Software Engineering Project
055:183/22C:183
Fall 2009

Team 1: Project Plan

for

'NoteShare'

Team1
Nathan Denklau
Nathan Fritze
Jonathan Hall
Joe Trapani
Satpreet Singh

Table of Contents

would anyone not familiar with your project be able to figure out what you intend to create based upon this description?

1 Introduction

1.1 Project Scope

The Scope statement should provide the business justification for the project, deliverables, objectives and description of the result of the project. If everyone in the project is clear about the Scope Statement, scope creeps are easier to control.

Does Yours achieve this objective?

NoteShare is a social-networking application extension which provides users with the ability to enter into a community with the overarching goal of exposing users to alternative perspectives and trains of thought. It is believed that through this effort, both a deeper and broader understanding of course material is possible.

Since everyone interacts with individuals from varying backgrounds (from slight differences to extremes), a broadened, more encompassing perspective should provide better understanding and cooperation between all individuals.

about what?

1.2 Glossary

Facebook	A social networking site which will provide NoteShare with indirect access to its targeted audience
SVN	A software repository system which provides revision history and source code versioning
COCOMO	Costructive Cost Modeling is a system used to estimate development time and effort

1.3 Version Number(s)

1.3.1 Document Modifications

Addressed later in the configuration management section, project documents will be maintained on Google Documents. A major version number will be granted upon delivery of the revised document, and future major numbers will be based off post-submission group revisions and consensus.

1.3.2 Source Modifications

With the source stored on Google Code's SVN, revision numbers themselves will be automatically tracked. Software release numbers will be based off of scheduled delivery of iterations. More detail about the software process is included below in the development procedures section.

1.4 System Overview

?

1.4.1 Purpose

NoteShare will be designed to expose students to alternative mindsets by connecting through language. Overall, the application will provide increased value to Facebook as well as contribute to the acedemic community.

1.4.2 General Description

NoteShare will be primarily a social-networking plugin. Each specific component is addressed in the initial requirements analysis, but in brief, NoteShare will provide the following capabilities (in no particular order):

- Note Upload and Sharing
- Note Review
- Note Categorization by Course and User-Defined Categories
- Community Billboard
- Privitized Study Rooms
- Multi-School Integration
- Private User Homepages

1.4.3 Past History

The team has not had direct experience developing under the Facebook API. However, team members do have programming experience in other languages (primarily Java).

1.4.4 Project Stakeholders

University Students	These will be the primary intended users of the system. They will largely contribute to the requirements and dictate design decisions. Emphasis will be placed on delivering a system that is convenient and easy to use for this demographic.
Development Team	The NoteShare development team has a direct and obvious stake in the design and completion of the project. Other development teams also could have a stake in the design of NoteShare if it takes off. Other developers will become necessary for integrating other Universities. Support staff could also become necessary if user count is sufficient (i.e. larger than one server at >10Mbps can handle)
Professors / Acedemic Staff	The professional acedemic community will probably be highly interested the content of the application. Specifically, this platform needs to be design around the concept of preventing cheating. Further requirements gathering will probably be necessary to develop a common understanding of what will (and will not) be allowed on the applicaiton for sharing.
Facebook	This company will have a particular interest in NoteShare since the application will be developed on their platform. Their actual needs and requirements still remain undefined.
Hosters	The hosters of the internet service will need sufficient technical resources to maintain the servers as well as provide the hosting service.

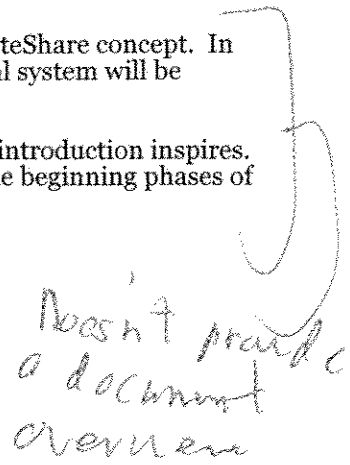
1.4.5 Current and Planned Operating Sites

At the moment, the domain name for the system operation has yet to be determined. The actual system itself is planned to interact with Facebook as a plugin application, but all submitted content and application execution will reside on a separate server unit.

1.5 Document Overview

The ultimate goal of this document is to make an initial effort mapping out the NoteShare concept. In future documents, more details and specifics will be provided about how the actual system will be implemented.

The rest of this document should address any unanswered questions that the this introduction inspires. Details to some answers will be limited due to the fact that this project is still in the beginning phases of development.



2 Initial Requirements

2.1 Project Objectives

As a social networking based application, NoteShare has one main objective: **EXPOSURE**
From that main objective, these are the following sub-objectives:

- Provide a comfortable ground wherein students can experience differences
- Provide a balance between individual privacy and public community
- Alleviate the tension of language by providing indirect encounters between individuals
- Improve note taking ability and style through community feedback (exposure)
- Remove personal biases in teaching style by mass-cancellation (hyper-exposure)

2.2 Major Functional Requirements

2.2.1 Note Upload Capability

2.2.1.1 Upload procedure is convenient.
Since note uploading is a key functionality of NoteShare, the process for submitting notes has to be convenient, intuitive, and quick. While this highly subjective metrics will be somewhat difficult to track in the final product, customer reviews could potentially highlight obstacles of the procedure. Given the social networking platform driving NoteShare, these customer reviews should be plentiful and immediate.

2.2.1.2 Upload procedure is non-intrusive.
A non-intrusive uploading procedure will allow relatively any sort of note content. Direct homework answers, test answers, and other obvious forms of cheating will need to be disallowed, addressed in later sections. However, to accomplished the delicate nature of exposure, no arbitrary enforcement of note style can occur, if at the very least, to protect the community of language itself.

2.2.1.3 Upload procedure facilitates easy categorization of submitted notes.
This point, if implemented poorly, could highly conflict with requirement 1.2. At no point should the categorization of notes become intrusive and limiting. User generated tags could more than likely provide flexible categorization, while still allowing for practical programming implementation. Impressing upon students that their classes are not completely and totally interrelated would be completely counter to the main objectives of the NoteShare application (in addition to self-negating).

2.2.2 Note Review Capability

2.2.2.1 Note Review should provide democratic rating system.
For useful feedback and exposure, any note submitter should be made aware of insightful how his or her peers find submitted notes. This could be accomplished by a rating system of pluses and minuses similar to the commenting systems on www.digg.com and www.youtube.com.

2.2.2.2 Note Review should prevent belligerent or nonconstructive comments.
Feedback (i.e. comments) that represent personal attacks or otherwise not useful "babbling" (technical term meaning empty or ignoble speech) should be prevented and filtered from community feedback. This will definitely be a difficult task as this can be directly interpreted as limiting language, but hopefully, community encouragement will mainly provide this functionality. Otherwise, more respected community members could obtain moderator status over comments.

2.2.2.3 Note Review should prohibit unauthorized content.
Who authorizes content is the most glaring obstacle to this requirement. However, allowing the posting of homework solutions or test answers will degrade NoteShare into another meaningless Facebook application. "Solutions" in this sense are in direct contrast to the exposure-objective of NoteShare by limiting interpretation to a particular answer. Addressed later in categorization, differentiation will be necessary between subject matter that does "in fact" have a single solution.

2.2.2.4 Note Review should embrace the significances of silence.

what do you mean by "note categorization"

Not speaking is just as important as speaking. The note review system should take notice of notes that are not commented on or not viewed for an extended period of time and somehow remove them.

2.2.3 Note Categorization

of what?

2.2.3.1 Categorization shall be based on course numbers and user opinions.
There is a simple fact of matter is that courses are already organized by most (if not all) universities. It would be rather silly to not utilize a categorization system already in place. However, this categorization is insufficient since relations between courses exist that aren't exposed by department number, course number, etc. User driven categorization could provide this additional categorization to help further expose similarity between classes / note-sets.

2.2.3.2 Categorization should be flexible and user driven.
As mentioned in requirement 3.1, user generated tags seem the most enabling solution for categorization. Strict limitation of notes to particular classes would impede the notion of exposure. User generating tags would allow for a variety of categorizations and more importantly, cross-class relations.

2.2.3.2 Categorization should be used for referencing similar material.
Categorizing uploaded notes without the ability to search would nearly defeat the purpose of the categorization in the first place.

2.2.4 Community Billboard

2.2.4.1 Community billboard should provide...community.
The community billboard component should be designed to provide a common area for students to discuss ideas (most likely through threads or forums), coordinate study sessions, and possibly see upcoming significant course related dates (e.g. tests, homework due dates).

2.2.5 Privatized Study Rooms

what is a privitized study room.

2.2.2.5 Study rooms should be invitation based.
In addition to providing community, private space should be allocated as well for more intensive or detailed discussions of content (most likely relating to upcoming course dates like tests).

2.2.6 Multi-School Integration

2.2.6.1 Students should be able to find similar content from other universities.
Both universities and professors tend to have a teaching style or mindset behind the approach to their course content as well as content delivery. Enabling students to experience approaches at other universities would provide greater exposure and understanding of course material.

2.2.7 Private User Homepages

2.2.7.1 Users should have homepages with personalized content.
What a user would like to have on his or her homepage will be a large component of the requirements gathering and analysis. However, at this point in the process, foreseeable content would include:

- A calendar
- Students with courses in common
- Recent ratings of notes
- Recently submitted class notes
- Recently submitted notes under "watched" tags

2.3 Major Nonfunctional Requirements

2.3.1 Not Limiting Language

The basic notion behind NoteShare is exposure. Limiting language is, to a degree, the opposite of that. However some limitations can actually provide more exposure and more meaningful interchanges. This greatly complicates matters, and NoteShare will have to maintain a delicate balance between limits and non-limits.

2.4 Other Constraints and Requirements

The only remaining constraint or requirement is customer satisfaction. Given the nature of a social-networking application, customer satisfaction should become almost immediately apparent through lack of use. However, some sort of customer feedback (probably billboards or e-mail) should be established to ensure a successful product.

Many of the things in your functional requirements are actually non-functional

Isn't integration w/ Facebook a MAJOR constraint.

3 Project Processes

This section specifies the processes that will be followed to facilitate the completion of the project. This includes the **Development Process**, **Team Organization**, **Quality Control**, **Tools,** and, **Configuration Management**.

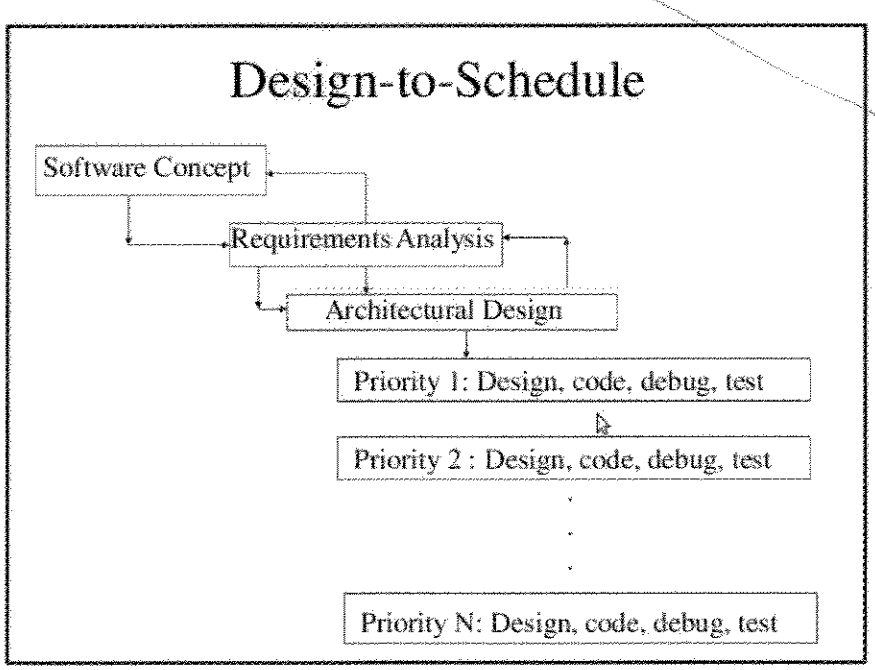
3.1 Develoment Process

The development process that the team utilizes should allow the team to mitigate and control the influence of the risks perceived in section 5.2.5 . Some of this project's largest challenges are inherent in a class project, namely a definitive end of schedule, and fixed resources. Domain knowledge is also another concern in developing the project.

3.1.1 Process Model

The plan that we intend to follow addresses and mitigates these concerns through a staged design with prioritized requirements as well as the use of prototyping to evaluate ideas and improve domain knowledge. This would be a design-to-schedule process model with added prototyping features. The design-to-schedule model address the limitations of finite resources on a fixed deadline by prioritizing iteration requirements and acknowledging that low priority requirements may not be implemented in the final iteration. Furthermore, we shall utilize prototyping to evaluate and pursue team decisions and experiment with the problem domain. This model will allow a more focused development and may improve morale; while accepting that the end-of-term deadline is final, we can still provide a meaningful product.

looks ok but make sure you fully understand all requirements of first



not sure how prototyping fits with your process model? when/where do you require to do prototyping?

3.1.2 Process Model Activities

The process activities can be split into two types: serial activities and parallel activities. Serial activities are those which require the completion of earlier products to continue, constituting a majority of the design-to-schedule process. In contrast, parallel activities are those designed to research and prototype ideas, as well as to reach consensus; they contribute to the overall serial process. Examples of parallel

activities would be researching social network utilization and finding hosts; however all deliverable will be the products of the serial activities.

Outlined below are the processes of the serial component of our design-to-schedule model:

1. **Planning**
   - *Entrance Criteria:* Scope Proposal
   - *Exit Criteria:* Project plan is approved.
   - *Products:* Scope Document, Project Plan, Schedule

2. **Requirements Analysis**
   - *Entrance Criteria:* Scope and domain is well understood, Project plan in place
   - *Exit Criteria:* Approved, fully specified, prioritized, and stable requirements
   - *Products:* Needs, Requirements, Revised Schedule

3. **Architectural Design**
   - *Entrance Criteria:* Requirements
   - *Exit Criteria:* Approved architecture
   - *Products:* Architectural Plan, Revised Schedule, Revised Requirements

4. **Prioritized Development Stages:** This activity encapsulates the development of the staged iterations that are outlined in section 4.4. Each stage follows the same activities, building on the last, in a loop until all requirements are met or until the scheduled end-of-term deadline is met. These stages can be further broken up into a design phase, a coding phase and a testing/debug phase. Stages will proceed from the prioritization of requirements.
   - *Entrance Criteria:* Architecture
   - *Exit Criteria:* Baseline met, all tests pass. OR schedule terminated
   - *Products:* Stage Designs, Stage Code, Stage Tests, Revised Schedule, Revised Architecture

Don't see any prototyping here.

3.1.3 Process Benefits and Short Comings

In choosing this life-cycle model, identified risks were addressed with perceived benefits of the process. This section bullets these benefits and identifies some short comings of choosing this life-cycle process model.

Benefits:
- Staged delivery allows users to use product as its developed
- Prioritized requirements allows development team to focus on important features
- Realistic schedule constraints improve morale (no crunch time)
- Prototyping can add valuable insight into problem domain and technology choices

Short Comings:
- Process requires stable requirement priorities
- Stakeholders priorities may differ ← Need to resolve this up front.
- Stakeholders may fear an incomplete project

3.2 Functional Processes

This section defines the processes by which the **Process Model** will be achieved. Foremost in this regard is how we work as a team and how the team will get things done. From **Quality Control** to **Tools**:

3.2.1 Team Roles and Responsibilities

We do not wish to assign fixed roles to our team-members beforehand and hope that over the course of the project, we are all able to experience the various possible roles on offer. We expect to all be analysts, designers, coders and testers over the course of the project. The only exception to this is the role of Team Co-ordinator (Satpreet), who is responsible for keeping the instructor updated with the progress of the team. For all roles that exist, it is assumed that people will either take the initiate volunteer for them by themselves, or step in to fill up roles that haven't been taken up.

ok but make sure that roles/responsibilities of team members are well understood at all times.

3.2.2 Team Meetings

Meetings are held both online and face-to-face. We will try to meet at least once a week in the first half of the semester, and then increase frequency of meeting as required for the second half of the semester. Responsibility for taking meeting notes is rotated amongst the team members. Decisions are taken on a majority vote.

3.2.3 Quality Control and Testing Overview

Quality is ensured by team review. Every product has an initial contributor who puts forth a initial draft proposal. A member of the team must then review this document and make in-line comments on it and suggest revision. At least one assigned team member, not the first contributor, will review any document. Any comments made by a predetermined deadline will be collected and placed in a separate section by the initial contributor where they can be addressed per the documentation guidelines. This section should provide an indication of revisions that were made from previous draft.

Code takes a slightly different approach toward, QA. Here the emphasis is on testing and not review. Test suits will be packaged by the same developer as the code which address the appropriate requirements; the developer is responsible for maintaining a working product, and may not commit un-tested and un-working code. The developer will be responsible for unit testing as well as testing against previous integration test suits.

3.2.4 Release Process

For each document deliverable, we will have have

- *Initiator(s):* a person or group of persons, who produce the first draft of the deliverable. Brainstorming is usually done collectively, actual production is usually done independently.
- *Reviewer(s):* a person or group of persons, who collectively review the first draft of the deliverable produced by the Initiator(s). The Reviewer(s) for a part of the deliverable must be a different person from the Initiator(s) for the part of the deliverable.
- *Submitter(s):* a person or group of persons, who ensure that the document reaches the instructor by the deadline. As submission is through usually email, all the remaining team members are on the CC on the submission email.

For other deliverables, only a Submitter is defined. Everyone takes up coding/testing/design responsibilities in parallel.

3.2.5 Tools and Technologies

These are the tools that have been decided on at present to enhance the project experience and implement the development process.

- *Google Docs:* Collaborative document editor, document version control
- *Google Code:* SVN repository, version control
- *Facebook API*: Social network integration (under review)
- *OpenSocial API:* Social network integration (under review)

3.2.6 Configuration Management

The configuration management principles outlined in this section will apply to all products produced by or team. Our aim is to utilize the tools in the above section to streamline configuration management; however, we acknowledge that coordinating baselined versions will be a continuous team activity. All documentation, code, and tests produced by the team will be automatically versioned by the tools. At such time as a scheduled milestone of the schedule is reached, requirements met or all reviews addressed, that product will be forked. The fork will receive a new version number and the original will become the baseline. The baseline shall remain unmodified, while work continues on the fork to implement the next stage or in the case of documents they need further revision.

4 Schedule

4.1 Estimation of the size of the product:

We have approximately ten working weeks on the project and paired with the fact the traditional methods like function point analysis are extremely difficult to compute given that none of the team members have moderate experience in this type of project and process. The size of our project will be determined in screen's of the interface and lines of code (LOC).

Objective:	**Objective Details:**	**Size of Objective:**
User course screens	(1) Join course screen (1) Browse schools screen (1) Browse courses screen (1) View course members user's screen	33% *Average Approximation:* 2000 lines of code (2 KLOC)
Administration management screens	(1) User statistics screen (1) Course management screen (add/drop users)	17% *Average Approximation:* 500 lines of code (.5 KLOC)
Adding notes screens	(1) Adding notes screen (1) Viewing notes screen (1) Browsing notes screen	25% *Average Approximation:* 500 lines of code (.5 KLOC)
User bulletin board screens	(1) Add a topic screen (1) Browse topics screen (1) Reply to topic screen	25% *Average Approximation:* 500 lines of code (.5 KLOC)

this is a good effort

The formula to approximate the size is a very basic trivial approximation by the number of objective details divided by the total number of details of the entire project. This method is by no means perfect, and is very possible to change once the team becomes more familiar with the project and our platform of choice.

As another reference, we also ran our projections through the COCOMO model. This model was developed by Barry Boeham in 1981 and it uses Lines of Code to calculate the size and effort of a large-scale software project. In our terms, a line of code consists of a line within the source code that has a character on it that is not a comment. Given the 3 years of professional Delphi experience that one team member has (Delphi is a RAD environment), he has drawn the rough estimates of the lines of code from previous work (PHP and win32 programming). Using lines of code is beneficial at this point, because it is difficult to generate the specific knowledge that is required for a complete function point analysis. We have chosen to use the organic mode project because our project is rather small and simple with flexible design decisions. We felt that the semi-detached mode and embedded projects do not accurately match our description. According to COCOMO, an organic project's development time will be an average of 2.5 * 8.94 (look below to gather this information) ^ .38 = 5.75 months. Given that our team consists of five members who can work on different parts of the project simultaneously, we are approximating that we could divide the time by 2.5, meaning we would always have two different sets of development subteams, and for about half of the time frame we might even have a third subteam. This is required in order to bring our development time to a level where our deadline is attainable.

4.2 Estimation of the effort:

Using the basic COCOMO model the estimated effort involved would be an average of 2.4 * 3.5 ^ 1.05 = 8.94 / 4 (active teammates) = 2.24 months. Although we have five teammates in our team, we would count one person out of the equation. This would be to account for an individual who may not motivated,

or a person whose skill set is inadequate. The teammates divide the total effort because of the collaborative, fast reacting group dynamics that we have set up for this project. From the project introductions, we all are around the same skill set level, with around 5 - 7 hours available to work on this project per week. It is likely that this is optimistic schedule will fall short, so swift recalibration is essential to the project's schedule.

Refer to the chart below for three scenarios of how our project size may effect our effort and development time:

Lines of Code	COCOMO Effort Calculation	COCOMO Development Time Calculation
2500	6.28 Months / 4 = 1.57 Person-Months	4.82 Months / 2.5 = 1.93 Months
3500	8.94 Months / 4 = 2.24 Person-Months	5.75 Months / 2.5 = 2.30 Months
4500	11.64 Months / 4 = 2.91 Person-Months	6.35 Months / 2.5 = 2.54 Months

4.3 Estimation of the Schedule:

Using McConnell's magic formula, the project will take approximately (3 * 1.93 ^ 1/3) 3.74 months, (3 * 2.30 ^ 1/3) 3.96 months, or (3 * 3.8 ^ 1/3) 9.36 months. With this in mind ranking priority becomes an important concept, since we can not extend our ten week deadline. Areas that will affect our schedule are:

4.3.1 Talent of the staff

Overall, the talent pool is relatively close with all of the group members. For this type of project, our experience is minimal, due to new plateform/API to learn for the entire team. However, some team members have some PHP experience, which will be useful. Two of the five team members have professional development experience.

4.3.2 Management

Since our project does not have an actual customer or manager, we do not have to consider that management would divert our time away from activities unrelated to technical development work.

4.3.3 Tool support

Everyone in the group should have access to the same set of open source software for application development. One team member is a communitor student, so that would enforce communication methods between the team members to be a vital necessity.

4.3.4 Methodology

We must ensure that the most time-efficient development methods and tools are to be used on our project. Using Facebook poses as a serious risk, but it does provide us with a strong user base, and a handy user-management system that we can access through the API. There are a plethora of tools for Facebook development, so hopefully, we will find something to fit our concept of a sound, solid methodology.

4.3.5 Compression

Our schedule has been compressed as much as possible, given that we only have a ten week span to completed the given iterations (posted below).

All of these factors will have an impact on our schedule, and will undoubtedly lead us to a nominal schedule from an efficient schedule. Because of this, prioritization is a critical part of our analysis on what will be completed within the given time frame. The chart below displays the order of importance for the major objectives of this project.

Objective:	**Objective Details:**	**Difficulty of Objective-Order of Importance:**
User course screens	(1) Join course screen (1) Browse schools screen (1) Browse courses screen (1) View course members user's screen	1 (Most Important)
Administration management screens	(1) User statistics screen (1) Course management screen (add/drop users)	4
Adding notes screens	(1) Adding notes screen (1) Viewing notes screen (1) Browsing notes screen	2
User bulletin board screens	(1) Add a topic screen (1) Browse topics screen (1) Reply to topic screen	3

4.4 Timeline

09/16/2009 Project Plan due.
Brief Details: This first milestone requires the team to submit a project plan to the professor/class.

09/25/2009 Stakeholders Needs due.
Brief Details: The team is required to submit the stakeholders needs document to the professor/class.

10/02/2009 Requirements Specification due.
Brief Details: Gather the requirements of the project, while trying to be as straight-forward and well defined as possible.

10/12/2009 System Architecture Specification due.
Brief Details: Construct the system architecture for the project.

09/21/2009 Custom Iteration 1-Setup server/database connection.
Brief Details: Set up a test application to test database connections and server capabilities.

10/23/2009 Custom Iteration 2-Add identity submission page.
Brief Details: This iteration will deal with submitting input to the server for typed in notes, the bulletin board, tagging, etc... Upon completion of this iteration, supporting development documents and the user manual will need to be created.

11/06/2009 Custom Iteration 3-Create school/class/semester/session structure.
Brief Details: This iteration will have the largest amount of time devoted to it, considering it contains the main intent of the project. It will have to be carefully implemented and well planned. The majority of the architecture and layout of the program will be implemented in this iteration. Upon completion of this iteration, supporting development documents and the user manual will need to be updated.

11/20/2009 Custom Iteration 4-Implement notes feature.

Brief Details: This feature will be implemented in the fourth custom iteration. The user will be able to upload .doc, .xls, .docx, .xlsx, .pdf, and .txt files to the application. It may have a member from the team working on it from earlier in the process. Upon completion of this iteration, supporting development documents and the user manual will need to be updated. A possible third party tool may be used to help us implement this feature, such as FileUp.

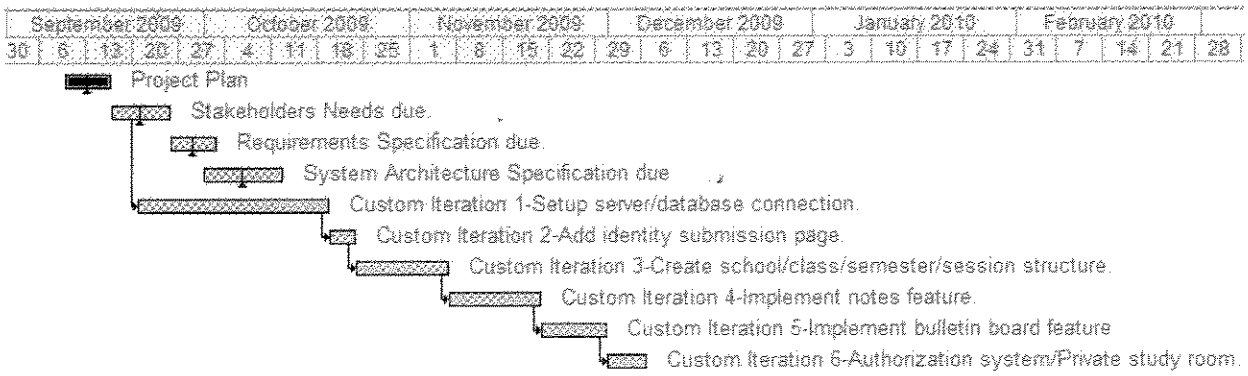
11/30/2009 Custom Iteration 5-Implement bulletin board feature.

Brief Details: This feature will be implemented in the fifth custom iteration. It may have a member from the team working on it from earlier in the process. Upon completion of this iteration, supporting development documents and the user manual will need to be updated.

12/06/2009 Custom Iteration 6-Authorization system/Private study room.

Brief Details: Set up the administrative pages to manage the system, and finalize any loose ends that may occur from earlier in the process. We also will attempt to get the private chat rooms working during this final iteration. Upon completion of this iteration, supporting development documents and the user manual will need to be finalized.

*In the event that we cannot complete all of our iterations, we will present the latest successful iteration to the class. This consists of formulating the final PowerPoint presentation, gathering the stable code for that iteration, and finalizing the user manual and documentation.



(Gantt Chart for Project)

Good initial schedule. Haven't addressed testing or documentation. Appears that there are included in stated milestones. Make sure that they don't get neglected or pushed to the end of the schedule

5 Risk Analysis

5.1 Purpose

The purpose of this section is to provide a process and tools to manage project risks.

5.2 Procedure

5.2.1 Risk Identification

1. The group will identify risks by looking at the uncertainty of a situation and its impact on the project.
2. The group will list all risks that might occur while working on this project.

5.2.2 Risk Evaluation

1. The group will judge each risk and evaluate the probability and impact of each risk.
   i. Probability of occurrence will be rated either low, medium, or high.
   ii. Impact of occurrence will be rated either negligible, marginal, critical, or catastrophic.
2. The group will then look at the probability and impact and give the risk a status of either green, orange, or red.
   i. Green status means that the risk will not affect the scope or delivery of the project.
   ii. Orange status means that the risk will have a minimal affect on the project. The team may have to adjust schedule or scope.
   iii. Red status means that the risk will have a major affect on the project. The team needs to reexamine the scope and schedule.
3. If there is any disagreement within the group about how to categorize a risk, the decision will be made by a simple majority vote among group members.

5.2.3 Risk Response

The group will decide how to react appropriately to a risk that it determines requires response planning. The group will deal with risks in one of four ways:

i. Contingency, where the plan will be invoked one the risk materializes.
ii. Mitigation, where scope will be adjusted to reduce probability of the risk.
iii. Transference, where the risk will be shared and taken by all team members
iv. Do nothing

5.2.4 Risk Reporting

1. The team will discuss the most significant risks at a weekly project meeting. The risks will be evaluated using the Risk Assessment Matrix that is below.
2. Risk Assessment Matrix

	Negligible	**Marginal**	**Critical**	**Catastrophic**
Low	Do Nothing	Mitigation Transference	Mitigation Transference Contingency	Mitigation Transference Contingency
Medium	Mitigation Transference Contingency	Mitigation Transference Contingency	Mitigation Transference Contingency	Mitigation

High	Mitigation Transference Contingency	Mitigation Transference Contingency	Mitigation	Mitigation

5.2.5 General Risk Plan

If a risk occurs, our group will rethink whether the project can recover and at what cost. We will then take a step back in the process and figure out where the problem lies and figure out how to fix it. Since the project has fixed deadlines, we may have to reconsider the scope of the project. If that is the case, we will recalibrate our schedule for the remaining scope in the project. All risks will be analyzed with the Risk Checklist, monitored by the likelihood and status of the risk, and managed by the group.

Risk Type	Description	Probability	Impact	Action	Status
Human	Illness Family Emergency	Medium	Marginal/ Critical	Redistribute Work	Green
Natural	Weather Accident	Medium	Critical/ Catastrophic	Back up source code from SVN on a weekly basis	Orange
Operational	Loss of assets Failure after distribution	Medium	Critical	Extensive testing before distribution Keep track of assets	Green
Procedural	Internal System Failure of accountability	High	Critical	Track progress on current procedure and find ways in can be improved during weekly meetings.	Orange
Project	Underestimate Cost Time Shortage Poor quality Changing Requirements Feature Creep	High	Marginal/ Critical	Track deadlines and adjust schedule if necessary Hold code review to see others work Discuss requirements at weekly meetings	Orange
Technical	Unknown technology Technical Failures Component Integration	Medium	Critical/ Catastrophic	Research unknown technology Create early test iteration to learn about new technology Back up local data	Orange

Is this reflected in your development plan/ schedule?

You do not seem to have treated unfamiliarity w. Facebook development as a specific risk. Would suggest a more specific risk mitigation strategy

5.2.6 Risk Checklist

This checklist will be reviewed at each group meeting to help gain an overview on how the project is doing in terms of risks.

1. Is there sufficient commitment to the project?
2. Is there a well defined process?
3. How often do requirements change?
4. Are deadlines more important than code quality?
5. Are there enough people with proper skills to finish the project?
6. Can deadlines be met with current scope/requirements?
7. Can the project still succeed at this point and how can this success be measured?
8. Are there sufficient resources available for the project?
9. How much and how important are the risk at this time?
10. How will the risks impact other risks?
11. Is there a plan for each risk?
12. Is the process maturing over time?
13. Is optimally quality achievable?
14. Is the project getting closer to completion?
15. How well is the project being managed?

Appendix 1: Documentation Guidelines

Our tool of choice for the project is: **Google-Docs**
Some of the team members have had a successful experience using Google-Docs (a.k.a. GDocs) on earlier projects. It is easily accessible over the web, has a simple interface, and allows easy collaboration and revision control.

A1.1 Issues

There were also some issues with using GDocs; they are listed below with suggested workarounds:

Issue 1: Loss of one data when multiple people edit simultaneously.
Solution: Although this was only an intermittent problem, we should like to avoid it completely by having people create their part of the document in a separate GDoc and merge them after they have cleared review stage and require minimal changes. Parallel (section) documents should be deleted after the main document has been released.

Issue 2: Lack of quality support for Images and other embedded objects.
Solution: It is suggested that other programs (e.g. MS Powerpoint, MS Visio, Paint) be used to create diagrams or other similar objects. These can then be exported to image files (e.g. JPG, GIF, PNG) and inserted into the GDoc. The original diagram/object file should be stored in the SVN repository in a folder with the name of the deliverable it belongs to.

Tables can be made within GDocs itself.

Issue 3: Import/Export to external formats (other than PDF) does not always retain formatting
Solution: When importing/exporting from/to DOC, XLS & PPT, it was often found that formatting was lost. It is thus suggested that, for producing content in parallel/offline, either:
- another GDoc be used, or
- for text, minimal formatting be carried out in the external program; for PPT/XLS, avoid external program.

A1.2 Standard Document Format

The following formatting instructions should be followed:

Cover Page: A cover page including: Title, Team No., Team members' names and Submission Date, is required.

Table of Contents: After the cover page, a Table of Contents (TOC) should follow. GDocs has an automatic TOC creation function and supports up to 3 levels of headings, accessed using the shortcuts Ctrl+1, Ctrl+2, Ctrl+3 for Heading 1, Heading 2 and Heading 3 respectively. Beyond that one should **embolden** the heading for the section, although it will not appear in the TOC.

Text Formatting:
- Font to be used: Georgia 10pt.
- Heading font: Georgia; not bold if fits into a heading-style (i.e. Heading 1, Heading 2 and Heading 3), use **bold** if not a heading-style; Size as per GDocs automatic heading styles, 10pt if not.
- Use *italics* or **bold** reasonably to highlight important ideas within text.

Section Numbers:
Section Numbers are finalized in the review stage, after which no more re-shuffling will be done. Instead of using GDocs' automatic numbering feature for section numbers, manual numbering is used. The following format is used to avoid inconsistency:
- Level 1: 1 <Section Name>, 2 <Section Name>, 3 <Section Name> etc. *(Style: Heading 1)*
- Level 2: 1.1 <Sub-Section Name>, 1.3 <Sub-Section Name> *(Style: Heading 2)*
- Level 3: 1.1.1 <Sub-Section Name>, 1.3.4 <Sub-Section Name> *(Style: Heading 3)*
- Level 4: 1.1.1.1 <Sub-Section Name> *(Use Bold and/or Italics)*
  etc.

Although, there is no limit on levels, it should be mentioned that authors should try to not unnecessarily introduce more levels.

Appendices:
Appendices use the same formatting rules as normal sections. Just append an 'A' to the number, starting from A1

Header/Footer: No special Header/Footer required as it has not been asked for.

Images & Embedded Objects: Unless self-explanatory, all images and other embedded objects should be accompanied by a caption, placed adjacent to it.

Print/Export Settings:
- In "Print Settings", **page numbers** should be enabled to appear in the bottom-center of each page.
- Printing of **Comments** should not be enabled.
- Standard margins apply.

Document Naming Convention:
- Use the convention: Team1_DocumentName(_Section if reqd.)_V(version-no.)
- Section name only required if the document is being generated in parallel with the main document. Parallel documents should be deleted after the main document has been released.
- Version numbers as defined in section 1.3

Internal Review Summary Information:
The last-to-one (unnumbered) section in each document should be the left for reviewers comments
- Format for comment: In section X, (change | error | suggested improvement) .... (by person Y @ Date Z)
- Reviewers can correct minor errors (e.g. spelling mistakes) within text itself
- The author must make correction and respond succinctly to reviewers comments.

E.g.
- Recommended that bounds on the length of User fields first name and last name, if they are to be requirements in the SRS, should appear under the Create User section (Andrew, 10/28/2008)
  - Action Taken:Fixed By Satpreet. (10/28/2008)

Tail-piece:
At the end of each document, the following information should be included:
- Initiator(s) for each section
- Reviewer(s) for each section
- Submitter(s), and
- Time of submission of final document to Instructor

Document Control Tailpiece

Version 1:

Document Contributions:

Project Plan Section	Initiator	Reviewer(s)
1 Introduction	N. Fritze	N. Denklau
2 Initial Req.	N. Fritze	N. Denklau
3 Project Processes	J. Hall	S. Singh, N. Fritze
4 Schedule	J. Trapani	N. Fritze
5 Risk Analysis	N. Denklau	J. Trapani, J. Hall
Appendix and Misc. Contribution	S. Singh	J. Hall

Submitters: S. Singh, J. Hall

Time of Submission: September 16, 2009 5:00PM