­

Software Requirements Specification­

for

Pro-Crastinator

Version 1.0

Prepared by

Group Name: Once We Were Programmers

|  |  |  |
| --- | --- | --- |
| Spencer Ross | <student #> | s.ross@wsu.edu |
| Sarah Mathes | 11065825 | sarah.robison-mathes@wsu.edu |
| Kyle Stennfeld | 10722379 | k.stennfeld@wsu.edu |

|  |  |
| --- | --- |
|  |  |
| Date: | October 25,2019 |
|  |  |
|  |  |
|  |  |

Contents

Revisions iii

1 Introduction 1

1.1 Document Purpose 1

1.2 Product Scope 1

1.3 Intended Audience and Document Overview 1

1.4 Definitions, Acronyms and Abbreviations 1

1.5 Document Conventions 1

1.6 References and Acknowledgments 2

2 Overall Description 3

2.1 Product Perspective 3

2.2 Product Functionality 3

2.3 Users and Characteristics 3

2.4 Operating Environment 3

2.5 Design and Implementation Constraints 4

2.6 User Documentation 4

2.7 Assumptions and Dependencies 4

3 Specific Requirements 5

3.1 External Interface Requirements 5

3.2 Functional Requirements 6

3.3 Behaviour Requirements 6

4 Other Non-functional Requirements 7

4.1 Performance Requirements 7

4.2 Safety and Security Requirements 7

4.3 Software Quality Attributes 7

5 Other Requirements 8

Appendix A – Data Dictionary 9

Appendix B - Group Log 10

Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| 1.0 | Sarah Mathes, Spencer Ross, Kyle Stennfeld | First version of system require specification requirements document. System is in design phase. | 10/25/2019 |

# 

# Introduction

College students are in need of a concise schedule organizer in order to meet the demands of their busy schedules. The goals of this system are:

1. To be a time management system that is unencumbered by unnecessary features or complicated graphic user interface.
2. To improve the lives of students by organizing their classes, homework, and tests.

This system is intended to be a simple interface that the student can customize for their own workload.

This document outlines the high-level requirements for this system. The use-case of the system is also included, along with the overall description, and other non-functional requirements.

## Document Purpose

This document describes the requirements for the web application Pro-Crastinator version 1.0. The first section serves to introduce the scope of the system, its audience, and an overview of the document itself. The second section details overall description of of the system, including the operating environment, system functionality, and user documentation. The third section details the system’s hardware and software interfaces, and user interfaces.

The scope of this document encompasses all features of the system, and includes the system log-in, class management, grade calculation, and assignment organization. The system log-in is the process by which the student logs into their schedule. Class management is the sub-system where a student adds a class to their weekly schedule. Grade calculation is the sub-system where a student can edit the breakdown of their class grades and calculate a theoretical final grade based on the syllabus and submitted assignments. Assignment organization is the sub-system by which a student can add an upcoming assignment or test to their schedule.

## Product Scope

The software described in this document is the web application Pro-Crastinator. Pro-Crastinator is a schedule planner intended to be used by students in order to manage their school workload. The software system saves a user’s information, primarily their name, classes, and assignments, and recalls them upon the user logging in. Once logged in, the user can view a calendar of their classes and assignments, edit their class schedule, add a new assignment, edit existing assignments, and calculate their class grade.

Although many student planner applications exist, one of the main goals of Pro-Crastinator is to have a simple graphic user interface where the student can easily edit their class schedule and assignments from day to day. Upon logging in, the student is presented with options to either view, edit, or add to their class workload.

One of the core advantages to this system is that the user will not have to navigate through several menus, calendars, or course webpages in order to organize their workload. This lightens the burden of needing to mentally keep track of multiple assignments and tests across a number of classes. The system is a simple way to organize and then visually see the user’s school workload.

## Intended Audience and Document Overview

The primary intended audience for this document are developers of the system Pro-Crastinator. The purpose of this document is to formally state the functionality of this software system and its subsystems. Developers can refer to this document during development in order to solidify core goals, and to maintain consistency. The section that is most relevant to developers is section 3.1.3, Software Interfaces, as well as 2.2, Product Functionality.

Similarly, project managers should use this document to drive consistency as the system is developed and updated. Project managers should primarily refer to the section 2.1, Product Perspective, and 2.2, Product Functionality, in order to keep the main goals of Pro-Crastinator in mind. They should also familiarize themselves early in development with the Use Case View in section 3.3.1, which outlines the functionality of the system and the relationships of its subsystems.

The secondary audience for this document is the professor of Computer Science 320: Introduction to Computer Science, Dr. Xinghui Zhao, who will be reviewing and grading the development of Pro-Crastinator. The section that is most useful for Dr. Zhao is section 3.3.1, the Use Case View, because it is the easiest way to visually understand the relationships between the system and its users and its subsystems. Additionally, section 2.2, Product Functionality, will be useful for Dr. Zhao to refer to during system development, and when grading the final version of Pro-Crastinator.

This document is also intended for product testers of Pro-Crastinator. If at any point product testers have questions about the intended functionality of the system, they should refer to this document. Product testers should start with the use case view, section 3.3.1 which will give them an overall view of the system’s operation. They should then move on to section 3.1.1, User Interfaces, which provides guides to the user interface and its operation.

## Definitions, Acronyms and Abbreviations

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.

TO DO: Please provide a list of all abbreviations and acronyms used in this document sorted in alphabetical order.>

## Document Conventions

Formatting coventions in this document follow the standard IEEE formatting requirements. Standard font for this document is Arial, size 11, with one inch margins. Lines are single spaced, with double spaces between paragraphs. New paragraphs are not indented. Comments are italicized.

## References and Acknowledgments

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document.

TO DO: Use the standard IEEE citation guide (attached) for this section.>

# Overall Description

## Product Perspective

The Pro-Crastinator is a new, self contained product that utilizes existing Javascript libraries. The user is able to update their personal info, including their username, email address, and password. They can then add a class to their schedule, which Pro-Crastinator saves in the server.

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. In this part, make sure to include a simple diagram that shows the major components of the overall system, subsystem interconnections, and external interface. In this section it is crucial that you will be creative and provide as much information as possible.

TO DO: Provide at least one paragraph describing product perspective. Provide a general diagram that will illustrate how your product interacts with the environment and in what context it is being used, i.e., context diagram.>

## Product Functionality

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, will be effective.

TO DO:

1. Provide a bulleted list of all the major functions of the system

2. **(Optional)** Provide a Data Flow Diagram of the system to show how these functions relate to each other. This is useful when there is a clear sequence for the functions being performed.>

## Users and Characteristics

<Identify the various users that you anticipate will use this product. Users may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.

TO DO:

1. Describe the pertinent characteristics of each user. Certain requirements may pertain only to certain users.

3. Distinguish the most important users for this product from those who are less important to satisfy.>

## Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist. In this part, make sure to include a simple diagram that shows the major components of the overall system, subsystem interconnections, and external interface

TO DO: As stated above, in at least one paragraph, describe the environment your system will have to operate in. Make sure to include the minimum platform requirements for your system. >

## Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer’s organization will be responsible for maintaining the delivered software).

TO DO: In this section you need to consider all of the information you gathered so far, analyze it and correctly identify relevant constraints.>

## User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.

TO DO: You will not actually develop any user-manuals, but you need to describe what kind of manuals and what kind of help is needed for the software you will be developing. One paragraph should be sufficient for this section.>

## Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.

TO DO: Provide a short list of some major assumptions that might significantly affect your design. For example, you can assume that your client will have 1, 2 or at most 50 Automated Banking Machines. Every number has a significant effect on the design of your system. >

# Specific Requirements

## External Interface Requirements

### User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., Cancel) that will appear on every screen, error message display standards, and so on. Define the software components for which a user interface is needed.

TO DO: The least you can do for this section is to describe in words the different User Interfaces and the different screens that will be available to the user. Optional: You may also provide an initial Graphical User Interface design (does not have to be final).>

### Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware. You are not required to specify what protocols you will be using to communicate with the hardware, but it will be usually included in this part as well.

TO DO: Please provide a short description of the different hardware interfaces. If you will be using some special libraries to communicate with your software mention them here. In case you have more than one hardware interface divide this section into subsections.>

### Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems (Windows? Linux? Etc…), tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.

TO DO: The previous part illustrates some of the information you would usually include in this part of the SRS document. To make things simpler, you are only required to describe the specific interface with the operating system.>

### Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.

TO DO: Do not go into too much detail, but provide 1-2 paragraphs were you will outline the major communication standards. For example, if you decide to use encryption there is no need to specify the exact encryption standards, but rather, specify the fact that the data will be encrypted and name what standards you consider using. >

## Functional Requirements

*< Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. This section is the direct continuation of section 2.2 where you have specified the general functional requirements. Here, you should list in detail the different product functions with specific explanations regarding every function.*

*TO DO: Break the functional requirements to several functional areas and divide this section into subsections accordingly. Provide a detailed list of all product operations related to these functional areas.*

## Behaviour Requirements

### Use Case View

<A use case defines a goal-oriented set of interactions between external actors and the system under consideration.

TO DO: Provide a use case diagram which shows the entire system and all possible actors. Do not include detailed use case descriptions (these will be needed when you will be working on the Test Plan), but make sure to include a short description of what every use-case is, who are the actors in your diagram.>

# Other Non-functional Requirements

## Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.

TODO: Provide relevant performance requirements based on the information you collected from the client. For example you can say “1. Any transaction will not take more than 10 seconds, etc…>

## Safety and Security Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product’s design or use. Define any safety certifications that must be satisfied. Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements.

TODO:

* Provide relevant safety requirements based on your interview with the client or, on your expectation for the product.
* Describe briefly what level of security is expected from this product by your client and provide a bulleted (or numbered) list of the major security requirements.>

## Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.

TODO: Use subsections (e.g., 4.3.1 Reliability, 4.3.2 Portability, etc…) provide requirements related to the different software quality attributes. Base the information you include in these subsections on the material you have learned in the class. Make sure, that you do not just write “This software shall be maintainable…” Indicate how you plan to achieve it, etc.>

# Other Requirements

<This section is **Optional.** Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A – Data Dictionary

*<Data dictionary is used to track all the different variables, states and functional requirements that you described in your document. Make sure to include the complete list of all constants, state variables (and their possible states), inputs and outputs in a table. In the table, include the description of these items as well as all related operations and requirements.>*

Appendix B - Group Log

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist the Teaching Assistant to determine the effort put forth to produce this document>