PROJECT HANDOUT

Software Systems Design
CS190, Spring 2000
Steven Reiss

Welcome to CS190. In this course we attempt to develop USEFUL software systems. Your goal will be to develop a system that is useful, friendly and powerful enough so that people will actually use it. You are going to work as teams of around 10 people to develop a single software system. Everyone is expected to contribute to the specification and overall design of the software. Moreover, each individual will be responsible for particular portions of the detailed design and implementation of the overall system. The final product should be a system that everyone will want to use. This course will be intensive, and you will be expected to strictly honor the various deadlines, both individually and as a group. Failure to do so will probably result in the failure of the project which will result in an appropriate grade for everyone in the project.

TEAMS

CS190 is a course where you will work in teams. The class will be divided early on into groups of about ten students each. These groups are responsible for finding, defining, designing, and building a software system. Much of the course will involve learning to work effectively in teams and as a team. Teams are expected to organize themselves, to hold meetings out of class as appropriate, and to determine and solve problems as they arise.

PROJECT

Project(s) will be determined by a vote of the team (subject to approval of the professor and TA). Any project that is chosen or that you want to propose must meet the following criteria:

- It must be something that other people will use. This encourages you to have a set of users to get requirements from other than yourself. It also means that the project should be something new, e.g. something that doesn't already exist on the target platform.
- It must be something that can be scaled to a large project (i.e. larger than a group would typically expect to get done in one semester). We want to teach aspects of design that encompass systems that are commercial in scale even if we can't build them practically. Moreover, we want to stress design for extensibility, i.e. designing a system so as to anticipate change. The scale should be roughly thought of as that of a typical commercial PC product.
- The project must be amenable to being divided into logical components for group programming. This will typically be the case for a large enough projects.

- The project must be suitable for implementing in Java or C++ on Sun, Linux, or Windows platforms.

One way of meeting these criteria that we will use is to ensure that the project has a set of ready customers. To that end, the project must be sponsored by someone outside of the course. This could be a faculty member in CS who needs something done, a faculty member in any other department at Brown, an outside company, or any other organization with an interesting problem. If you just build a system for yourself to use you will miss out on a lot of what goes on in software engineering and the experience wont be as educational or as fulfilling.

As an example of a project, I would suggest implementing a decent debugger for linux, something that will replace gdb. The principal goals here would be to have a debugger that works correctly for C++, shared libraries, and multiple threads. It should provide both a textual and a graphical user interface. It should provide all the capabilities of dbx and gdb through a logical and not too cluttered user interface. It should be designed to handle and control multithreaded and multilingual applications. Extensions would include memory checking, profiling capabilities, data structure display, and race detection and analysis.

Another example project comes from Mike Pesta, the registrar at Brown. Brown has one of the most antiquated, inefficient, and messy registration processes in the country. Basically everything is done manually, nobody (students, faculty and the registrar) knows what is going on at any time, and everything is sure to get messed up. Brown is currently thinking about how to replace this system with a computer-based approach that would provide everyone with instant information, would offer fair ways of dealing with limited enrollment classes, and that could accommodate the Brown curriculum. While it is way beyond the scope of this course to build such a system, it would be very useful to the registrar and to Brown if someone would build a prototype system that would let students, faculty and the registrar see what things might look like and how things might work. Such a system would emphasize what the user interfaces to the different groups would look like and might not provide the security, integrity, and data accessibility that the final system would offer. It would, for example, allow experimentation with different ways of dealing with limited enrollment courses and with how permission signatures can be obtained electronically. It would also help convince the faculty, especially those that might not be computer literate, that they should want and could effectively use such a system.

The actual specification of the project (i.e. what it will actually do) is limited only by your and your team's imagination and the constraint that it be fully working by the end of the semester. You should think of what a minimal system implementation should provide, and you should dream of what a fully extended, lots-of-bells-and-whistles-type applications would provide as well. From a commercial point of view, we will be implementing version 0.1 of the system, but designing for version 6.0.

EXTREME PROGRAMMING

There are many approaches to software engineering. One that has been proposed in the last few years in XP or extreme programming. Extreme programming contains a lot of good ideas (and some that are probably not so good). It has been enveloped in a lot of hype and not that much substance. There have been a few experiments to see how effective it is, with mixed results — some show significant improvement while others show no improvement or some degradation in the software development process. In any case, since it is a viable approach to team projects, we are going to use it more or less to construct the projects.

Extreme programming consists of a set of ideas that are generally presented together but that can actually be separated and used individually. These include:

- *Strong interaction between the developers and the customers*. This comes from customer-oriented application development and has proven itself successful independently as long as you can find customers who are willing to commit the amount of time and effort that is needed to make it work.
- *Using stories to specify system requirements*. This comes from the use of scenarios to do software requirements analysis. It is a worthwhile, but probably incomplete approach. We will use it but will probably augment it with some standard requirements and specifications techniques.
- *Testing everything as you go*. This is a good idea in and of itself. It involves design and coding for testability as well as developing test programs and cases while developing the code.
- *Small releases*. The idea here is to get some initial version of the system up and running as soon as possible and then to incrementally update that until the final system is produced. This is a combination of prototyping, a spiral-model of development, and incremental development. While a good idea in practice, it takes a solid initial design to make it work smoothly — the initial design must take into account all the extensibility that will be required later on, some of which will not be known at the time. Making this work also requires doing a lot of planning both in advance and as the development progresses.
- *Code simplicity.* Simplicity is something that is impressed on the students in CS32 and is known as a good idea in general. The fact that XP endorses it is a plus for XP.
- *Pair Programming*. This might be the most controversial aspect of XP. It states that all programming should be done by two people working together at a keyboard, one typing and one looking over the shoulder. This has the advantage of having the code checked as it is typed — something that should reduce both syntactic and some semantic errors early on. It has the disadvantage of requiring both persons to be present and only having one of them work. It is something we will try and see how it works.

COURSE MECHANICS

This project will be done using an object-oriented language (Java, C++ or C#) on one of the platforms available in the department (Sun, Linux, or Windows). You are be expected to write efficient programs that will make use of all the tools that are available to you. The use of appropriate languages, libraries, and tools should be considered part of the course.

We will form teams early in the semester (within the first week of classes). Teams will consist of ten students each (more or less). While we hope to keep the teams intact throughout the semester, we will allow some flexibility and will let people move around during the early part of the semester if necessary.

The first job of the team will be to organize itself. Each group will choose a project manager and a project librarian for their project. The project manager is the one who will be responsible for coordinating and supervising everyone. She/he will have the power to make modifications to the design or specifications and to assign or reassign people to different tasks. The project librarian will be the individual responsible for maintaining a project notebook and eventually coming up with user documentation, both on-line and a user's manual. Note that responsibility here does not necessarily mean that he/she has to do the actual work, just coordinate it. Everyone else will be assigned a particular aspect of the implementation as their responsibility. Note that we expect each group to work as a coordinated team, with people helping out, providing feedback, etc. to others as necessary.

The second job will be to choose an appropriate project. The best way of doing this will probably be to have individuals or small teams go out and find project ideas and sponsors and then to discuss these with the rest of the team to ascertain the level of interest, expand on ideas, find additional sponsors, or come up with additional projects.

Once the project is discussed among the team members, each team will be responsible for developing a requirement document for the project. This will be done in two parts. The first part will list the user requirements. This should be done in terms of stories (see the XP book for details) as well as a list of the desired behaviors of the system along with priorities for these behaviors. The second part will detail the requirement specifications. Here you will state what it is that will actually be built (as opposed to what the customers wanted built). This should again be done in terms of stories, appropriate pictures (for example diagrams of the proposed user interfaces and system data flow), and a readable English description of what is to be done. This will form the basis for all later system development. The team can propose multiple possible projects at the first stage, but should settle on a single project (with the help of the professor and TA if needed) before producing the requirements specification. There will be in class presentations (with appropriate feedback) for both the initial project requirements and the project specifications.

The next stage in the course will involve developing a top-level design for the project. This should outline how the system is to be built, what are the major components, and how this is going to be done in an object-oriented fashion. Each team will be responsible for developing an on-line design (using UML as appropriate) that describes the major components

and provides an incremental implementation strategy in line with extreme programming. This will be handed in.

The top-level design will then be the basis for an initial (minimal) implementation of the project. This implementation does not have to do that much, but it should provide at least a kernel that the system can be developed on top of. It might, for example, provide a user interface skeleton or a default back end as appropriate. We are targeting March 1 as the date where this version will be operational. At that time each team will give a presentation of their design and development plans along with a demonstration of the system.

The remainder of the course will involve extending this initial version to a final version of the system using a series of releases. Tentatively we have scheduled a new system release each Friday and will provide class time for the team to demonstrate the functionality of the release, discuss the design issues that arose in building it or that are involved in developing the next release, and get feedback and ideas from the rest of the class. Each team is also responsible for showing these releases to their customers and obtaining and presenting appropriate feedback from this source as well.

Note that each release should contain not only the working code, but also appropriate design and user documentation. The project librarian is responsible for gathering and organizing this information. It will be checked periodically by the TA and/or professor.

The final systems (i.e. full functionality), should be done by the public demo at the end of reading period. Everything should be implemented by then (or completely forgotten) and almost all bugs should have been found and eliminated. This will give us a week and a half for formal user testing, polishing, and leeway in case deadlines are otherwise missed.

MANAGEMENT

You will find that CS190 is as much a course in management and interpersonal relations as it is a course in programming. This is especially true as groups get large (i.e > 5 people). The effort that is required for communication and coordination within a group is substantial. The project managers that are chosen will effectively be the class dictators (subject to being overruled by the professor or TA), who will have the final say in all decisions, etc. While democracy is nice in practice, it tends not to work well in software design.

PROJECT TIMETABLE

- ***1/25 Project Ideas***: Everyone in the class is responsible for coming up with one or more project ideas at this point. You need not have a particular customer, but should have one or more customers in mind. You should be prepared to briefly describe your idea to the class. You should also be prepared to briefly describe the types of projects you might be interested in working on.
- ***1/28 Team Selection***: Based on peoples project preferences, friendships, and a random number generator, we will divide the class in teams of around 10 students each.

- **2/1** ***List of Requirements***: Each team is responsible for providing a description of one or more possible projects from the customer's standpoint by this date. This means that there should be one or more identified and committed customers, a set of stories, and a list of what the system might do, prioritized as appropriate. There will be in-class presentations of the proposed projects in class on 2/4.
- **2/11:** ***Specifications Document***: Each team will choose one of the projects that was described in the previous presentation and will develop a relatively complete description of the software system they propose building by this date. This proposal should include a description of the proposed system and how it will be implemented. The description should include a discussion of the objectives and how they will be met; the inputs, outputs and actions that the system performs, complete with pictures as needed for understanding; and a discussion of the user's model of the system. The latter should be enough to give the reader a feel for whether the system will be useful, and should be done in terms of the story motif used by XP. The implementation discussion should outline a strategy whereby a core system can be built and then extended to achieve the XP concept of small releases. The specifications will be discussed by the class as part of your (and others) presentations, and should be amended appropriately.
- **2/20:** ***Top-level Design Proposals***: Each team is responsible for developing and turning in an implementation framework for their project by this date. This should describe the major components of the system, and provide a detailed strategy for developing the system in a series of releases. The actual design should be a 5-10 page document that includes lots of pictures and stories and provides:
  - A user interface specification, with diagrams if appropriate.
  - A functional specification.
  - A module breakdown for the implementation, with an interconnection diagram.
  - A proposed implementation schedule.
  - A breakdown of responsibility for the various components.
- ***3/1: Initial System Release****:* On this date each team is expected to have an initial implementation of their system. Such an implementation might have very little functionality, but it should provide the framework on which the rest of the system will be built. There will be in-class presentations of the design of the system along with demonstrations of the initial prototype.
- **3/8, 3/15, 3/22, 4/5, 4/12, 4/19, 4/26:** ***Next System Release***: Each Friday following the initial system release will be devoted to analyzing the subsequent release of the system. This will include a presentation by the team of what was done during the week, what design problems arose and how they were solved, as well as what is planned for the next release and the design issues that are involved there. The presentation should also include a demonstration of the system in its current state.
- **5/8:** ***Public Demo:*** The "final" release of the system (version 1.0) should be done by this point. We will organize a public event (with refreshments) so that each team can present its project to the rest of the department as well as potential customers and show off their work.

- **5/16:** ***System Submission***: This is the final date for submitting the system (it can be submitted any time after the public demo). The final submission should include hard copy of all the code, the design documents, and the user documentation. It should include evaluations of the project from each team member (these are private and will be kept so). It should also include a statement from the projects' customer describing their impression of how well the team did and how well the project meets their needs.