

Department of Computer Science
Concordia University

Software Engineering — COMP 354 Project Outline
Fall 2004

Introduction

An important activity in the course is doing a group project. The most important aspect of the project is experiencing the process of software development in order to appreciate how the various phases interact, and to see the roles of the deliverables. The second most important aspect is to experience the group dynamics and interpersonal relationships of team work.

You should expect to **average** a total of 10–12 hours per week on the course as a whole. It is important to balance the effort that group members have available to devote to the project against the group’s expectations for their product. To this end we discourage the aimless introduction of “bells-and-whistles”, but do encourage due attention to quality through careful reviews and testing. The largest waste of effort in projects occurs because of **rework**: having to re-do something (perhaps several times over) because

1. the task was not understood the first time it was performed, or
2. poor-quality work was done the first time, or both.

So devote time to discussing the purpose of each task, and whether the input(s) such as the user’s requirements are understood well enough to perform the task. Also do your best to do a high-quality job the first time around: you need to understand how your output will be used and by whom, and present the right information in the right format. Follow standards such as an agreed style for the requirements document, the design documents, and *the code standard*.

Project

The project is to develop a *Material Tracking System* **and** the simulator which is the test infrastructure. You will need to produce a *requirements document*, *test documents*, *design documents*, as well as thoroughly documented and tested code.

Title of Project: Material Tracking System

Brief Project Description:

A material tracking system tracks the location of material. Examples include inventory systems, UPS logistics systems, and laboratory management systems. The key concepts are the material themselves, and the locations. Note that both these concepts may be

composite (or recursive) entities: that is, a material can be made of components which are themselves material, and a location may contain more precise locations. For example, for a grocery inventory system a material might be a pallet of boxes, where each box is a material. Also the box may contain individual items, which are material. The location of a crate might be in a building, on a particular floor of the building, in a particular room on the floor, or in a particular refrigerator or shelf in the room.

Material and/or locations might be identified by labels, bar codes, or radio-micro-chips.

The basic scenarios of the material tracking system are (1) to identify an entity, ie by scanning a barcode; (2) to place or move some material; (3) (perhaps) to move locations, ie, a refrigerator might be moved from one room to the next; (4) to report the location of a material; (5) to report the inventory at a location; and (6) to unpack (or pack) a material into its subcomponents.

The test infrastructure requires a simulation of the barcode readers, as well as entry of the appropriate sequence of move, locate, and report events. The simulation produces a log of events and system states as an audit trail, so that the correct working of the system can be validated.

Programming Language: Java (or C++)

Development Environment: H-929 Lab; Linux preferably, or Windows.

Project Phases

The incremental project will follow three phases — Requirements Analysis (SRS), Design, and Code & Test (IVV) — each lasting four weeks, and each with major **deliverables** at the end of the phase. A *review/presentation* is given prior to the end of the phase, so that all teams and groups are kept up to date, and to provide an opportunity for feedback, in the hope that all serious problems will be detected and removed from the deliverables before (!) delivery. Still, it is likely that later phases will detect minor problems and issue a *change request* so that the problems can be addressed.

Groups

Each group consists of three teams (SRS, Design, IVV) each with about 3 team members. Each team has a leader and each group has a group coordinator. Ideally a group should consist of 10 ± 1 members.

Each team is busiest during its corresponding phase of the project. During that phase, the team is loaned one member from each of the other two teams. So for individual weeks your workload will be much higher than average depending on your role in the project and the phase of the project. So plan your time accordingly.

Each group needs a diversity of skills: communicators, writers, organizers, modelers, as well as C and Unix technical expertise. When you form your group make sure you have the necessary skills amongst your members.

General Overview of Project Schedule

Week 1 Group Formation

Week 2–5 Requirements Analysis (SRS)

wk2 SRS Team studies SRD Guidelines, particularly use cases
SRS Team studies sample SRD

wk3 SRS Team drafts SRD, performs its own review of draft SRD;
IVV Team develops test cases from use cases

wk4 SRS Team Review/Presentation of SRD
Design Team studies sample Design Documents
IVV Team provides feedback on SRD

wk5 SRS Team Deliverable: SRD

Week 5–9 Design

wk8 Design Team Review/Presentation of Design

wk9 Design Team Deliverables: Architecture, Interfaces, Detailed Design
IVV Team Deliverables: Test Plan

Week 9–13 Code and Test (IVV)

wk10 IVV Team should have completed all test drivers

wk11 IVV Team should have tested all modules

wk12 IVV Team completes integration and subsystem testing

wk13 System Demonstration and Final Deliverables

SRS Team: SRD

Design Team: Architecture, Interfaces, Detailed Design

IVV Team: Code, Test Plan with test cases, schedule, results

Group Coordinator: Final status report

Everyone: evaluations

Prologue: Group Formation

The class will be divided into GROUPS with approximately 10 members in each group. This will be done in the first week. The formation of groups is the responsibility of the students, so discuss your individual skills, strengths and weaknesses, and form a balanced group of people with whom you can work closely. The group will need communicators (for writing documents and giving presentations), organizers (to make sure work is distributed fairly and deadlines are met), and technical skills (particularly in Java, C++, but also Linux).

Each group will elect a COORDINATOR by the second week.

Each group will be divided into three TEAMS called

- SRS (Software Requirements Specification) team,
- DESIGN team, and
- IVV (Implementation, Verification, Validation) team.

Each team will have three members and the group coordinator will be a member of each team. Include C programmers in your IVV team. Every team will elect a team LEADER who will be the permanent member of that team for the duration of the project.

The project begins in the second week and the project demonstration is held in the final week of classes. That is, the project is spread over twelve weeks. There are three phases, called PHASE-1, PHASE-2 and PHASE-3 ending at weeks 5,9 and 13 respectively. At the end of each phase, a major deliverable is due and it will be counted for overall grading.

In PHASE-1, SRS team will have a heavy load. It will have its members and two additional members, one from the DESIGN team and one from the IVV team, on loan for this phase on developing Software Requirements Document. In PHASE-2, the DESIGN team will have a heavy load and will have one member from each of the SRS and IVV teams on loan to them. A similar procedure is to be followed for PHASE-3 with the IVV team getting an additional member from each of the SRS and DESIGN teams. During each phase the team leader is responsible for coordinating the activities, scheduling meetings, submitting deliverables (on time!) and conducting reviews. Reviews may be conducted during lectures, tutorials, or lab hours. All members of the team must be present to give presentations to the class, answer questions and participate in the discussions.

Members of a team report to the team leader; team leaders report to the group coordinator; the group coordinators report to the lab demonstrators and the instructor. Follow the same hierarchical structure for getting or sending information and/or report problems of technical and interpersonal nature. When you think that you are in trouble, get help fast.

By the end of second week, the groups and teams should be in place. The coordinator of each group should submit to the instructor and the lab demonstrators the following information:

GROUP NUMBER

1. SRS Team (leader, members, extra members for heavy phase)
 2. DESIGN Team (leader, members, extra members for heavy phase)
 3. IVV Team (leader, members, extra members for heavy phase)
- Name of Group Coordinator

Make sure everyone knows how to contact everyone in the group, and that everyone knows everyone's weekly schedule. In particular, each group and each team should agree on a convenient time for group meetings and team meetings. Keep in mind that a team may need to meet several times a week.

It is especially important for those people in the evening classes, part-time students, and working students that times for meetings be arranged during group formation.

Hints on Group Formation

Up-Front Setting of a Joint Goal

Many group problems can be avoided by being very honest at the beginning. Each person has different priorities, different workloads from other courses, jobs, and family, and different skills. As a group you must share a goal: to perform the project tasks, deliver the documents and software, and to learn from the process. That goal can be achieved at many different levels depending upon the effort expended, the initial skills of the group members, and the synergy that develops in the group. Each level has merit! At each level you will learn!

Much time, energy, and goodwill can be dissipated in futile inter-personal accusations that X is not doing enough, or that X is not meeting deadlines, or that X is doing poor work. Typically these views are due to differing priorities and external pressures.

So discuss the issues of expectations, priorities, workload effort, and external pressures when you form a group. Be very honest. Agree on a goal or workload that everyone is comfortable with, and then do the best you can within those limits. Be supportive. Look for ways of reducing effort by doing things right the first time! Make sure each person knows what it means to "do their job right" before they begin.

Friends or Strangers

Many groups form around friendships, and some have a tendency to exclude strangers. Yes, you need to be able to work with the members of the group, but it is not necessarily true that a good personal relationship means a good working relationship.

You also need a range of skills in a group and these may only be found outside of your circle of friends.

Another tendency is to exclude people based on their seemingly poor GPA. Unfortunately GPAs reflect performance in examinations, and are not a good indicator of a person's

group skills or technical knowledge.

Keep an open mind, seek diversity, and listen to what everyone has to contribute.

Ego-less Programming

It has been recognized that a major obstacle to better software is our personal attachment to our own programs. Our egos refuse to let us see the deficiencies in our own work, so we close our minds to suggestions from others about how to improve our work. This applies to written documents, designs, etc as well as programs. Remember one main trait of humans is that we make mistakes! Especially in team work, one should strive for “ego-less programming” where each idea is considered on its merits, where any deficiencies are explored for improvements, and the best selection of partial solutions are melded into one team solution.

This is easier said than done. You must stay alert for signs that others (or yourself) are letting ego rule the discussion. Often just pointing it out (tactfully) is enough to bring the discussion around to more productive lines.

Phase 1: Requirements Analysis

In addition to submitting hardcopies of all deliverables, you should ensure that the electronic copies of *all* your deliverables are available in the group subdirectory, with permission for all group members to read them. A **README** file in the subdirectory should explain how to access all the documents, including those under RCS control.

Each team is responsible for issuing *change requests* for the deliverables that it makes use of.

During the project, the group coordinator will act as *Configuration Manager* (CM), with the following responsibilities:

- Upon receipt of a Change Request (CR) from a team, perform the following actions:
 1. Assign a *CR Label* to the change request. This label should have the following form:
Group No./Sending Team/Date/Time
 2. Acknowledge receipt of the CR by reporting the CR label back to the originating team.
 3. Pass the CR (with label) on to the team that produced the document.
 4. Maintain the CR Log, which contains all CRs and the status of each.
- Upon receipt of a new release of a document, perform the following actions:
 1. Update the CR Log according to which CRs have been accounted for in the new release.
 2. Ensure that the new release is available to all group members. (You may place it under RCS control in the group directory for this purpose.)
 3. Maintain a **README** file in the group directory describing which releases are available, and how to access them.
 4. Notify the other teams and the Instructor of the availability of the new release.

Besides the required deliverables mentioned explicitly, each team should release a new version of a document whenever a major set of changes has been completed.

SRS Team

- Familiarize yourself with the SRD Guidelines and sample documents.
- To write the SRD describe each functionality clearly: separate the description of its purpose/rationale, each of the inputs (including those inputs needed for each special case), the decision table (if necessary), and each of the effects.

Use cases and scenarios that document how a user would interact with the utility are valuable tools to focus your discussion. You should make the scenarios very concrete by picking actual inputs, actual files and filenames, etc. (*Coordinate with the IVV Team who need concrete use cases as their test cases.*)

- Prepare a draft copy of your SRS Document to be used for review by the IVV Team. This SRD should be as close to the finished document as possible.
- Present to the class in week 4 an overview of your SRD. Clearly discuss the system model, and provide detailed requirements for one major aspect of the functionality.
- Incorporate feedback into the SRD.
- Deliver the SRD in week 5.

Design Team

- Familiarize yourself with the sample Design Documents.
- Focus on the architectural breakdown into layers, subsystems, and modules: think about desirable client-supplier and peer-to-peer relationships between the modules of the subsystems; design the interfaces of the modules; extend the event traces to consider the internal details of the subsystem.

IVV Team

- Familiarize yourself with the SRD.
- From the use cases — by looking at the range of possible inputs — develop a thorough set of test cases for the functionality of the utility. (*Coordinate with the SRS Team who need concrete use cases for their work.*)
- Develop an automated test script that runs each test case and creates a report of what worked and what did not work.

The script may be written in Unix shell. The shell script loops over all input cases, calls the driver with the input, and uses `diff` to compare the output with the expected output.

- Provide feedback on the draft SRD.

Phase 2: Design

The emphasis shifts to Design. There are two main sub-phases: the **Architectural Design**, which looks at the overall structure of the modules and their interfaces; and the **Detailed Design**, which fleshes out the interfaces between modules and the choice of algorithms and data structures internal to a module.

SRS Team

- Meet with the Design Team to brief them on the SRD. The Design Team must be fully conversant with the required functionality, and any other constraints.
- Process change requests (CRs) received from the CM, and revise the SRD accordingly.
- When a CR has been fulfilled, inform the CM by giving the CR label and number of the SRD release into which the change has been incorporated.

Design Team

- Meet with the SRS Team to be briefed on the SRD.
- Produce the first release of the Architecture. Inform the IVV Team.
- For each module identified in the Architecture, write the first draft of its Interface. Inform the IVV Team.
- Incorporate the feedback into the Architecture and Interfaces.
- Develop a Detailed Design for each module.
- Complete the Architecture and Interfaces. Inform the IVV Team.
- Present an overview of the Architecture and the detailed design of one major module to the class in week 8.
- Incorporate the feedback into the Architecture, Interfaces, and Detailed Design.
- Deliver the Design document incorporating the Architecture, Interfaces, and Detailed Design in week 9.
- If you find any errors or omissions in the current release of the SRD with which you are working, fill out a Change Request (CR) and submit it to the configuration manager. *Do not send the CR to the SRS team.*

IVV Team

- Continuously evaluate the SRD for completeness, consistency, unambiguity, etc.
- When the architectural design stabilises enough to feel confident about a module and its interface (though maybe not its internal details) then you can write a *Module Test Document* that defines a series of tests which will be used to determine if the module conforms to its Module Interface Specification. A driver to perform the module test should also be implemented.
- Begin coding appropriate modules as soon as their design is complete.
- In preparation for each presentation by the Design Team, review the design documents according to the following criteria:
 - Does the design satisfy every requirement in the SRD?
 - Is the design complete, consistent, and unambiguous?
 - Assess the quality of the design.
- Issue any change requests necessary to improve the design or SRD documents.
- Deliver Test Plan Release 1.0 in week 9. This includes the Test Cases, the Module Test Documents, and a *Implementation and Testing Schedule*. You are required to follow this schedule. The Implementation and Testing Schedule states the planned dates by which the implementation and testing of each system module will be completed, as well as the schedule for integration testing, and subsystem testing.

Phase 3: Code and Test

The emphasis shifts to IVV. The preparation for testing should be well underway, but there is much to do to code the modules and actually perform the testing.

SRS Team

- Continue to respond to CR's. However, there should not be any major changes to the SRD at this stage.

Design Team

- Ensure that your design satisfies the SRD fully.
- Respond to CR's from the IVV team.
- *Note:* After week 11, resist the temptation to make major revisions to the design at this stage—you don't have time! Above all, avoid changes to the AD and MIS, as these will have the widest repercussions throughout your design. In you *do* have to change the design, it is vital that you advise all concerned parties right away, especially the IVV team.

IVV Team

- *Reminder:* Your implementation must satisfy the design! An implementation that does not follow the design (at *all* levels: Architecture, Interfaces, and Detailed Design) is not acceptable from an engineering point-of-view.
- Implementation should proceed on units (i.e. individual procedures) and low-level modules as quickly as possible.
- Give feedback via CR's to the Design team on any problems you find.
- Programmers should perform unit testing, and most of the module testing themselves.
- V&V personnel are responsible for subsystem testing and integration testing, including coding any test drivers they need.
- Deliver source code components according to your implementation schedule. The code should follow the *source code standard*.
- Test each module according to your testing schedule and Module Test Documents.
- All modules should have been implemented and tested by week 11.

- Finally, test the entire system according to your Test Plan. Submit the test results. If the system fails any test and time permits, you may revise it and perform the test(s) again. If bugs remain in the system when it is finally delivered in week 13, these must all be documented in the test results document.

The *System Demonstration* is made by the whole group in week 13, at which time all final deliverables are also due.

Epilogue: Feedback and Evaluation

At the conclusion of the project, each group must prepare a **final status report** that clearly states what has been accomplished and what remains to be done. This report should be very specific, indicating any unsatisfied requirements, unimplemented design components, and incomplete testing. Reports couched only in general terms will not be acceptable.

Each group member must hand in an **individuals' report and assessment** detailing the activity undertaken by all members of the group in relation to the project. Based on the report, each student must assess their own contribution and the contribution of each other member of the group, and provide a reasoned report for their assessment.