CSC 7014 Practice Computer Programming
Fall 2016
Prof. Nguyen Thai

SOFTWARE PROJECT GUIDELINE

I. PROJECT PROPOSAL

The most important aspect of a software engineering proposal is that the proposal is about a problem that needs to be solved; it is not about programming. The proposal should clearly describe the motivation for the proposed work and the proposed solution, along with its expected business value. The proposal should not contain any detailed technical descriptions.

There are no unique or standardized templates for writing a software project proposal. However, there are general concepts that one should follow in writing a software-project proposal.

a. PROBLEM DIAGNOSIS

Describe the problem that you are planning to solve. Describe clearly what type of issues exists in current practice that you would like to address. Make sure the description of your concepts is not generic or too technically detailed.

Start with a brief description of the high-level context, then describe some specific issues that you are interested in solving, follow by providing more specific details about the sub-issues that your work will tackle.

b. PROPOSED SOLUTION

Describe how you are going to solve the diagnosed problems.

- What specific interventions or techniques will you introduce?
- What kind of metrics will you use to evaluate your success in solving the problems?
- How will you know that you have achieved your objective?

Discuss the business value of your proposed solution.

- What will your users gain from your proposed system that they are lacking now?
- Be as specific as possible in describing the envisioned benefits of your proposed solution.
- Provide example scenarios of how your proposed system will be used and explain how your solution is better than the currently in used system.

c. **PLAN OF WORK**

Provide step-by-step, details about what needs to be accomplished.

- Estimate how long will each step takes
- Software languages to be used (in our case, Python)
- Are there external resources required?
    - Reused software
    - Development tools
    - Hardware development environment
    - Facilities
- Describe your team.
    - What are the strengths and expertise of each team member?
    - Is your team size adequate to tackle the problem?
- Describe your software project test plan

Keep in mind that in a "real life" project, this is only an initial plan so that you can give your customer a preliminary estimate of costs and expected completion date. You will need to adjust these estimates as you progress, but hopefully not by much.

Assignment: Project proposal (2 pages)

II. SOFTWARE DEVELOPMENT LIFE CYCLE

Software is generally developed by a process known as Software Development Life Cycle (SDLC). I am modifying the formal SDLC to better fit our project development due to the size of our projects and time constraints.

1. **Gather and Analyze Program Requirements** – The developer must obtain information that identifies the program requirements and then document these requirements. For our project, we use the Project Proposal as our system requirements. Our software requirements are derived from the Project Proposal.

2. **Design the Program**

    - **Design the User Interface** – After the developer understands the program requirements, the next step is to design the user interface. The user interface provides the framework for processing that will occur within the program.

    - **Design the Program Processing Objects** (the core of the program) – The developer must determine what processing objects are required and then design each object to provide the capabilities of the program.

3. **Code the Program** – After a processing objects have been designed, the objects must be implemented in the program code. The languages of the program code is generally dictated by the project. In our project, the Python computer language is required.

4. **Test the Program** – Unit testing is performed on each object to verify whether each object is functioning as per requirements. Program testing is performed after all objects are coded and integrated to form a complete program.

5. **Demo the Program** – The developer showcases the program to the class.

6. **Document the Program** – The developer must document the program as it is being developed. After the program is coded and tested, the developer must document the instructions for using the program. In our class, that is the Project Report.

III. SOFTWARE DEVELOPMENT PROCESS

Software is developed in industry using standardized processes. Some of the most well-known processes are the Waterfall model and the Agile/Scrum model. The Waterfall model has been used for more than thirty years. It is a multiple phase software development processes, see Figure 1: Waterfall Model. It looks like a staircases with water falling from the upper stair steps to the lower stair steps. Notice that the steps are not aligned. The next phase starts before the previous phase finishes. There are also feedback from next phases to previous phases for error corrections.

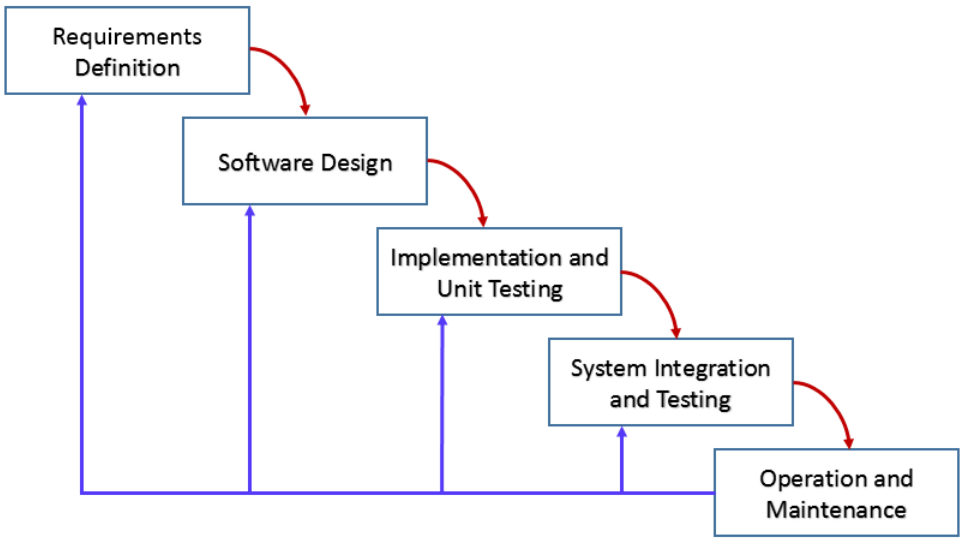

Figure 1: Waterfall Model

The Agile/Scrum model is the more recent process that is being used in industry. It is "supposed" to be an improved model from the Waterfall model. The Agile/Scrum processes are built on an iterative and incremental development with flexibility concepts.

I am again modifying these processes to support our classroom software development environment, see Figure 2: CSC 7014 Waterfall Model. We are following the Waterfall model with some adjustments. We do not implement all the phases of the model. The System Integration and Testing phase and the Operation and Maintenance phase of the Waterfall model have been replaced with the Project Demo phase and the Project Report phase, respectively in our model.

With the exception of the last phase, we align the project development process phases (i.e., we do not start the next phase until after we have completed the previous phase). We start the Project Report phase while the Project Demo phase is in progress. All data should be available for the report writing process during this timeframe. We do not have time for feedbacks; therefore, each team will modify their requirements, designs and code on their own along the development cycle as needed. We adapt part of the Agile model for our software coding phase. Software is developed and submitted to Blackboard in three increments.

- Increment 1: Infrastructure/framework software development
- Increment 2: Capability/feature software development
- Increment 3: Software Integration and test

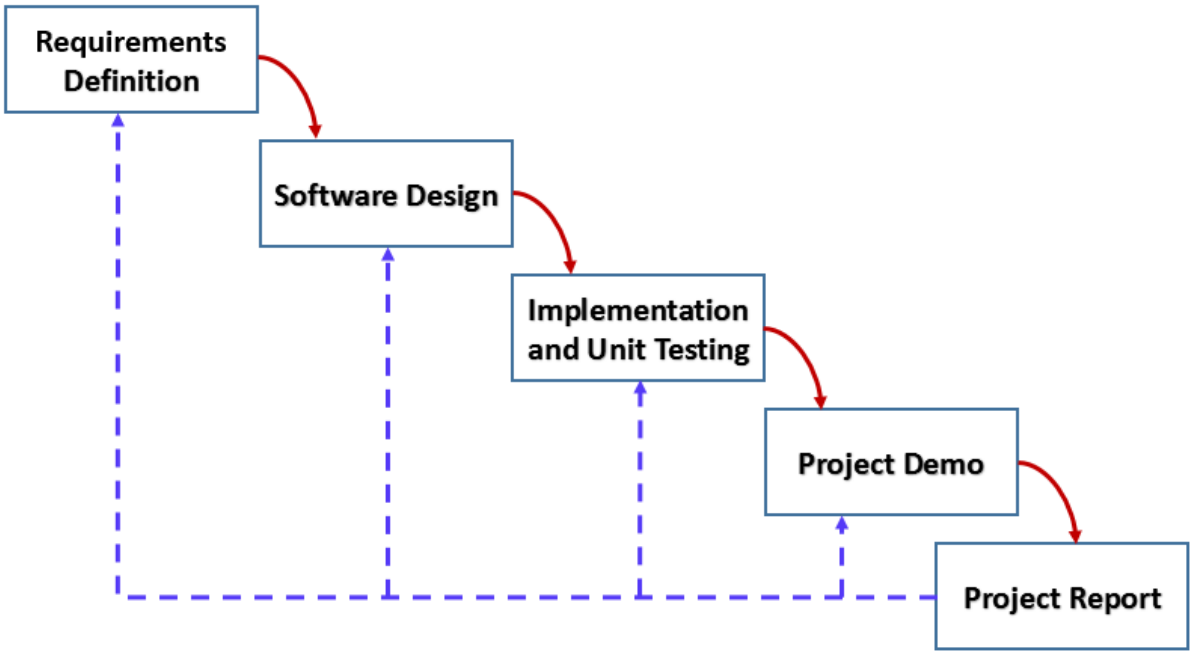

Figure 2: CSC 7014 Waterfall Model

IV. SOFTWARE REQUIREMENTS

Developing software requirements is hard; it takes skills, practice and patience. The most important about software requirements is that it is a contractual binding between the software provider and the customer.

Software requirements are generally captured in a set of documents called Software Requirements Specification (SRS). It is also called Functional Specification. We are developing a modified version of an SRS in this course. Your SRS must have the following items:

a) Cover Page

Include SRS title (Software Requirements Specification for the "project name"), student names, professor name, course number, course name and current date.

b) Table of Contents, Table of Figures (new page)

c) **Introduction** (new page)

Provide an overview of the entire SRS. Describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in next section (Specific Requirements). In a sense, this section tells the requirements in plain English for the consumption of the customer. The Specific Requirements section contains specifications written for the software developers.

d) Specific Requirements

This section contains all the software requirements at a level of detail sufficient to enable software designers to design a system to satisfy the requirements, and testers to test the requirements. These requirements should include a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:

- Requirements should be stated with accurate characteristics
- Requirements should be cross-referenced to earlier documents that related
- All requirements should be uniquely identifiable (numbering like 3.1.2.3)
- Careful attention should be given to organizing the requirements to maximize readability

Remember that requirements are not designs. Do not require specific software packages, unless the customer specifically requires them. Use proper terminology:

The system shall…… A required, must have feature

The system should… A desired feature, but may be deferred till later

The system may…… An optional, nice-to-have feature that may never make it to implementation.

Each requirement should be uniquely identified for traceability. Usually, they are numbered 3.1, 3.1.1, 3.1.2.1 etc. Each requirement should also be testable. Avoid imprecise statements like,

"The system shall be easy to use."

"The system shall be developed using good software engineering practice."

e) External Interfaces

This section contains a detailed description of all inputs into and outputs from the software system. It contains both content and format as follows:

- Name of item
- Description of purpose
- Source of input or destination of output
- Valid range, accuracy and/or tolerance
- Units of measure
- Timing
- Relationships to other inputs/outputs
- Screen formats/organization
- Window formats/organization
- Data formats
- Command formats
- End messages

f) Performance Requirements

This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements may include:

- The number of terminals to be supported
- The number of simultaneous users to be supported
- Amount and type of information to be handled

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks, and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

g) Design Constraints

Specify design constraints that can be imposed by other standards, such as hardware limitations.

h) Security

Specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:

- Utilize certain cryptographic techniques
- Keep specific log or history data sets
- Assign certain functions to different modules
- Restrict communications between some areas of the program
- Check data integrity for critical variables

i) Maintainability

Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirements for certain modularity, interfaces, complexity.

j) Portability

Specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems.

k) References

Assignment: Software Requirements Specification (5 – 10 pages)

V. SOFTWARE DESIGN

Software design is a very critical component of the software development processes. Software design is the process of transforming software requirements into physical realizable systems and subsystems. The software design process generally consists of two key tasks:

- Design the User Interface
- Design the Program Processing Objects

a. Design the User Interface

After the developer understands the program requirements, the next step is to design the user interface. Designing user interface is extremely important as it provides the framework for the external interaction to the system. In additions to ease of learn and use, the interface must consider data security and control issues, if applicable.

A user interface (UI) describes how users interact with the system, and can consists of:

- Humans

- Hardware
- Software

The key to a user interface design is to study the system's usability, and to create schemas for the implementation of the system such that it can provide user satisfaction, support for business functions, and system maintenance effectiveness. Effective user interfaces are the ones that users do not even notice the interfaces. These interfaces make sense because they do what the users expect them to do.

The methods used to design user interfaces vary from application to application, and from company to company. One of the commonly used technique that describes the interactions between the user and the system is the Use Case Definition or Use Case Diagram. The Use Case Definition specifies each of the sequences of actions by describing what the user will do and how the program will respond.

Figure 3: Hotel Room Selection Use Case Definition shows an example of a Use Case Definition for a simple event-driven program: Hotel Room Selection website.

USE CASE DEFINITION: Hotel Room Selection Website

1. User clicks Standard Room or Deluxe Room button.
2. Program displays a picture of the room chosen by user and enables the room selection button.
3. User clicks room button to view rooms if desired. Program displays the picture of the chosen room.
4. User clicks the Select Room button.
5. Program displays a room selection confirmation message, and disables both room buttons and the Select Room button. The Exit Window button becomes active.
6. User terminates the program by clicking the Exit Window button.

Figure 3: Hotel Room Selection Use Case Definition

Figure 4: Hotel Room Selection User Interface Design shows a technique that is commonly used in designing webpages. The wireframe diagram provides the preliminary look and feel for the to-be implemented user interface for the Hotel Room Selection website.

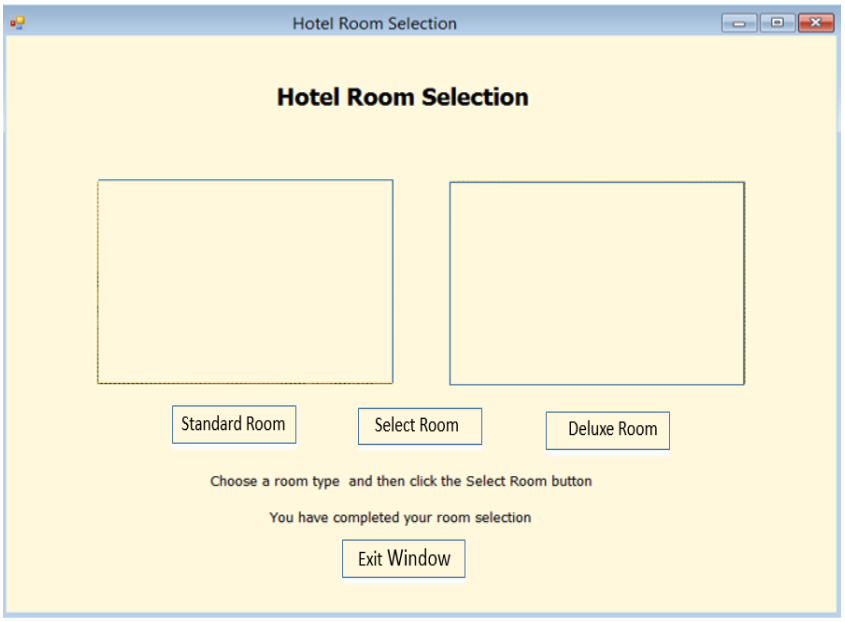

Figure 4: Hotel Room Selection User Interface Design

Figure 5: Hotel Room Selection User Interface Implementation shows one of the final implemented webpages. As you can see the user interface design diagram accounts for all the user interaction components of the webpage.

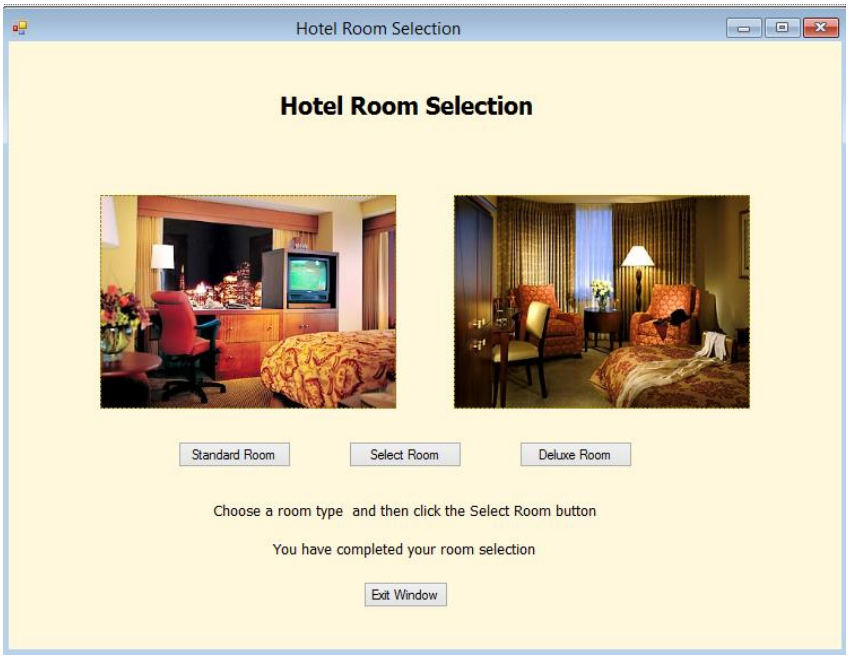

Figure 5: Hotel Room Selection User Interface Implementation

b. Design the Program Processing Objects

Once the user interface is designed, the developer must determine what processing objects are required and then design each object to provide the capabilities of the program. Various diagram types are commonly used for this design phase:

- Data flow diagrams
- Sequential diagrams
- Class diagrams
- etc.

Notice that in our Hotel Room Selection example, the user interface design is just a wireframe drawing. Although I did the design in Visual Basic, because it has a webpage design tools built-in; I could have created the user interface design using paper and pencil. There are many software design tools available, both commercial and freeware.

One of the very well-known software design tools is the Unified Modeling Language (UML). It is a widely used method of visualizing and documenting software system design. UML uses object-oriented design concepts, but it is programming language independent. UML provides various graphical tools, such as use case diagrams, sequence diagrams, class diagrams, etc.

Due to time constraint and the scope (size) of our projects, we are not using the full capabilities of UML. The following items are required for your software project design:

1. Use Case Definition
2. User interface design diagrams
3. UML class diagrams

Assignment: Software Design (Use Case Definition, user interface design diagrams, UML class diagrams)

VI. REFERENCES

1. *Project Guideline*, Thai, Nguyen, COMP 1050 Computer Science II, Spring 2016 Wentworth Institute of Technology
2. *Systems Analysis and Design*, 11th Edition, Tilley, Rosenblatt, Cengage
3. *IEEE 830-1998*
4. *MIL-STD-2167A*
5. *MIL-STD-498*
6. *Learning Python*, by Lutz, 5th Edition, O'Reilly
7. *Introduction to Programming Using Python*, by Liang, Pearson
8. Resources from the Internet