1. **Outcome****(c)**: An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
2. **Interpretation**: This outcome states that students who have gone through the program should have the ability to design, implement and test a computing product based on the results of requirement analysis. We divide this outcome into three components as follows:
3. Design: Students should be able to design a computing system/product based on user requirements. The product design should be documented following a standard design document.
4. Implementation: Students should be able to implement the computing product according to the design document. The implementation program should follow a coding convention established at the beginning of the implementation phase. Students should observe the established coding practice in implementing the system.
5. **Student Work Sampled**: Student work samples are collected in CMSC 4513 – Software Design and Development. There are three types of student work corresponding to the three components of this outcome.
6. Design: Design documents are sampled. Design documents should be completed based on the following template (a detailed version of the template is given at the end of this document).

**Table of Contents for the Design Document**

0. Revision History

1. Introduction

1.1 System Overview

1.2 Supporting Materials

1.3 Definitions and Acronyms

2. Design Considerations

2.1 Assumptions

2.2 Constraints

2.3 Goals and Guidelines

3. Architecture

3.1 Overview

3.2 Strategies

3.3 Software Specification

3.4 Hardware Specification

4. High Level Design

4.1 Hierarchy Chart of Components

4.2 Sequence Diagrams and Descriptions

5. Low Level Design

5.1 Component C1

5.1.1 Class D1 of Component C1

5.1.1.1 Behavior Diagram/Description for Class D1 Of Component C1

5.2 Component C2

…

6. User Interface Design

6.1 User Interface Design Overview

6.2 User Interface Navigation Hierarchy

7. Requirements Traceability Matrix

8. Appendices

1. Implementation: Coding convention documents are sampled. Coding convention documents should follow the template given below (a detailed version of the template is given at the end of this document).

**Table of Contents for the Code Convention Document**

1. File names

2. File organization

3. Indentation

4. Comments

5. Declarations

6. Statements

7. White space

8. Naming conventions

9. Programming practices

1. **Assessment Procedure**: Student work will be assessed by at least two faculty members, one of whom must be the instructor of CMSC 4513 – Software Design and Development.

Two different types of assessment instruments are used: faculty-reviewed and student peer-reviewed questionnaires. Faculty-reviewed questionnaires are used for all three components. A student peer-reviewed questionnaire is used for the implementation component. The questionnaires are given at the end of this document.

The questionnaires are created based on three categories of competency as reflected in the student work. The three categories are completeness, relevancy, and quality:

* The completeness category examines whether the required sections given in the corresponding template (see Section 3) are present in student work.
* The relevancy category examines whether the content provided in the student work under each section under a certain template is relevant to the purpose stated for that section. In other words, this category checks whether students put contents under the right sections. For simplicity, a selected number of sections in the template are used in the assessment.
* The quality category examines the quality of student work. For simplicity, a selected number of sections in the template are used in the assessment.

The questionnaires contain a list of yes-no questions corresponding to the above three categories of competency. The percentage of yes answers to every question is evaluated using the rubric given in Section 5. Questions receiving Unsatisfactory or Developing under the rubric are used to identify opportunities for improvement based on the category of competency that the questions reflect.

1. **Assessment Metrics**: Consider the rubric given where is the percentage of yes answers to a particular question. If fewer than 25% of the requirement analysis documents receive a yes answer to a particular question, then the result is unsatisfactory. A close examination of the question and its corresponding competency category is needed to determine if some improvement can be made. To a lesser degree, if between 25% and 50% of the requirement analysis documents receive a yes answer to a particular question, the question and its competency category need to be investigated. If more than 50% of the requirement analysis documents receive a yes answer to a particular question, the rubric may be updated to reflect higher expectations or a conclusion can be drawn that the goal corresponding to that particular question has been achieved.

|  |  |  |  |
| --- | --- | --- | --- |
| **Unsatisfactory** | **Developing** | **Satisfactory** | **Excellent** |
|  |  |  |  |

1. **Course Cross Reference**: The courses that cover the topics relevant to this outcome are CMSC 3303 System Analysis and Design, CMSC 4283 Software Engineering, and CMSC 4513 Software Design and Development.

**Template of the Design Document**

0. Revision History

<*Show the time, version and who write the sections or subsections of this document.*>

1. Introduction

<*Provide an overview of the entire document, including the purpose, the scope and the intended audience of the document. Identify the product using any applicable names and/or version numbers.*>

1.1 System Overview

<*Provide a general description of the software product including its functionality and matters related to the overall system and its design.*>

1.2 Supporting Materials

<*Provide references for any other pertinent documents such as related and/or companion documents, prerequisite documents and documents which provide background and/or context for this document.*>

1.3 Definitions And Acronyms

<*Define any important terms, acronyms, or abbreviations.*>

2. Design Considerations

2.1 Assumptions

<*Describe any assumptions or dependencies regarding the software and its use. These may concern such issues as related software or hardware, operating systems, end-user characteristics, and possible and/or probable changes in functionality.*>

2.2 Constraints

<*Describe any global limitations or constraints that have a significant impact on the design of the system's software. Such constraints may be imposed by any of the following (the list is not exhaustive): hardware or software environment, end-user environment, availability or volatility of resources, standards compliance, interoperability requirements, interface/protocol requirements, data repository and distribution requirements, security requirements (or other such regulations), memory and other capacity limitations, performance requirements, network communications, and other requirements described in the requirements specification.*>

2.3 Goals And Guidelines

<*Describe any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software. Such goals might be: the KISS principle ("Keep it simple stupid!"), emphasis on speed versus memory use, or working, looking, or "feeling" like an existing product. For each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability.*>

3. Architecture

<*Describe any design decisions and/or strategies that affect the overall organization of the system and its higher-level structures. These strategies should provide insight into the key abstractions and mechanisms used in the system architecture. Describe the reasoning em-ployed for each decision and/or strategy (possibly referring to previously stated design goals and principles) and how any design goals or priorities were balanced or traded-off.*>

4. High Level Design

*<Provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components. Don't go into too much detail about the individual components themselves (there is a subsequent section for detailed component descriptions). The main purpose here is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together to provide the desired functionality.>*

4.1 Hierarchy Chart Of Components

<*Describe the major responsibilities that the software must undertake and the various roles that the system (or portions of the system) must play. Describe how the system was broken down into its components/subsystems (identifying each top-level component/subsystem and the roles/responsibilities assigned to it). Describe how the higher-level components collaborate with each other in order to achieve the required results. Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected).* >

4.2 Sequence Diagrams And Descriptions

<*Describe the interaction behavior between classes. The typical sequence diagram will depict some or all of the behavior described in a use case. Therefore, sequence diagrams serve as an intermediary between use cases and classes. As a result, classes can be traced back to the requirements.* >

5. Low Level Design

*<Provide more detailed discussion in a subsection/component of the component hierarchy section 4.1. If necessary, describe how the component was further divided into subcomponents, and the relationships and interactions between the subcomponents* (*similar to what was done for top-level components in Section 4.1*)*.>*

5.1 Component C1

<*Provide and describe a class hierarchy diagram that depicts the set of classes for Compo-nent C1*>

5.1.1 Class D1 Of Component C1

*<Provide a class diagram for Class D1 of Component C1. Provide a description of any relevant characteristics of Class D1.>*

5.1.1.1 Behavior Diagram/Description For Class D1 Of Component C1

<*For each class that exhibits behavior, provide one or more state or activity diagrams for that class as appropriate. If the class behavior is trivial or the class has no behavior, provide a description of the class, instead of a diagram.*>

6. User Interface Design

6.1 User Interface Design Overview

*<Provide a high-level description of the user interface for this software application. Describe any systems requirements (e.g., performance or usability) associated with all of the user interfaces. >*

6.2 User Interface Navigation Hierarchy

*<Provide and describe a diagram of the navigation hierarchy that illustrates how a user moves through the user interface.>*

7. Requirements Traceability Matrix

*<Provide reference to the location of the Requirements Traceability Matrix that indicates traceability from the software requirements document to the design elements documented.>*

8. Appendices

**Template of the Coding Convention**

1. File names

<*Define commonly used file suffixes and names. For example, if the programming language is Java, the common suffixes are .java and .class. Frequently used file names are README*, *GNUmakefile*, *etc.*>

1. File organization

<*A file consists of sections that should be separated by blank lines and an optional comment identifying each section. Files longer than 2000 lines are cumbersome and should be avoided. For example, Java source files have the following ordering:*

1. *Beginning comments*
2. *Package and Import statements*
3. *Class and interface declarations*>
4. Indentation

<*Use either spaces or tabs as indentation unit to format the source code. The coding convention should clearly define the makeup of indentation in terms of number of spaces or tabs.*>

1. Comments

<*Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program. If Java is the programming language, Doc comments should be defined. Doc comments are meant to describe the specification of the code, from an implementation-free perspective, to be read by developers who might not necessarily have the source code at hand.* >

1. Declarations

<*Define how many declarations can be put in one line (e.g., one declaration per line to make comments easier), how to initialize the variables (e.g., better at the time of declaration), the best place to put the declarations (e.g., at the beginning of the blocks), and class and/or interface declarations.*>

1. Statements

<*Define the formats of simple statements, compound statements, if-then-else statements, loop statements, try-catch statements, etc.*>

1. White space

<*Define how to use blank lines and blank spaces to improve readability.*>

1. Naming conventions

<*Naming conventions make programs more understandable by making them easier to read. Rules for naming packages, classes, interfaces, methods, variables, constants, etc., should be defined.*>

1. Programming practices

<*Define how to reasonably use “public” access, constants, variable assignments, etc.*)

**Assessment Instrument for Outcome (c)**

* **Outcome(c)**: An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
* **Evaluator:** Faculty
* **Student Work:** Design documents, coding convention documents, test plans and test cases completed by students in CMSC 4513 Software Design and Development.
* **Evaluation Questions:**

**Design:**

*Completeness:*

Does the design document contain the following sections?

* 1. “0 Revision history” Yes No
  2. “1.1 System overview” Yes No
  3. “1.2 Definitions and acronyms” Yes No
  4. “1.3 Supporting materials” Yes No
  5. “2.1 Assumptions” Yes No
  6. “2.2 Constraints” Yes No
  7. “6.1 User Interface Design Overview” Yes No
  8. “8 Appendices” Yes No

*Relevancy:*

Do the following sections contain the right contents as described?

* 1. Section 2.3 (Goals and Guidelines) describes any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software*. Such goals might be: the KISS principle ("Keep it simple stupid!"), emphasis on speed versus memory use, or working, looking, or "feeling" like an existing product. For each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability.* Yes No
  2. Section 2.4 (Development Environment) describes the environment, in which the software development will be performed. *It specifies OS, JDK version, other SDK version, etc.* Yes No
  3. Section 2.5 (System Environment) describes the environment, in which the system will be running. Yes No

*Quality:*

* 1. Does the document specify strategies that provide insights into the key abstractions and mechanisms used in the system architecture in Section 3? Yes No
  2. Does the document provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components in Section 4.1? Yes No
  3. Is the system properly broken down into components/subsystems in Section 4.1? Yes No
  4. Does the document correctly capture the interaction behavior between classes in Section 4.2? *This part covers some or all of the behavior described in a use case. Therefore, it serves as an intermediary between use cases and classes. As a result, classes can be traced back to the requirements.* Yes No
  5. For each component decomposed from Section 4.1, does the document have a section that describes the set of classes for that component? – Related to Section 5.x (x = 1, 2, 3, …) Yes No
  6. For each class within the component, does the document have a subsection that provides a description of the class? – Related to Section 5.x.y (y = 1, 2, 3, …)*.* Yes No
  7. If there are state transitions associated with a specific class, does the document contain a subsection that provides a detailed and correct description of the state transitions? – Related to Section 5.x.y.z (z = 1, 2, 3, …) Yes No
  8. Is the low level design in Section 5 consistent with the high level design in Section 4? All the components should be traced back to the high level design. Yes No
  9. By following the low level design, is it possible to perform the development without any ambiguity? Yes No
  10. Does the document provide a clear hierarchical description of user interface navigation in Section 6.2? Yes No
  11. Is the design document consistent with the requirement specification? *This can be checked through the requirement traceability matrix in Section* 7. Yes No

**Implementation:** Review student peer review questionnaire. Also complete the following questions.

1. File names

2. File organization

3. Indentation

4. Comments

5. Declarations

6. Statements

7. White space

8. Naming conventions

9. Programming practices

Does the coding convention contain the following sections?

* 1. “1. File names” Yes No
  2. “2. File organization” Yes No
  3. “3. Indentation” Yes No
  4. “4. Comments” Yes No
  5. “5. Declarations” Yes No
  6. “6. Statements” Yes No
  7. “7. White space” Yes No
  8. “Naming conventions” Yes No
  9. “Programming practices” ” Yes No

**Student Peer Review Questionnaire**

*Correctness:*

1. Does each source file have a meaningful name that follows “1. File names” of the coding convention?
2. Is each source file divided into different sections, such as beginning comment, import/include statements, and class/interface declarations, following “2. File organization” of the coding convention?
3. Does the indentation in each source file follow “3. Indentation” of the coding convention?
4. Does each source file follow “4. Comments” of the coding convention?
5. Does each source file follow “5. Declarations” of the coding convention?
6. Does each source file follow “6. Statements” of the coding convention?
7. Does each source file follow “7. White space” of the coding convention?
8. Does each source file follow “8. Naming conventions” of the coding convention?
9. Does each source file follow “9. Programming practices” of the coding convention?

*Quality:*

1. Does the beginning comment of each source file include the name of the author, the date of creation, and the purpose of the file?
2. If the source file is to be modified, does the beginning comment include the name of the modifier, the date of modification, and the reason for the modification?
3. Does the source file contain comments that are easy to read and understand?
4. Does the source file properly use blank lines and blank spaces so that the file is pleasing to eyes and easy to read?
5. Are the access to methods and member data correctly defined (in terms of public, private, protected, and/or package)
6. Does the source file properly use constants so that no magic numbers appear in the file?
7. Are the names in each source file meaningful?

Students enrolled in CMSC 4513 – Software Design and Development are evaluated by their peers and by the course instructor on their term project. The term project is examined based on the criteria: design, implementation and evaluation. Each criterion is scored for completeness, relevance, quality by the instructor. The instructor answers 53 questions and peers of the student being evaluated answer 16 questions. The percentage of correct answers is scored using the rubric: 0-24%: unsatisfactory; 25-49%: developing; 50-74%: satisfactory; 75-100%: excellent.

|  |  |  |  |
| --- | --- | --- | --- |
| **Unsatisfactory** | **Developing** | **Satisfactory** | **Excellent** |
|  |  |  |  |