data Structure Module Polynomial Solver Module Plotting Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table **??**.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden

by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M??)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

5.2.1 Input Format Module (M??)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Your Program Name Here]

5.2.2 Etc.

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

5.3.1 Etc.

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	Modules
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure **??** illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

Figure 1: Use hierarchy among modules