

**Software Design Document Specification Template**

The Software Design Specification (SDS) sections provide you with guidelines related to the structure and the contents of the SDS document. The Software Design Specification document includes at least these sections.

For the project, your team may have good reasons for wanting to deviate from this proposed outline. If a section is not applicable in your case, do not delete it; instead, give the topic heading and write "Not applicable".

You will note that there is some overlap in the content between different documents (i.e. the User Requirements Specification Document and the Software Design Specification Document.) This redundancy allows each document to stand on its own.

***ONLY THE SECTION TITLES COLORED IN ORANGE ARE REQUIRED TO BE COMPLETED.***

***DO NOT DELETE THE SECTIONS YOU ARE NOT COMPLETING AS THEY ARE A PART OF THE DOCUMENT***

Contents

[1 Introduction 3](#_heading=h.gjdgxs)

[1.1 Purpose of this document 3](#_heading=h.30j0zll)

[1.2 Scope of the development project 3](#_heading=h.1fob9te)

[1.3 Definitions, acronyms, and abbreviations 3](#_heading=h.3znysh7)

1. [1.4 References 3](#_heading=h.2et92p0)

[1.5 Overview of document 3](#_heading=h.tyjcwt)

[2 System architecture description 3](#_heading=h.3dy6vkm)

[2.1 Overview of modules/components 3](#_heading=h.1t3h5sf)

[2.2 Structure and relationships 3](#_heading=h.4d34og8)

[2.3 User interface 3](#_heading=h.17dp8vu)

[2.4 User interface issues 3](#_heading=h.3rdcrjn)

[3 A detailed description of components (ONLY 2 ARE REQUIRED) 3](#_heading=h.26in1rg)

[3.1 X Component (or Class or Function ...) 3](#_heading=h.lnxbz9)

[3.2 Y Component (or Class or Function ...) 4](#_heading=h.35nkun2)

[4 Reuse and relationships to other products 4](#_heading=h.1ksv4uv)

[5 Design decisions and tradeoffs 4](#_heading=h.44sinio)

[6 Pseudocode for components 4](#_heading=h.2jxsxqh)

[7 Appendices (if any) 4](#_heading=h.z337ya)

[Software component template for section 3 4](#_heading=h.3j2qqm3)

# Introduction

## Purpose of this document

For our application, the overall goal of our SDS is to outline the intended structure and utilization of the functions, classes, and dependencies for the application. On top of this, the SDS will be used to display the graphically based prototype of our application’s user interface.

## Scope of the development project

The overall scope for this project, when deployed, is to create and manage a service-based application for Microsoft Windows desktops. To build the application, we are using Microsoft’s .NET Framework as a platform of development. We plan to implement a comprehensive WPF front end that allows users to view and edit their character build sheets, as well as create new ones. Due to the nature of Dungeons and Dragons, a primary goal for the team is to allow character creation for all players to be as simple and intuitive as possible. To achieve this, the project should have an emphasis on creating an intuitive user interface that is easy to use.

Since this application is highly reliant on data, we also plan to incorporate a local database for the application. Due to the nature of software development, we plan to use programming methods that ensure there is as little dependency as possible between our front end design and back end local database to allow expansion of the software to be more efficient after release.

As of now, the application’s scope is limited to the desktop it is being used on. Therefore, besides the initial install, the application will not directly require internet access or external connections to databases.

## Definitions, acronyms, and abbreviations

Be sure to alphabetize!

* D&D: Stands for Dungeons and Dragons, a popular roleplaying game that this character builder is designed for.
* NET Framework: Software framework developed by Microsoft.
* WPF: Windows Presentation Foundation. Graphical subsystem developed by Microsoft for use in rendering user interfaces in Windows based applications.
* SDS: Software Design Document.
* SRS: Software Requirements Specifications.

## References

This section will include technical books and documents related to design issues. Be certain that the references you give are complete and in the appropriate format.

## Overview of document

parts!

# System architecture description

## Overview of modules/components

This subsection will introduce the various components and subsystems.

## Structure and relationships

Make clear the interrelationships and dependencies among the various components. Structure charts can be useful here. A simple finite state machine can be useful in demonstrating the operation of the product. Include explanatory text to help the reader understand any charts.

## User interface

### 2.3.1. Opening Screen

Welcome screen, simple introduction, offer the choice of creation or viewing the previous character. The user can selection character creation to go to step 2.3.2 If select view previous character, go to 2.3.7

### 2.3.2. Character Name, General Information

Option 1 on the taskbar, titled General Char info.

Window one appears it contains boxes for name, player name, race is a drop-down selection of options from player’s handbook, the class is the same.

The right side window appears when class/race is selected, a brief summary of class/race, with a recommended selection stated under the brief blurb on the class.

### 2.3.3. Primary Stat Roller

Shows the primary stats, explains what they are, and has the ability for the player to enter what they rolled or to generate the stats there.

### 2.3.4. Secondary Stat Screen

Shows the secondary stats, explains to them when they are moused over or selected, and explains how the primary stats affect and mold the secondary stats.

### 2.3.5. Alignment Picking Screen

The alignment screen will be done in the style of the traditional chart “meme”, where typically funny or pop culture references are inserted to explain what each alignment stands for. Examples are Captain Picard for Lawful Neutral or the Joker for Chaotic Evil.

### 2.3.6. Background Screen (Bonds, Flaws, Ideals)

Allows for the input of Bonds, Flass, Ideals, and a Background. As these are only for aesthetic purposes, they don’t need a statistical background or large mechanical descriptions.

### 2.3.7. Finished Character Viewer

Complete design with user interactive windows, error messages, etc. Include all windows and messages that will be viewed by the user and also the ones that the user will be interacting with. This will be similar to the end state of 2.3.6.

**2.3.8.**

## User interface issues

This section will present the main principles of the product's user interface. Use the personas/use cases defined in the SRS to make specific examples. This section should not touch on technical details. You may want to include sketches and specific text messages.

The user interface is designed to be open for users to view their data as they move through the steps of character creation.

# A detailed description of components (ONLY 2 ARE REQUIRED)

## Created Character Class

|  |  |
| --- | --- |
| Identification | *<The unique name for the component and the location of the component in the system.>*  Created Character Class |
| Type | *<A module, a subprogram, a data file, a control procedure, a class, etc.>*  Class |
| Purpose | <*Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS but are implied or adjunct to formally stated SDS requirements.*>  Holds the overall information of the character that is being built with the application. |
| Function | <*What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.*>  As this is the class that stores the character information, it would hold the Character Name, Class, Race, Ability Scores (Strength, Dexterity, Constitution, Intelligence, Wisdom, Charisma), Secondary Attributes, etc.  This class would be provided information through the User Interface by the user and would return information from itself to be displayed by the User Interface. |
| Subordinates | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| Dependencies | *<How the component's function and performance related to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components.>*  This class will be able to have its contents written to a file. The class will also have the ability to read in information from the said type of file. |
| Interfaces | *<Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.>*  The interface relating to this will be in how the data listed above will be displayed, information for a character name, statistics, and choices will all be displayed by the interface. |
| Resources | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services. |
| Processing | The full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| Data | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary. |

## Ability Information Storage

|  |  |
| --- | --- |
| Identification | *<The unique name for the component and the location of the component in the system.>*  Ability Information Storage |
| Type | *<A module, a subprogram, a data file, a control procedure, a class, etc.>*  A data file |
| Purpose | *<Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS, but are implied or adjunct to formally stated SDS requirements.>*  To store the large number of options |
| Function | *<What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.>*  This will store the data for character options, for example this will hold feat choices that give bonuses to other statistics or variables. |
| Subordinates | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| Dependencies | *<How the component's function and performance related to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components.>*  The dependencies would be the feat and spell choices, as they are stored in the data files. Another dependency would be the stats that are changed or augmented by the choice of options from the data files. |
| Interfaces | *<Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.>*  The interface is all behind the scenes other than displaying the possible choices or the end result of those choices. All the actual data is stored in the back and pulled forward for when the user makes choices. |
| Resources | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services. |
| Processing | The full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| Data | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary. |

## Y Component (or Class or Function ...)

…

1. File storage: Containing data that will be options for the user to select.
2. Client Application: This is what the user will interact with.

Examples of a component are:

1. Database
2. Server
3. Client application
4. Search Application
5. Etc.

# Reuse and relationships with other products

For teams doing enhancement work, reuse is an important issue. Most enhancement work should focus on extending, rather than replacing, the design and product development from earlier semesters. For teams doing new development, reuse can also be an important strategy. In some cases, there is a freeware that could be incorporated. In other cases, there are existing modules or classes that could be adapted. Another possibility is the use of special tools that produce open-source results and thus permissible under the terms of this course.

This section should include the following subsections as appropriate:

* How reuse is playing a role in your product design
* How reuse is playing a role in your product implementation (and the motivation for changes)
* If you are not reusing material that is available, then give motivation for why it is being thrown out.

As we are building our project using Microsoft’s .NET framework and their proprietary development environment Visual Studio 2019, our new development project will be inherently using a wide variety of software and NuGet packages. These sources can be found listed below, along with their overall scope and implementation. Please note that the project templates and default NuGet packages are encapsulated within the X.X.X sections, and additional NuGet packages /reusable code will be explained for each project template under the section code X.X.X.X.

**4.1 Visual Studio Project Templates:**

**4.1.1 .NET Framework Web API Project -** This project template included in Visual Studio 2019 for C# development includes a great deal of basic “reusable code” that is often reused when extending or developing new projects. The project template includes important aspects such as incorporating the “Models Views and Controllers” design layout (MVC) and an option to include a SQL database (using entity framework) that can be used to store user information and credentials to use for API level authentication. The project, obviously, includes a basic template for an application API that can be used for user authentication and can be extended for the project’s required functions as it is developed.

**4.1.1.1 Swagger/Swashbuckle API Documentation -** One of the most important parts of developing an API is API documentation. When developing on Visual Studio 2019, it is possible to incorporate XML based comments and documentation that is handled by the development environment. However, to help automate API documentation, it has been decided that using a popular C# NuGet package “Swashbuckle” will be used for documentation. Swashbuckle, importantly, includes Swagger, a tool for software developers to test and organize their API and API functionality. The tool is designed to gather information on your API and document its functions and can be viewed when running your Web API project by navigating to the “Swagger” subpage in the application. The swagger UI allows developers (and potentially users if incorporated in deployment) to test and view API calls and has proven to be extremely effective for both developers and clients to test the application.

**4.1.2 .NET Framework DemoLibrary**

**4.1.3 .NET Framework WPF Project (non-web application) -** This project template is also included in Visual Studio 2019, and creates an extremely simple WPF user interface. The project generates a WPF front end along with a template XAML file which is the source code for the WPF application.

**4.1.2.1 Caliburn.Micro Nuget Package.**

# Design decisions and tradeoffs

Use this section to motivate any decisions that will help the reader understand the design that your team is using. This section can also capture good ideas that were abandoned and the reasons for leaving them out of the design.

# Pseudocode for components

Utilize the use cases to create pseudocode for components.

# Appendices (if any)

# 

# Software component template for section 3

The template given below suggests a reasonable structure for giving a thorough description of each component described in Part 3 of the SDS. The specific information depends in part on the design approach. Your team must adapt this template to your needs and describe it in section 3.1 of your SDS.

|  |  |
| --- | --- |
| Identification | *<The unique name for the component and the location of the component in the system.>* |
| Type | *<A module, a subprogram, a data file, a control procedure, a class, etc.>* |
| Purpose | *<Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS but are implied or adjunct to formally stated SDS requirements.>* |
| Function | *<What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.>* |
| Subordinates | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| Dependencies | *<How the component's function and performance related to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components.>* |
| Interfaces | *<Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.>* |
| Resources | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services. |
| Processing | The full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| Data | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary. |