CSE 403, Winter 2008
Group Project Specification

Your team has been funded to produce the software project outlined in your proposal. The "customer" hiring you to write the product is the wealthy firm, SteppCo. The SteppCo CEO intends to pay your team for its services by awarding points. SteppCo has upper-level managers ("TAs"), one of whom will meet regularly with your group to discuss its progress.

The overall scope of your project includes the following deliverables and other graded items:

	Points	Due
• Software Requirements Specification (SRS)	50	Mon Jan 28 2:30pm
• Software Design Specification (SDS)	50	Fri Feb 8 2:30pm
• Zero Feature Release (ZFR)	20	Mon Feb 11 2:30pm
• Test plan and initial test cases (TestPlan)	25	Fri Feb 22 2:30pm
• Initial implementation (Beta)	50	Sun Feb 24 11:59pm
• Updated Design Specification (SRS2/SDS2)	25	Fri Mar 7 2:30pm
• Feature-complete "Version 1.0" implementation (v1)	50	Sun Mar 9 11:59pm
• Refactored, updated implementation (v1.1)	20	Sat Mar 15 11:59pm
• In-person customer meetings	10	periodically
• Weekly progress emails	20	every Sun 11pm
• TOTAL	320	

Customer's Project Requirements:

The customers do not know exactly what they want, but they do have the following requests:

- The product must be a web-based application written with an object-oriented programming language.
- The product must have some non-trivial database or server-side data component.
- The product must have some meaning or context outside of computer science. For example, it cannot be a source control system or a web-based Ruby interpreter.
- The product should display the SteppCo logo (which we'd like you to design) prominently on the UI.
- Your product should have a way of generating (fictional) "revenue." A possible way to do this would be an ad-based approach, by providing space on your UI for an ad image. Note that revenue is not the same as profit.
- The product should be as usable as possible, even for people who are not expert computer users (with the exception of projects that are designed specifically for experts, such as development tools).
- The product must be robust against common errors such as invalid user input, lost network connections, etc.
- As before, the product must involve communication between two or more computers. In other words, it should be network enabled, or connect to a remote database back-end, or be a client-server application, etc. If your original proposal did not carefully take this into account, please do so now.
- Your project must be usable for a person on a standard computer. Since it is web-based, it must have a public URL that others can use to access it. Expect that the user will have any necessary libraries and tools (such as a Java or .NET runtime), but after that, the user should be able to use your system without hassle.

Beyond these requirements, you are largely free to make decisions of your own. You should, however, talk to your customers as you plan this project in order to make sure your product meets their needs. For full credit, you should discuss your proposed requirements in some way with your customers before submitting them.

Customer Discussion:

This document is a partial specification for your project's requirements, but much information is intentionally left out. This is to encourage your group to ask questions of "the customer." You may ask these questions in lecture, by email, or on the message board. Major turnins that do not reflect questions with the "customer" may not receive full credit.

Submission and Grading:

There will be an online submission system for turning in documents and code related to the phases of your project.

For phases that require written documents, if you choose to turn in your documents in class, turn them in as printed pages in lecture. If you turn them in electronically, submit them in Word (.doc), PDF (.pdf), or Violet (.violet) format, in a .ZIP file named `TeamName_PhaseName.zip`. For example, if your project is called "Booyah" and you are submitting the SRS phase, your .ZIP file should be named `Booyah_srs.zip`. The documents inside this .ZIP archive should reflect their contents, such as `Booyah_use_cases.pdf` or `Booyah_items_1-3.doc`. You may receive a deduction if you turn in clumsily named or organized files.

Make sure that your project's name and all group members' names appear clearly atop each document. Only one copy of the documents should be submitted for each group.

Specification Changes:

Part of the nature of software engineering is that things can (and often do) change while you are in the middle of a project. We reserve the right to amend or alter any contents of this document during the course of this quarter. If any such changes are made, they will be posted and announced clearly to everyone. We promise not to make last-minute changes to any phase unless absolutely necessary; any major changes will be posted at least one week before that phase is due.

Customer Meetings:

At various points during the project, your group should arrange to meet with your primary customer(s) to discuss its progress. This meeting will count as a small portion of your grade. Your group should come prepared to discuss what has been done, what is left to do, what is likely to be left out, any current problems or risks, and some specific questions you have for the customers as the release draws near. Your project manager plus at least two other group members must be present at each of these meetings.

The exact ranges of dates and times for these meetings will be announced in class and on the web site.

Weekly Progress Emails:

Every week by Sunday night at the latest, your group must send an email message to its primary customer briefly discussing the following information:

- Your team's progress so far
- What each member of your group is working on this week
- What transpired at your team's latest in-person meeting (your team is expected to meet at least once a week) (this is also sometimes called the "minutes" of the meeting, though you do not need to list every small detail)

PHASE 1: Software Requirements Specification (SRS)

The SRS milestone artifact is a set of documents related to your project's requirements and high-level UI design. Your SRS must contain the following items:

1. Requirements outline

Submit a document, 2-5 pages in length, with brief descriptions of each of the following areas:

a. **Product Description, Audience, and Feature Set**: What is your product? What is the target audience you expect to use the product? What are its major features? Include at least 4 major features you will provide, along with at least 2 other minor features or aspects you hope to complete.

b. **Software Toolset:** What programming languages, data sources, version control, and other tools will you use?

c. **Group Dynamics:** For the most part, your group organization is up to you. However, **we require that you choose a single person to serve as your Project Manager (PM)**. Who will be your project manager? What will be the other members' roles? Will everyone share in the development, or will you have designated designers, testers, etc.? Why have you chosen these roles? If a disagreement arises, how will it be resolved? Be specific.

d. **Documentation:** What external documentation will you provide that will enable users to understand and use your product? This could take the form of help files, a written manual, integrated help text throughout the UI, etc. (Comments and Javadoc in source code don't count; they are developer docs, not user docs.)

e. **Schedule / Timeline:** Provide a rough schedule for each member or sub-group within your team. For example, how long you believe your developers will spend working on each major feature listed in your product description? Who will work on the design, and how much time do you expect it will take? Which features are "beta" features? Provide reasonable guesses as much as possible, but you will not be graded on the accuracy of these predictions.

f. **Risk Summary:** Describe at least three specific adjustments you will make if the project begins to fall behind schedule. No more than two of the adjustments you list can be feature cuts; at least one must be some other change or cutback, such as changing specific areas of testing, adjusting your group dynamics or time schedule, etc.

2. UML use cases

Submit the following use case documents:

a. **Two (2) formal use cases** for scenarios you think are two of the most important to your product. They should be similar to Use Cases 1, 2, 3, and 5 from Cockburn's paper, and should include: primary actor, level, preconditions, minimal/success guarantees, a list of steps to the success scenario, a list of properly numbered extensions, and a failure-handling remedy for each extension as appropriate. It is impossible to think of every possible failure case ahead of time. But you should list a reasonable set of extensions and remedies if reasonable ones exist. If you do not have a known remedy, your use case should state this and explain what will be done to investigate possible remedies.

b. **One (1) casual use case** in paragraph form for one other scenario that you think is important to your product, perhaps of lower importance than the two you chose to depict formally. This use case should be similar to Use Case 4 from Cockburn's paper, describing the main success scenario first in paragraph form, then listing each extension and its remedy (if known) in a second paragraph.

3. UI prototype

Submit diagrams containing rough sketches of your product's user interface. These diagrams should depict the major UI used to complete the use cases you submit. For example, if one of your use cases is to Purchase Stocks, you should draw the initial UI that is presented when the user wishes to purchase a stock, along with any other major windows, messages, etc. that appear as the user navigates through this use case.

Submit at least **two (2) UI diagrams**. The diagrams can be drawn by hand or computer, or can be screenshots of an actual prototype. If a window leads to a dialog box, drop-down box, etc., include it as a sub-diagram.

Your diagrams do not need to be pretty to get full credit, but they should be legible and reflect some forethought about what options will need to be shown and how the user will use the software.

PHASE 2: Software Design Specification (SDS)

The SDS milestone is a set of documents about your project's design. Your design should specify how to implement an object-oriented product to meet the requirements in your SRS. Among other things, your SDS should answer the questions: what are your classes, what are the responsibilities of each class, and how do the classes collaborate?

Your SDS must contain the following items:

1. UML class diagram

Submit a **UML class diagram** for your system in the format shown in Fowler, Chapter 3. Your diagram should display all major classes, attributes (fields), methods (do not list `get` / `set` / `is` methods), inheritance/interface relationships, and associational relationships (named and directed, with multiplicity adornments).

Your design will be evaluated on completeness as well as level of thought, attention to principles discussed in class, and proper UML syntax. Follow Riel's OO design heuristics, such as:

- use encapsulation (Heuristic 2.1)
- keep related data and behavior in the same place (Heuristic 2.9)
- minimize each class's public interface (Heuristic 2.3, 2.6)
- emphasize cohesion and limit coupling (Heuristic 2.7, 2.8)
- avoid "god classes" (Heuristic 3.2)
- avoid insignificant or irrelevant classes (Heuristic 2.11, 3.7, 3.8)
- model-view separation and model independence from view (Heuristic 3.5)
- avoid irrelevant "agent" or "controller" classes (Heuristic 3.10)

Distribute your project's functionality and allow for features to be developed in parallel as much as possible.

2. UML sequence diagrams

Submit two **(2) UML sequence diagrams** that depict your product executing two of its important use cases. These can be the same use cases you wrote about in your SRS. The sequence diagrams should follow the format of the examples from Fowler, Chapter 4. Your diagram should show all participants (objects) in the sequence, all important directed messages between them and their return values (if any), as well as interaction frames with proper operator adornments as appropriate. Use good design with decentralized control; no one class or object should do the bulk of the work.

The sequence diagrams show the "life" of a user's web request. Show the request's path through your UI, server, and/or data layers as it interacts with each to accomplish the task.

Accompanying one or both of the sequence diagrams should be a pseudo-code description of the same algorithm, similar to Fowler's Figure 4.4.

3. Coding style guidelines

Submit a document explaining what style conventions you plan to follow (a reference document or link with an example would be helpful), and how you plan to enforce a consistent coding style between group members. Describe any tools you plan to use to enforce these conventions, and/or any methodologies your group members plan to use to enforce them, such as code reviews. If you plan to do code reviews, describe how you will provide evidence of these reviews to the grader. Will you take review notes? Will you use a tool to annotate the reviewed code? Etc.

4. Presentation

Submit a set of 5-10 slides in PDF, PPT, or ODP format of a brief presentation summarizing your project and the work you've done on it so far. You should talk about the overall project idea, its major features as outlined in your SRS, some high-level aspects of its design including at least one of your design diagrams, your languages and team roles, and so on. Each group will be given approximately **8-10** minutes. At least 3 group members must participate in the presentation.

Use at least two diagrams in your slides to receive full credit; these can be taken directly from your other documents.

PHASE 3: Zero Feature Release (ZFR)

The zero-feature release milestone consists of a skeletal implementation of your product and a document with instructions for accessing some of your development tools. The purpose of this milestone is to work out bugs in version control, bug tracking, deployment, etc. Unless otherwise specified, no functionality needs to be working other than a "front page."

1. Featureless live product web site

Since your project is web-based, provide us with a URL to reach the front page of your product.

2. ZFR instructions document

Your group should also submit a ZFR document describing the items below. Some documents describe processes the user or developer must perform. Part of your grade will be based on the simplicity of these processes, and how accurately your directions match what the user must actually do.

You should also put the resources in place so that the graders can examine and test the processes described in the ZFR document promptly after your submission. You may want to arrange a "dry run" with the customer beforehand. Your ZFR document should address the following items:

a. Source control and build process instructions

Your project's build process is the set of tools and commands to compile and "build" your system. Turn in a set of directions to find your build system, check out its files, and build them. In grading, we will follow these directions.

For full credit, **your team should have a reasonable resolution to the issue of version control**. For example, you could handle this issue by using a CVS or SVN repository in a reachable location, or by hosting the code on a public system such as SourceForge. Your instructions should explain how to access any such system or repository.

The directions should be written in sufficient detail that an intelligent developer can follow them. If the system(s) require login information to access them, you should provide this information.

Part of your grade for this item relates to the number and complexity of commands the developer must use. Ideally your system will have a single command that does a "one-step build," that checks out all source code from your repository, builds all necessary binaries, packages them, and places them in a known location.

Beginning with the night of your ZFR turnin, **the grader will log in to your version control system once a week** to count the number of files and lines present in the system, as well as the number of lines modified, and log your group's development progress. Lack of significant weekly progress may impact your team's grades on later phases.

b. Bug-tracking system instructions

Describe the set of tools used to document existing bugs and missing features in your system's code. Turn in a set of directions to your customers, describing briefly how they find your bug-tracking system, examine the list of current bugs, and file a bug. In grading, we will follow these directions and examine whether the bug tracking system exists and is usable. The directions should be written so that an intelligent developer can follow them.

Your bug tracking system does not need to contain a comprehensive list of bugs or missing features. But for full credit, it should have at least one bug filed for each active developer. These bugs may be requests for features, such as "TODO: implement login behavior." If possible, the bugs should list priority and a timeline to fix them.

Beginning with the night of your ZFR turnin, **the grader will examine your bug-tracking system once a week**. Part of your grade in later phases will reflect whether a significant number of accurate bugs are present that contain proper information such as severity and assignment to specific developers to fix them.

c. Data access instructions

Since your product must have a server-side data component, your ZFR should contain a set of instructions about where this data is stored and how to access it. Turn in a set of directions that tells the grader how to find your data, and how to briefly perform a trivial access of this data. For example, if your data is in a database, inform the grader how to connect to this data and perform one very simple query against it. In grading, we will follow these directions.

PHASE 4: Testing Plan Document (TestPlan)

Your test plan is a document describing your plans for how you will test your product to assure its quality. Submit a document, approximately **2-4 pages in length**, addressing the following categories of testing:

1. Unit testing

List which classes of your project you intend to unit test, and if appropriate, which of their methods and behavior. For any classes you do not plan to unit test, briefly justify your decision not to do so.

Also *briefly* describe the process of your unit test development. Who will write them? Will they be black-box tests (written by someone who has not seen the source code being tested), or white box (by someone who has seen the complete source code)? What are some of the more significant cases you plan to test, such as boundary conditions and expected error cases?

For full credit, your project's eventual code submission must demonstrate significant unit test coverage, covering approximately 30% or more of the code. Therefore, in your test plan document you should describe how you plan to prove to the customer that you have achieved this coverage. Perhaps you will want to use a tool that displays unit test coverage percentages, such as NoUnit.

2. System testing

Describe the ways in which you plan to test your product as a whole. For full credit, your eventual product must undergo non-trivial testing in at least **two (2) areas of system testing**, such as the following:

- Automated UI testing (such as by Selenium)
- Performance testing / profiling (memory or CPU usage)
- Load / stress testing
- Security testing and auditing
- Usability testing

In your test plan document, describe which kinds of system testing you plan to perform, and also how you intend to demonstrate to the grader that you have performed them. Testing that is done anecdotally without any resulting evidence may not be given credit.

PHASE 5: Initial Implementation (Beta)

For the "beta" product release, you must produce a working initial version of your project, reflecting many of the features listed in your SRS and SDS. Submit or provide the following items:

1. Binary Distribution

The binary distribution contains the resources necessary to run and use your system. Since your system is web-based, its binary distribution consists of the site being up and running by the due date.

This item will be graded on whether it reflects substantial work and effort on the part of your team, has a solid and polished user experience, and successfully implements the usage cases you have described in previous phases.

Your product need not necessarily be 100% bug-free to receive full credit, but any known bugs should be documented, and a user testing the system should not encounter a non-trivial number of bugs that are unlisted in your bug-tracking system. Your system should be robust so that errors occur gracefully as much as possible.

This item will be graded on whether it demonstrates the following attributes:

- a non-trivial amount work and effort
- completeness of a non-trivial number of features outlined in SRS document (especially a non-trivial number of the features you listed as "major features" in your SRS)
- robustness, to the degree that the system can be tested and used in non-trivial ways (system may have some bugs)

2. Source Distribution

The source distribution contains all source code and other resources that were created by the development team. These resources should be bundled into one or more compressed archives. Assume that this item is being prepared for one or more developers who would pick up development where you left off.

Your code will **not** be examined in detail in this phase, other than to verify the following:

- that it reflects non-trivial work and effort
- that each file has a clear comment that names its author(s) (or that code authorship is present in some way)
- that several members of the team have made contributions to the source code

Your code will **not** be graded on style and design, commenting, elegance, redundancy, and so on.

3. Initial Testing Resources

Your code does not need to be thoroughly covered by testing and quality-assurance tools in the beta phase, but we will look to see that you do have at least the following:

- a non-trivial unit test for one major class of your system
- one of the following three items:
    - a non-trivial unit test for another major existing class of your system, OR
    - a unit test for part of the system that is **not** yet implemented (in the "test-first" style), OR
    - at least one non-trivial testing item that is NOT a unit test.

PHASE 6: Updated Design Documents (SRS2/SDS2)

The "SRS2/SDS2" phase is an update of some documents produced in earlier phases. On real software projects, design documents and specs are "living" documents that must be updated to stay in sync with the product that is being created.

Submit or provide the following items:

1. Requirements Revisited

Submit a document briefly revisiting the following aspects of your initial software requirements spec (SRS):

a. **Progress Assessment**: How does your team as a whole view its progress to date on the project? Are you confident in the progress you are making, and are you happy with the work that has been completed so far? What is your outlook for the rest of the project?

b. **Features and Cuts**: Do you still believe that you will complete the major functionality and features listed in your SRS? What about the extra features and frills you listed? If not, why not? What features do you plan to cut to help you complete the project in time, and why have you chosen these to be cut? How much work do you estimate you will save by cutting these features?

c. **Group Dynamics**: What are your team members' current roles, and what will be their responsibilities for the remainder of the project?

d. **Revised Schedule**: In your original SRS, you submitted an approximate schedule for how long you thought your team would spend its time. In this phase, submit a new schedule showing how each member has actually spent his/her work time, which is likely very different than your original estimate. (Appropriate units of measure are "days" or "weeks" on a given task.) In particular, describe approximately how much time you have actually spent so far on each of those features, along with estimates for how much more time you think it will take to finish any of them that went unfinished. Try to be accurate within a few developer-days for each feature; you may want to look at your version control logs to get a more accurate picture of how long you've spent working on each feature.

2. UML Class Diagram, updated

Submit an updated version of your UML class diagram from your SDS that reflects the actual code that has been written so far for your project. (The closer your actual implemented design is to your original SDS design, the less work to do!)

3. Sequence Diagram, updated

Submit an updated version of **one** of your two UML sequence diagrams from your SDS that reflects the actual code that has been written so far for your project. (The closer your actual implemented design is to your original SDS design, the less work to do!)

These documents will be graded on whether they do in fact match the design and flow of your source code as turned in, on their adherence to UML syntax as taught in class, and on the soundness of the underlying design they depict.

PHASE 7: Feature-complete "Version 1.0" Implementation (v1)

For your "v1" product release, you must produce a working initial version of your project, reflecting many of the features listed in your SRS and your schedule. Submit or provide the following items:

1. Binary Distribution

The binary distribution contains the resources necessary to run and use your system. Since your system is web-based, its binary distribution consists of the site being up and running by the due date.

This item will be graded on whether it reflects substantial work and effort on the part of your team, has a solid and polished user experience, and successfully implements the usage cases you have described in previous phases. Your product need not necessarily be 100% bug-free to receive full credit, but any known bugs should be documented, and a user testing the system should not encounter a non-trivial number of bugs that are unlisted in your bug-tracking system. Your system should be robust so that errors occur gracefully as much as possible.

2. Source Distribution

The source distribution contains all source code and other resources that were created by the development team. These resources should be bundled into one or more compressed archives. Assume that this item is being prepared for one or more developers who would pick up development where you left off.

This item will be graded on whether it demonstrates the following attributes:

- work and effort
- well-designed according to the heuristics we have learned in class
- making use of design patterns as appropriate
- being otherwise elegantly and robustly designed
- documented using comments on each file and significant method as appropriate
- general cleanliness and elegance of the code

Since the code is to be submitted in a state suitable for being turned over to other developers, you should document it sufficiently so that they could read and understand it. This includes summary descriptions of each file along with comments on methods and complex sections of code as appropriate.

3. Testing Resources

Your testing resources include any unit tests, automated testing facilities, testing plan documents, and other testing tools or artifacts you have created as part of the quality assurance of your project. You are required to administer unit testing over a significant portion of your application's model code using a framework such as JUnit.

To receive full credit your project must include not only substantial unit testing but at least **two (2) other areas of significant additional testing**; for example, you could perform documented usability using clearly defined test scenarios with volunteer users from outside your group, or you could create automated UI tests using a system testing framework such as Selenium. Part of your grade for this item will depend on your choosing a reasonable subset of functionality to test and choosing effective test cases (in number, scope, and coverage) to ensure the quality of your implementation.

4. User Documentation

Your product should contain documentation explaining the usage of the system to the user. As in the previous releases, this documentation is directed at a user and not at a developer, so it should focus on the user experience and not on the system's implementation. The documentation will be graded on whether it covers all major areas of usage of the system as well as its quality and completeness. Some of this documentation may be integrated into the product itself, but this should not come at the expense of a solid user experience.

PHASE 7: Refactored and Feature-added Implementation (v1.1)

The final project milestone is the "v1.1" product release. Submit or provide the following items:

1. New feature implementation

Around the time that your "v1" release is being completed, we will announce a **new non-trivial feature** that your group must add to the product for its "v1.1" release. A major portion of your grade for this phase will be based on your completion of this feature. This is a test of your ability to design for change, and the ability of your team to maintain its code over time.

Not only will you be graded on correctly implementing this functionality, but your source code will be examined to see how invasive the change was to your code and design.

2. Bug fixes

Your v1.1 release should contain **fixes for at least 3 non-trivial bugs** that were present in your v1 release.

3. Demo / presentation

During the final week of the school quarter, each group will give a presentation demonstrating its project. This presentation should be roughly 10 minutes long. You do not need to submit slides for the presentation but may do so if you like. The bulk of your presentation should cover a demonstration of the usage of the project itself. For full credit, at least three (3) of your group members must participate in the presentation.