SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 275-4 E100 – Software Engineering I
Fall 2017

Assignment 2 (10%) - Requirements and Initial Design[1]

Deliverables	Due	Time
1) Requirement Document	**Wed Oct 18, 2017**	**11:55PM**
2) Design document + Quality Assurance Plan	**Fri Oct 20, 2017**	**11:55PM**

PLEASE NOTE THE DUE DATE AND TIME.
LATE SUBMISSION WILL NOT BE ACCEPTED.
SUBMIT DOCUMENTS TO BOTH CANVAS AND TURITIN.
INCOMPLETE SUBMISSION WILL RESULT IN PENALTY.

Initial Requirements and Design

For this assignment your team needs to prepare three documents: 1) a requirements document, 2) a design document, and 3) a quality assurance plan.

1) Requirements Document

Your requirements document will be in the form of a user manual. It might seem strange to write a user manual before you have written any code for the system, but it is a good way to focus on the user-level requirements without getting bogged down by implementation details.

While we call this document the user manual, this is in fact a combination user manual (meant for end-users) and requirements document (meant for system designers). You must therefore write this in a language and format that is useful for both end-users and designers: don't make it too technical, nor too vague.

At this early point in the project it is more important to make this user manual useful for designers. **Make sure there is enough detail and information in it so that a system designer could ideally design and implement your system from the manual alone.**

This is a very important point: vague requirements are perhaps the single biggest problem that teams have in this course. Many teams report at the end that they wish they had spent more time nailing down these requirements, and making them more detailed.

[1] Adapted from Dr. Toby Donaldson's note in the description of this assignment.

Structure your user manual as follows:

1. **Introduction:** Explain what the system does and what is it useful for. You may be able to borrow text from your Project planning document.

2. **Intended Audience List:** the kinds of users for this system in some detail (certainly more detail than in the Project planning document from the previous assignment). Make clear what experience, expertise, and goals you assume they have.

3. **Features/Functional Requirements:** Catalog every significant feature of the system. This should be an exhaustive list. Hint: Give each of these features its own anchor or page, so that in the design document (described below) you can easily link to them.

   This is a major part of this assignment, and you need to take it seriously and do it well if you want to be sure you deliver a high-quality final project.

4. **Non-functional requirements:** Describe your system's non-functional requirements.

5. **Example Tutorials:** Provide 3 - 5 (or more, if necessary) examples of typical usage scenarios. These need to be highly detailed, step-by-step tutorial-like descriptions of how to use your system's major features. You should include screenshots of how your system will behave (users don't want to read through a lot of text!).

6. **Glossary:** List all technical terms and their precise definitions here. These will be terms about your application, not about programming or software engineering. For example, in a grade-keeper application, terms like "course", "grade", "mark", and "GPA" all need precise definitions.

See

- Software Project Survival Guide - http://www.construx.com/Thought_Leadership/Books/Survival_Guide/SPSG_Resources_by_Subject/

Consult

- PM: Ch. 8 Understanding requirements, then the following chapters as needed Ch. 9 Requirement Modeling: Scenario-based method, Ch. 10 Requirement Modeling: Class-based methods, Ch. 11 Requirement Modeling: Patterns and Web / Mobile Apps
- IS: Ch. 4 Requirements engineering

2) Design Document
Your design document outlines the high-level organization of your system, and it is what the programmers will refer to during implementation.

Your design document should have these sections:

1. **Guidelines:** List all technical guidelines (e.g. "Development will be done using the Eclipse IDE, version XX"), and any relevant ethical or legal issues.

2. **System Diagrams:** This should include 2 or 3 (or more, if needed) substantial diagrams showing how the major modules, classes, and functions of your system relate and share data. Important: these must be diagrams as described in our (System models) textbook; you must follow the proper rules and notations for the diagrams you use. It is not acceptable to make up your own kind of diagram! If you are using an object-oriented programming language, than one of your diagrams ought to be a class diagram.

   Keep in mind the purpose of this is not just to draw pretty diagrams, but to create blue prints that will later on help you implement your system. Do not forget to describe the diagrams.

3. **Data Requirements** This is a summary of all the I/O for your system, including exact definitions of file and database formats, what other systems (if any) yours interacts with, and how the user interacts with the system via the mouse/keyboard/wii-mote/etc.

4. **Feature Priority** You will be going through three implementation iterations during the course of this project --- version 1, version 2, and version 3. Label each feature listed in your user manual, as being delivered in version 1, 2, or 3. Hint: An ordered list of hyperlinks to the relevant features in the user manual might be a good way to organize this.

Remember that your Project planning document from the previous assignment should have a similar (but less detailed) list of what features is planned for each version; so that would be a good starting point.

It is important to note that version 1 (and, of course, each following version) be a usable system. Version 1 is a working prototype and you should implement all your promised version 1 features completely and correctly! Also, keep in mind that version 3 is the final "gold" version of your project, and anything not implemented by version 3 will not be implemented at all (at least during this course!).

Think hard about a good ordering for your features, taking into account things like their dependencies, usefulness, and complexity. Do not over-promise (too many features) or under-deliver (too few features).

Useful information

- IEEE Standard for Information Technology--Systems Design--Software Design Descriptions (1016-2009) http://standards.ieee.org/findstds/standard/1016-2009.html

Consult

- PM: Ch. 12 Design Concepts and Ch. 13 Architectural design
- IS: Ch. 5 System modeling and Ch. 6 Architectural design

3) Quality Assurance Plan
This document details everything your team will do to ensure the quality of your project. It must include at least the following information:

1. The software tool(s) you are using to perform automatic unit testing, and any other kinds of software testing. Do a survey to find out what tools are available. Tell us how you generate and manage your test cases.

2. The internal deadlines your group sets for unit/system testing of the code for release. Naturally, these deadlines must be well enough before the project submission date to give you time to do the testing and fix errors that might arise, but also late enough so that the development group has time to get all the features implemented.

3. Give the time, date, and location for when user acceptance testing of version 3 of your project (the final version) will be done. If you also want to do user testing for one or both of the earlier versions, then include the time, date, and location for those tests as well.
   Also tell us how this testing will be done: what will you be asking the users to do? Describe your the testing you plan to do in enough detail so that someone not in your group could run it for you.

4. Detail what kind of integration testing you expect to do, and how you expect to do it. It's wise to avoid "big bang" integration, where all the models are combined and tested at once. Instead, tell us how you plan to incrementally integrate and test the system. Be clear about how this testing differs from the unit testing you are also doing.

5. The software tool(s) you are using to measure the size and complexity of your project. You should be automatically counting various software metrics such as number of files, number of lines of codes, number of classes, etc.

   In addition to providing the raw measurement values, graph this measurement data (e.g. as line-graphs using an Excel spreadsheet) for each version of your project. Clearly label your data and graphs so that someone unfamiliar with the details of your project can understand what it is about without having to look at anything other than the graph itself.

6. List everything else you are doing to ensure the quality of your project. Give specific times and dates whenever possible.

Some examples:

- http://www.mhhe.com/engcs/compsci/pressman/graphics/Pressman5sepa/common/cs2/sqa.pdf
- http://acis.mit.edu/acis/sqap/sqap.r1.html
- http://qallme.fbk.eu/DEL/QALL-ME_D9.1_final.pdf
- http://energy.gov/cio/downloads/software-quality-assurance-plan-example
- IEEE Standard for Software Quality Assurance Plans (IEEE Std 730-1998), http://users.csc.calpoly.edu/~jdalbey/205/Resources/IEEE7301989.pdf
- IEEE Standard for Configuration Management in Systems and Software Engineering (828-2012) http://standards.ieee.org/findstds/standard/828-2012.html

Consult
- PM: Ch. 19 Quality Concepts and Ch. 21 Software Quality Assurance
- IS: Ch. 24 Quality management

Submission

You need to submit your documents in two locations:

1) Submit via canvas.sfu.ca

Submit the three documents (Requirement Document, Design document, and Quality Assurance Plan) in both formats: a) MS Word and b) PDF.
Filename convention: Group-XX-Requirements.docx, Group-XX-Design.docx, and Group-XX-QA.docx. Note that X is your group number, for Group one, XX = 01.

2) Turnitin

Plagiarism detection software (Turnitin) will be used to screen assignments on this course. This is being done to verify use of all material and sources in assignments is documented. See HW 1 for submission information.

Please note that marks will be deducted if you do not follow the above specifications (include but not limited to filename convention, file format, and citation style).

-END-

Appendix A: IEEE Standards you might want to consult.

Comprehensive List of Important IEEE Software Engineering Standards[2]		
IEEE Std 610.12-1990	IEEE Standard Glossary of Software Engineering Terminology	Note: This standard will soon be superseded by another ISO/IEC IEEE joint standard IEEE Std 24765. It is much more comprehensive than 610.19. You can get the final draft now.

[2] Source http://cs.hbg.psu.edu/cmpsc487/IEEEStds_List.htm

IEEE Std 730-2002	IEEE Standard for Software Quality Assurance Plans	This standard specifies the format and content of Software Quality Assurance plans.
IEEE Std 828-2005	IEEE Standard for Software Configuration Management Plans	This standard specifies the content of a Software Configuration Management plan along with requirements for specific activities.
IEEE Std 829-2008	IEEE Standard for Software Test Documentation	This standard describes the form and content of a basic set of documentation for planning, executing and reporting software testing.
IEEE Std 830-1998	IEEE Recommended Practice for Software Requirements Specifications	This document recommends the content and characteristics of a Software Requirements Specification. Sample outlines are provided.
IEEE Std 982.1-2005	IEEE Standard Dictionary of Measures to Produce Reliable Software	This standard provides a set of measures for evaluating the reliability of a software product and for obtaining early forecasts of the reliability of a product under development
IEEE Std 1008-1987 (Reaffirmed 2003)	IEEE Standard for Software Unit Testing	This standard describes a sound approach to software unit testing, and the concepts and assumptions on which it is based. It also provides guidance and resource information.
IEEE Std 1012-2004	IEEE Standard for Software Verification and Validation	This standard describes software verification and validation processes that are used to determine if software products of an activity meets the requirements of the activity and to determine if software satisfies the user's needs for the intended usage. The scope includes analysis, evaluation, review, inspection, assessment and testing of both products and processes.
IEEE Std 1016-2009	IEEE Recommended Practice for Software Design Descriptions	This document recommends content and organization of a Software Design Description.
IEEE Std 1028-2008	IEEE Standard for Software Reviews	This standard defines five types of software reviews and procedures for their execution. Review types include management reviews, technical reviews, inspections, walk-throughs and audits.
IEEE Std 1044-1993 (Reaffirmed 2002)	IEEE Standard Classification for Software Anomalies	This standard provides a uniform approach to the classification of anomalies found in software and its documentation. It includes helpful lists of anomaly classifications and related data.
IEEE Std 1058-1998	IEEE Standard for Software Project Management Plans	This standard describes the format and contents of a software project management plan.
IEEE Std 1061-1998	IEEE Standard for a Software Quality Metrics Methodology	This standard describes a methodology--spanning the entire life cycle--for establishing quality requirements and identifying, implementing, and validating the corresponding measures.
IEEE Std 1062-1998	IEEE Recommended Practice for Software Acquisition	This document recommends a set of useful practices that can be selected and applied during software acquisition. It is primarily suited to acquisitions that include development or modification rather than off-the-shelf purchase.

IEEE Std 1063-2001	IEEE Standard for Software User Documentation	This standard provides minimum requirements for the structure, content and format of user documentation--both printed and electronic.
IEEE Std 1074-2006	IEEE Standard for Developing Software Life Cycle Processes	This standard describes an approach for the definition of software life cycle processes.
IEEE Std 1175.1-2002	IEEE Guide for CASE Tool Interconnections - Classification and Description	This standard is the first of a planned series of standards on the integration of CASE tools into a productive software engineering environment. This part describes fundamental concepts and introduces the remaining parts.
IEEE Std 1175.2-2006	IEEE Recommended Practice for CASE Tool Interconnection: Characterization of Interconnections	This recommended practice describes interconnections that need to be understood and evaluated when buying, building, testing, or using computer-aided software engineering (CASE) tools.
IEEE Std 1175.3-2004	IEEE Standard for CASE Tool Interconnections-Reference Model for Specifying Software Behavior	The purpose of this standard is to specify a common set of modeling concepts based on those found in commercial CASE tools for describing the operational behavior of a software product. This standard establishes a uniform, integrated model of software concepts related to software functionality.
IEEE Std 1175.4-2008	EEE Standard for CASE Tool Interconnections--Reference Model for Specifying System Behavior	This standard specifies a Conceptual Metamodel for understanding and describing the causal behavior for a system. The purpose of this Conceptual Metamodel is to express causal behavior and compositions of causal behavior in a model that integrates all observable operational features of a system into one behavior specification. It provides the necessary semantic elements for describing general hardware/software systems, including hardware-only, software-only, or mixed system components, and it allows these different types of components to be treated in a consistent manner, providing a basis for representing a wide variety of systems.
IEEE Std 1220-2005 (ISO/IEC 26702)	IEEE Standard for the Application and Management of the Systems Engineering Process	This standard describes the systems engineering activities and process required throughout a system's life cycle to develop systems meeting customer needs, requirements and constraints.
IEEE Std 1228-1994	IEEE Standard for Software Safety Plans	This standard describes the minimum content of a plan for the software aspects of development, procurement, maintenance and retirement of a safety-critical system.
IEEE Std 1233-1998	IEEE Guide for Developing System Requirements Specifications	This document provides guidance on the development of a System Requirements Specification, covering the identification, organization, presentation, and modification of requirements. It also provides guidance on the characteristics and qualities of requirements.

IEEE Std 1320.1-1998	IEEE Standard for Functional Modeling Language— Syntax and Semantics for IDEF0	This standard defines the IDEF0 modeling language used to represent decisions, actions, and activities of an organization or system. IDEF0 may be used to define requirements in terms of functions to be performed by a desired system.
IEEE Std 1320.2-1998	IEEE Standard for Conceptual Modeling-- Language Syntax and Semantics for IDEF1X 97 (IDEFobject)	This standard defines two conceptual modeling languages, collectively called IDEF1X97 (IDEFObject). The language support the implementation of relational databases, object databases, and object models.
IEEE Std 1362-1998 (Reaffirmed 2007)	IEEE Guide for Information Technology-- System Definition-- Concept of Operations (ConOps) Document	This document provides guidance on the format and content of a Concept of Operations (ConOps) document, describing characteristics of a proposed system from the users' viewpoint.
IEEE Std 1462-1998 // ISO/IEC 14102:1995	IEEE Standard--Adoption of International Standard ISO/IEC 14102: 1995-- Information Technology--	Guideline for the Evaluation and Selection of CASE tools
IEEE Std 1465-1998 // ISO/IEC 12119	IEEE Standard Adoption of International Standard ISO/IEC 12119:1994(E), Information Technology-- Software packages-- Quality requirements and testing	This standard describes quality requirements specifically suitable for software packages and guidance on testing the package against those requirements.
IEEE Std 1471-2000 (ISO/IEC 42010)	IEEE Recommended Practice for Architectural Description of Software Intensive Systems	This document recommends a conceptual framework and content for the architectural description of software-intensive systems.
IEEE Std 1490-2003	IEEE Guide-- Adoption of PMI Standard-- A Guide to the Project Management Body of Knowledge	This document is the IEEE adoption of a Project Management Body of Knowledge defined by the Project Management Institute. It identifies and described generally accepted knowledge regarding project management.
IEEE Std 1517-1999	IEEE Standard for Information Technology— Software Life Cycle Processes— Reuse Processes	This standard provides life cycle processes for systematic software reuse. The processes are suitable for use with IEEE/EIA 12207.
ISO 9001:2000	Quality Management Systems-- Requirements	This standard specifies the requirements for an organizational quality management system aiming to provide products meeting requirements and enhance customer satisfaction.
ISO/IEC 9126-1:2001	Software Engineering-- Product Quality- -Part 1: Quality Model	This standard provides a model for software product quality covering internal quality, external quality, and quality in use. The model is in the form of a taxonomy of defined characteristics which software may exhibit.
IEEE/EIA 12207-2008	Systems and Software Engineering - Software Life Cycle Processes	This International Standard establishes a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry. It applies to the acquisition of systems and software products and services, to the supply, development, operation, maintenance, and disposal of software products and the software portion of a system, whether performed

		internally or externally to an organization.
IEEE/EIA 12207.1- 1996	Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology-- Software Life Cycle Processes--Life Cycle Data	This document provides guidance on recording data resulting from the life cycle processes of IEEE/EIA 12207.0.
IEEE Std 14143.1- 2000 // ISO/IEC 14143-1:1998	IEEE Adoption of ISO/IEC 14143-1:1998-- Information Technology— Software Measurement— Functional Size Measurement— Part 1: Definition of Concepts	This standard describes the fundamental concepts of a class of measures collectively known as functional size.
ISO/IEC TR 14471:1999	Information technology-- Software engineering--Guidelines for the adoption of CASE tools	This document provides guidance in establishing processes and activities that may be applied in the adoption of CASE technology.
ISO/IEC 14764:2006	Information Technology-- Software Maintenance	This standard elaborates on the maintenance process provided in ISO/IEC 12207. It provides guidance in implementing the requirements of that process.
ISO/IEC 15026:1998	Information Technology-- System and Software Integrity Levels	This International Standard introduces the concepts of software integrity levels and software integrity requirements. It defines the concepts associated with integrity levels, defines the processes for determining integrity levels and software integrity requirements, and places requirements on each process.
ISO/IEC TR 15271:1998	Information technology-- Guide for ISO/IEC 12207 (Software Life Cycle Processes)	This document is a guide to the use of ISO/IEC 12207.
ISO/IEC 15288:2008	Systems Engineering-- System Life Cycle Processes	This standard provides a framework of processes used across the entire life cycle of human-made systems.
ISO/IEC TR 15504 (9 parts) and Draft IS 15504 (5 parts)	Software Engineering-- Process Assessment	This technical report (now being revised as a standard) provides requirements on methods for performing process assessment as a basis for process improvement or capability determination.
ISO/IEC 15939:2002	Software Engineering-- Software Measurement Process	This standard provides a life cycle process for software measurement. The process is suitable for use with IEEE/EIA 12207.
ISO/IEC 16085:2006	IEEE Standard for Software Life Cycle Processes- Risk Management	This standard provides a life cycle process for software risk management. The process is suitable for use with IEEE/EIA
ISO/IEC 19761:2003	Software engineering-- COSMIC-FFP-- A functional size measurement method	This standard describes the COSMIC-FFP Functional Size Measurement Method, a functional size measurement method conforming to the requirements of ISO/IEC

		14143-1.
ISO/IEC 20926:2003	Software engineering - IFPUG 4.1 Unadjusted functional size measurement method - Counting practices manual	This standard describes IFPUG 4.1 Unadjusted Function Point Counting, a functional size measurement method conforming to the requirements of ISO/IEC 14143-1.
ISO/IEC 20968:2002	Software engineering--Mk II Function Point Analysis-- Counting Practices Manual	This standard describes Mk II Function Point Analysis, a functional size measurement method conforming to the requirements of ISO/IEC 14143-1.
ISO/IEC 23026 (IEEE Std 2001-2002)	IEEE Recommended Practice for the Internet--Web Site Engineering, Web Site Management and Web Site Life Cycle	This document recommends practices for engineering World Wide Web pages for use in Intranet and Extranet environments.
ISO/IEC 90003	Software and Systems Engineering-- Guidelines for the Application of ISO 9001:2000 to Computer Software	This standard provides guidance for organizations in the application of ISO 9001:2000 to the acquisition, supply, development, operation and maintenance of computer software.