CIS 422 Software Methodologies - Winter 2019
A. Hornof - 2-15-2019 v2

Use this template (available on the course website) to develop and manage your Project 2.

An initial version of the entire document is due on Canvas on Thursday, February 21, at 10PM.

This is derived from

https://uocis.assembla.com/spaces/cis-f17-template/wiki

Modifications 5-23-2018

Removed from SRS:

6. Expected Subsets
7. Expected Changes

Table of Contents

I. Project Plan

Version 1, last updated by sfaulk at 2017-09-22

<Project Title>

Revision History:

Date	Author	Description

dd

1. Purpose and Audience

<Every technical document should clearly specify who the document is written for, and the purpose it should serve for its audience. This section describes the purpose and audience for the Project Plan.

- *The intended audience*: who the document is written for and what, if anything, is assumed about what they know
- *Purpose*: what purpose the intended audience is expected to use the document for (what will they get out of it?)
- Any additional information they need to know about how to use the document

>

2. Project Background

<Use this section to give an overview of any background someone needs to understand the project plan. Where possible, this should be done by providing links to existing documentation (e.g., other parts of the Wiki).>

3. Team Roles and Responsibilities

<This provides an overview of each team member's tasks and responsibilities. You must change the table to reflect the actual roles and responsibilities on your team. Each team member should play more than one role over the course of the project. Modify the table to fit your team organization.

Note: you must have a backup person for every key role, the backup person tracks what the Primary is doing and reviews any artifacts the primary produces. This puts the Backup in a good position to take over if the Primary cannot finish his/her tasks for any reason.
>

Role	**Team members taking on the role**	**Artifacts for which the role is responsible**

Requirements Analyst (1)	Primary: Backup:	ConOps, Software Requirements
Architect (1)	...	Software Design documents including architectural views and module interface specifications
Developer (>1)	...	Module implementation
Tester & Integrator (>1)	...	Test planning, Module tests, System generation and verification plan, test results report
Project Manager (1)		Communication, project plan, project measures, retrospective report
Configuration Control (build master)		Configuration management policy (coordinate policy with team members, ensure build works).

4. Risks and Risk Mitigation

<This section describes the perceived project risks, likelihood, consequences, and the planned mitigation strategy. The set of risks should be updated to reflect emerging risks or better understanding as the project evolves.>

Risk	Description	Mitigation
Risk 1		
Risk 2		
...		

5. Process

<Use this section to provide an overview of the process you will follow in terms of the activities, artifacts, roles, and relationships. Provide a justification for the chosen process in terms of

1. The project objectives and constraints.
2. How you have adapted the process to meet your specific developmental objectives.
3. Mitigation strategies to address project risks (e.g., running out of time.

Every organization will implement their own version of an agile process like Scrum, adding, removing, or modifying the activities, roles, or artifacts produced. For example, a student team has many other responsibilities and might choose too have a stand-up meeting only once every few days. This section should make clear how your are adapting the process to your needs.>

6. Mechanisms, methods, techniques

<Use this section to help the reader understand the software engineering methods, techniques, or tools that you are using.>

7. Detailed schedule and milestones

<Specifies the current sequence of tasks and due dates. Someone should be able to look at your schedule and determine who is working on what and when those artifacts are due.

This is probably most easily done using something like a Gantt chart and linking to it. You can create a simple Gantt chart in Excel from a template (an Excel template is provided on the course website). There are also free web services that support creating such charts.

You can also create another wiki page or a google calendar and link to it. The format is less important than that the information be kept up to date.

.>

8. Resources and References

<Who or what resources can answer questions about the product or process?>

Project Schedule and Milestones

Version 1, last updated by sfaulk at 2017-09-22

<Use this page to provide a detailed project schedule. At a minimum, the schedule should tell the reader:

1. A description of each task allocated to each team member. These should include ALL of the required course deliverables.
2. The team member(s) assigned to each task.
3. The calendar time scheduled for each task including the start date, end date, and expected level of effort (in hours of work).
4. Current task status: e.g., pending, in progress, completed, late.

>

Meeting Notes

Version 2, last updated by sfaulk at 2017-09-22

< As discussed in the first week's lectures, you must record the results of your team meetings. This will be easiest if you take notes during the meeting then post them for all the team members to see. This is where they should be posted.

Each meeting summary should include:

- **Date**: meeting day and time
- **Attendees:** list of attending team members
- **Agenda:** purpose of meeting, list of topics

- **Action items:** description of specific team member assignments as in the table below

Task and Deliverable	Assignee	Start Date	End Date	Estimated Time

Following the meeting, the Project Manager should put each Action Item/Deliverable on the schedule. Each team member should put their individual assignments into their blog.

II. Software Requirements Specification (SRS)

Version 4, last updated by sfaulk at 2018-01-08

Revision History:

Date	Author	Description

1. Introduction

1.1. Intended Audience and Purpose

<Describes the set of stakeholders and what each stakeholder is expected to use the document for. If some stakeholders are more important than others, describes the priorities.>

1.2. How to use the document

<Describes the document organization. This section should answer for the reader: “Where do I find particular information about X?” This section could be omitted if there is a useful table of contents.>

2. Concept of Operations (from the User’s Point of View)

<Use this section to give a detailed description of the system requirements from a user's point of view. The ConOps should be readable by any audience familiar with the application domain but not necessarily with software. The ConOps should make clear the context of the software and the capabilities the system will provide the user.>

2.1. System Context

<Specifies the system's operational context. This includes any software systems, hardware, web utilities, and so on that your system uses but you are not developing. The system context defines the boundaries of the system you are writing and shows its relationships to other systems in terms of the inputs expected from other systems and the outputs to other systems. For example, if your system uses the Google Maps API, this would specify what inputs your system gets from Google Maps and the outputs it sends to the mapping application. May include an illustration or *context diagram*. >

2.2. System capabilities (from the user’s point of view)

<This section should comprehensively summarize what the system can do from the user’s perspective, which could include a list of the use cases that are detailed in the next subsection as well as descriptive prose.>

2.3. Use Cases

<Scenarios showing how the user interacts with the system to accomplish specific tasks. See class materials on how to write good use cases. If specified as part of the requirements, should use a standard template and some justification for their completeness.>

2.2.1 Use case “Create an observation event”

2.2.2 ....

>

3. Behavioral Requirements

<Specification of the observable system behavior.>

3.1. System Inputs and Outputs

3.1.1 Inputs

<Each input should have an identifier and type specification (i.e., its value space). Likewise for system outputs.>

3.1.2 Outputs

3.2. Detailed Output Behavior

<A black box specification of the visible, required behavior of the system outputs as a function of the system inputs. Tables, functions, use cases or other methods of specification may be used.>

4. Developmental Quality Attributes

<Specification of additional required qualities of the system's construction such as maintainability>

5. Fundamental Assumptions

< Use this section to record assumption about the software or the relevant environment that you assume will not change over the life of the software. For example, if you plan to develop the software only to run on the current Android platform, this is a fundamental assumption. Software dependencies are fundamental assumptions.>

6. Definitions, Abbreviations, References

6.1 Definitions

Keyword	**Definitions**

6.2 Acronyms and abbreviations

Acronym or Abbreviation	**Definitions**

6.2. References

III. Software Design Specification

Version 1, last updated by sfaulk at 2017-09-22

1. Software Architecture

Version 2, last updated by sfaulk at 2017-09-22

<Use this page to specify the system's architectural structure and relationships. The goal of the architectural documentation is to capture and communicate all of the important design decisions concerning the way the system is decomposed into parts and the relationships between those parts. In general, this will require more than one architectural "view." Each view should capture one particular set of architectural structures (components, relationships and interfaces). These might include, for example the decomposition into independent work assignments (modules), the data flow, calls structure, thread structure, etc. as needed.

It is essential that the system architect provide sufficient documentation for the reader to understand:

- The overall design for the system including:
  - The set of components
  - Functionality allocated to each component
  - How the components work together to implement overall system functionality
- The rationale (reasons) for the major system design decision. I.e., tell the reader why this design was the best choice to solve the problem. Ideally, this should be done in terms of how the design addresses the functional and quality requirements.

2. Module Interface Specification

Notes on Interface Specification

Version 1, last updated by sfaulk at 2017-09-22

Purpose of this document

Interface specifications, including specification of application-level protocols across http connections, will be a crucial part of the distributed project undertaken in spring 2011 by students at Peking University and University of Oregon. This document offers some clarification about what an interface specification is, and some advice about how to construct a useful specification. Much of the advice is generic and would apply to any kind of interface specification, but we have drawn examples from the spring 2011 project and tried to focus on points particularly relevant to that project.

Objectives

An interface specification describes precisely how one part of a program interacts with another. Usually (but not always) the purpose of this interaction between program components is to allow one component (which we will call the *client*) to make use of a service provided by another component (which we will call the *server*). Note that the roles of client and server are relative to a particular service. A component can be a server with respect to one service and client with respect to another (as the Face Information Service is a server to the cell phone application and a client of faces.com).

A high quality interface specification tells the client developer everything he or she needs to know to use the service, and the service developer everything he or she needs to implement a service. If it is sufficiently clear and precise, the client developer and service developer can work independently, and integration will *just work* with no surprises. It is not easy to achieve this level of clarity, completeness, and precision. Here are some hints on how to get close.

Advice

Find the right abstraction

An interface presents some abstraction of a service to a client. It is essential to think carefully about this abstraction. It must tell the server and client developers everything they need to know, *and nothing more*. An interface that reveals too much information is called *over-specified*, and we say it limits *implementation freedom*. For example, the right abstraction for database access is probably a set of (*name, value*) pairs, or perhaps a set of rows, but it should certainly not constrain the developer of the database access service to use (or not use) an SQL database.

Describe every operation

An interface definition must describe every operation and every possible result. An operation could be a method or procedure call, access to a global variable or public field, or message transmission over a network interface. Results include not only the intended results of operations, but also possible exceptional results, such as errors that can occur. Be particularly careful to define "edge case" results. A good example of this is that the faces.com API notes explicitly that the array of face tags it returns may be an empty array if no faces were detected in a picture.

Factor the interface

Interface definitions are read by people, so we want them to be as simple, clear, and concise as possible. Often we can make them a good deal simpler through factoring different aspects of the interface. Factoring is a way of achieving *separation of concerns*, which also makes it easier to revise an interface later.

As an example, when an interface is implemented by communication over a network connection, it is generally best to separate the sequencing of messages, the content of each message, and the physical ("on the wire") format of each message. When sequences more complex than simple send-reply pairs are needed, they may be specified using state machines or message sequence charts (also known as sequence diagrams). Substantial parts of message format and the communication protocol may be defined by reference to existing standards, e.g., by choosing JSON or XML format over an HTTP channel.

Provide examples and test cases

A suitably precise and concise interface definition will be much easier to read and interpret if it is accompanied by examples, for both the expected cases and each exceptional case. Examples often uncover differences in understanding of something that everyone thought was precise and unambiguous. Well-designed examples can also be adapted as test cases for both the server and client side of the interface.

Negotiate, Review, Revise

An interface between major components of a system is often an interface between parts of the software development team. Details of the interface often entail decisions of who has responsibility for what (e.g., which errors are detected on the client side and which on the server side), and may have important consequences that are not obvious to developers who are responsible for only one side of the interface. Design of an interface should therefore be a negotiation of stakeholders on both sides of the interface. Each proposed version of the interface must be thoroughly reviewed by developers on both sides. You should expect several rounds of revision before you settle on an interface definition that is as good as it can be. This effort will be amply repaid in easing integration.

2011-03-25 by Michal Young

IV. Quality Assurance Plan

<These pages record plans and results from quality assurance activities including requirements validation, design reviews, module tests and system tests. the QA plan should describe how the set of reviews, tests, etc. work together to establish how well the system meets its quality goals. Use thia page to describe the team's QA goal and planned QA activities. For example:

- How will you establish that the requirements are complete?
- How will you determine if the design satisfies all the necessary functional and quality requirements?
- How will you check that the code meets its functional requirements?
- etc.

Reviews

Version 1, last updated by sfaulk at 2017-09-22

<Every artifact that cannot be executed should be reviewed for completeness and quality. The starting point for your reviews should be one or more internal reviews by team members. The natural choice for this job is the Backup for each artifact. To understand the qualities against which each artifact should be reviewed, a reasonable starting point is the Project Grading page. This page gives a brief overview of the qualities the instructor will look for in each artifact.>

Testing

Version 1, last updated by sfaulk at 2017-09-22

<
Testing activities must be planned in advance. Among other things, this helps ensure both developers and testers have the same understanding of expected software behavior at a sufficiently detailed level. It should go without saying that delivering untested software to the customer (instructor) is a recipe for disaster.

While we will address this more formally later in the quarter, the test plan for Project 1 should address the same issues informally. Specifically, you need to show me which inputs the system was tested against:

- What are the overall testing goals?
- What kinds of tests will be used to meet those goals?
- What tests were actually run?
    - Test inputs (include pointers to test sets)
    - Expected outputs
    - Results against each series of tests.

>

V. Software Documentation

Version 1, last updated by sfaulk at 2017-09-22

<Documentation for customers, users, etc. to be provided with the software. Exactly what should be provided depends on the type of application. For a standalone application, it should include an Installation Guide and a User's guide. If you are developing an application that must also be installed and run on a server, it should also include an Administrator's Guide describing the installation environment and how the software is administered.>

VI. Developer Logs

Version 1, last updated by sfaulk at 2017-09-22

<

Each team member is required to keep a long of their weekly activities and progress. The developer blog makes visible what you have been working on to the instructor as well as the team. It ensures that there is concrete evidence of your contributions to the project. While you may spend some time in activities that do not produce any tangible result, it is important that most of what is captured identifies specific, tangible products (e.g., a draft of a document, an initial interface design, a set of test cases, specific code components, etc).

Create separate page for each developer. Log entries should be kept short and to the point. The goal is to convey the essential information as efficiently as possible. Blog entries should objectively address the project and plans, not people. The entry for each week should contain at least the following:

	Date
Assigned Tasks & Progress	• Tasks you have agreed to undertake and are working on. This should correspond directly to the tasks agreed to in team meeting and appearing in the project plan/ schedule. Show progress as tasks are completed.
Issues	• Problems encountered that need to be resolved. This would include things like incompletely defined requirements, waiting on output from another team member, technical problems, etc. Particularly, this should identify any problems that are holding up completion of assigned tasks.
Plans	• Plans for the next week particularly including any necessary revisions to the planned work or schedule.
Comments	• Any useful insights or observations about the project or development process.

>