

Information Packet

P465-6 & P565-6

Software Engineering
for Information Systems I & II

2008-9



Department of Computer Science
School of Informatics
Indiana University

P465–6/P565–6 – Information Systems

Table of Contents

<i>Section</i>	<i>Description</i>	<i>Page</i>
Instructors	Office, hours, phone, etc.	1
Course Description		1
Texts and References		1
Fall Lecture and Task Schedule		3
Spring Lecture and Task Schedule		5
Grading	How grades will be computed	6
Professional Conduct		7
General Writing Issues	Highly relevant to everyone!	7
Team Project Description		9
Project Schedule		10
Detailed Specifications of Milestones:		
Milestone 0	Project Proposal	15
Milestone 1	Feasibility study	18
Requirements Reviews		20
Milestone 2	Requirements Specification, Project Plan, and Quality Assurance Plan	21
Milestone 3	Prototype	25
Milestone 4	Preliminary Design	28
Project Presentation		31
Detailed Design Review		32
Milestone 5	Detailed Design and Test Plan	33
Milestone 6	Implementation	35
Milestone 7	Testing	36
Milestone 8	Installation	37
Post Partum Presentation		39
Reporting: Forms and Logs		40
Supervisor evaluation form		41
Team member evaluation form		43

This information packet should answer many of your questions about the Information Systems courses. You are expected to read and understand the entire document.

P465–6/P565–6 – Information Systems

Section: (P465) 8292 (P565) 8296
Lecture: 16:00–17:15 TR *Room:* Lindley 102
Special meetings: 10:10–11:00 F *Room:* Lindley 102
Web page: www.cs.indiana.edu/classes/p465/index.html
NFS repository: [/nfs/paca/u/edrbtn/InfoSys](http://nfs/paca/u/edrbtn/InfoSys)

	Instructor	Associate Instructor
	Edward Robertson	Sofia Brenes-Barahona
<i>Office:</i>	Lindley 401D	Lindley 401A
<i>Phone:</i>	855-4954	855-4318
<i>E-mail:</i>	edrbtn@indiana.edu	sbrenesb@cs.indiana.edu

Catalogue Descriptions

P465–P466 Software Engineering for Information Systems I–II (3–3 cr.) N&M. P: C335, C343; B461 concurrently. Analysis, design and implementation of information systems. Project specification. Data modeling. Software design methodologies. Software quality assurance. Supervised team development of a real system for a real client. I Sem., II Sem. Credit not given for both P465-6 and P565-6.

P565–P566 Software Engineering I–II (3–3 cr.) P: C343, B461 previously or B561 concurrently. Analysis, design and implementation of software systems. Requirements specification: data and process modeling. Software design methodologies. Software quality assurance: testing and verification. Software development processes. Credit not given for both P465–P466 and P565–P566.

Texts & References

The following texts all have material valuable to this course. Required books are marked \star and strongly recommended ones \diamond . Items marked \dagger are on reserve in the Swain Hall Library.

\diamond BCN *Conceptual Database Design: An Entity-Relationship Approach*. C. Batini, S. Ceri, and S.B. Navathe. Benjamin/Cummings, Redwood City, CA, 1992. The team should have this on hand during data modeling.

$\diamond\dagger$ BH *Software Engineering Fundamentals*. Ali Behforooz and Frederick J. Hudson. Oxford University Press, New York, NY, 1996.

DavA *201 Principles of Software Development*. Alan M. Davis. McGraw-Hill. 1995. Just what it says: 201 one liners. The Table of Contents alone is a good software engineering course.

\dagger Dav *Tools and Techniques for Structured Systems Analysis and Design*. William S. Davis. Addison-Wesley, Reading, MA, 1983. Although old, some of the modules are still on-target.

\diamond G UW *Database Systems: the Complete Book*. Hector Garcia-Molina, Jeffery Ullman, and Jennifer Widom. Prentice-Hall, 2002.

Jo *Software Engineering*. Gregory W. Jones. Wiley, 1990. Good on modularization.

- KingS “Cost Benefit Analysis in Information Systems Development and Operation.” J. King and E. Schrems. *ACM Computing Surveys*, Vol. 10, No. 1, March 1978, pp. 19-34.
- KS *Database System Concepts*. Henry F. Korth and Abraham Silberschatz. McGraw Hill.
- ON *Database: Principles, Programming, and Performance*. Patrick O’Neil. Morgan Kaufmann, 1994. Only text with a good discussion of embedded database programming.
- Pet *Petri Net Theory and the Modeling of Systems*. James L. Peterson. Prentice-Hall. 1981.
- † Pr *Software Engineering. Sixth Edition*. Roger S. Pressman. McGraw Hill, 2005. Fourth and fifth additions are adequate.
- ◇ Ram *Database Management Systems. Second Edition*. Raghuram Ramakrishnan and Johannes Gehrke. McGraw Hill.
- * R *Running the River: A Survival Guide for the Waterfall Model*. Edward L. Robertson. Course packet available locally.
- * R2 *Information Modeling*. Edward L. Robertson. Lecture notes available locally.
- Sch *Software Engineering*. Stephen R. Schack, Aksen Assoc., Homewood, IL, 1990.
- † Sv *Software Engineering. Sixth Edition*. I. Sommerville. Addison-Wesley, Reading, MA, 2001.
- † St *Software Engineering with Systems Analysis and Design*. Donald V. Steward. Brooks/Cole, Monterey, CA, 1987.
- ◇ SW *The Elements of Style. Second Edition*. Strunk and White. Macmillan Publishing, New York, 1972. A superb aid for writing tasks, recalling the injunction “English is your most important professional tool, use it with precision.”
- † vM *Software Engineering: Methods and Management*. Anneliese von Mayrhauser. Academic Press, 1990.

Many valuable books, especially documentation for commercial products, are available online at <http://library.books24x7.com/topics.asp> .

P465/P565–Lecture and Task Schedule

Classes will be held every Tuesday and Thursday for lecture and discussion. Fridays will be used for tutorials and help sessions. Fridays will occasionally feature “guest lecturers” from industry, who will talk about information systems and software engineering in their organization. Some Fridays will be free for team meetings (more so in the Spring).

<i>Week</i>	<i>Date</i>	<i>Day</i>	<i>Reading</i>	<i>Topic</i>
1	Sept 2	T	R I-IV	Introduction to the course. Project organization. Project Proposal.
	Sept 4	R	BH 1	Information systems engineering.
	Sept 5	F		Career preparation
2	Sept 9	T	BH 2, Appx A; KingS	Systems analysis. Benefit/cost analysis.
	Sept 11	R	R v; St 12	Client perspective, interviewing.
	Sept 12	F		Career preparation
3	Sept 16 ¹	T	R vi	Feasibility Study document.
	Sept 18	R	R vii; BH 3; St 6,7	Project scheduling, management, and control.
	Sept 19	F		Managing large document development.
4	Sept 23	T	R viii; BH 4; Pr 7; Sv 5	Requirements Specification document.
	Sept 25	R	BH 5	Enterprise modeling and specification.
5	Sept 30 ² -Oct 2	TR	R2	Information modeling. ER models.
	Oct 3	F		Commercial database packages.
6	Oct 7-9	TR	R2; BCN	Information modeling.
	Oct 10	F		Commercial database packages.
7	Oct 14	T	Dav D; BH 5.4	Data Flow Diagrams
	Oct 16	R	R ix	Development platforms. Prototyping.
	Oct 17	F		Commercial database packages.
8	Oct 21	T		Statecharts.
	Oct 23	R	Pr 26	Quality assurance I.
	Oct 24	F		Commercial database packages.

Deliverables:

¹ Project Proposal due.

² Feasibility Study due.

P465/P565–Lecture and Task Schedule

Note that classes with underscored topics require presentations by project teams.

<i>Week</i>	<i>Date</i>	<i>Day</i>	<i>Reading</i>	<i>Topic</i>
9	Oct 28-31 ³	T-F		<u>Requirements Specification reviews.</u>
10	Nov 4	T	R x; Pr 9; St 4; BH 7	Preliminary Design document. Design methodologies.
	Nov 6 ⁴	R	G UW 3.6	Relational design.
11	Nov 11-13	TR		Special topics
12	Nov 18 ⁵ -Nov 20	TR		Human factors. User interface design.
13	Nov 25 ⁶	T		Test plans.
	Nov 27	R		Thanksgiving recess.
14	Dec 2	T		Exam.
	Dec 4 ⁷	R		<u>Presentations.</u>
15	Dec 9-12	T-F		<u>Presentations.</u>
16	Dec 11, 14:45	T		<u>Presentations.</u>

Deliverables:

- ³ Requirements Specification *draft* due to supervisor.
- ⁴ Requirements Specification due.
- ⁵ Prototype due.
- ⁶ Preliminary Design *draft* due to supervisor.
- ⁷ Preliminary Design due.

P466/P566–Lecture and Task Schedule

<i>Week</i>	<i>Date</i>	<i>Day</i>	<i>Reading</i>	<i>Topic</i>
1	Jan 13	T	BH 9	Detailed Design document. Configuration management. Program documentation.
	Jan 15	R	R XI; PR 14	Testing I.
2	Jan 20-23	T-F		<u>Design reviews.</u>
3	Jan 27-30 ⁸	T-F		<u>Design reviews.</u>
4	Feb 3	T	BH 10,11	Testing II.
	Feb 5 ⁹	R	R XII	Documentation. Installation. User training.
5	Feb 10-12	TR	BH 14	Software metrics. Cost estimation.
6	Feb 17-19	TR	BH 18; Pr 28	Quality assurance II – verification.
7	Feb 24	T	BH 13; vM 11	Maintenance.
	Feb 26	R		Distribution configuration.
8	Mar 3 ¹⁰⁻⁵	TR		Information theory.
9	Mar 10	T	BH 19	Real-time systems engineering.
	Mar 12	R		First exam.
*	Mar 16-20	MF		Spring break.
10	Mar 24-26	TR		Special topics.
11	Mar 31 ¹¹ -Apr 2	TR	BH 5.6; Pet	Specifying concurrency – Petri nets.
12	Apr 7-9	TR	BH 18	Quality assurance III.
13	Apr 14 ¹²	T		
	Apr 16	R		Second exam.
14	Apr 21 ¹³⁻²⁴	T-F		<u>Post Partum presentations.</u>
15	Apr 28-May 1	T-F		<u>Post Partum presentations.</u>
16	scheduled exam			<u>Post Partum presentations.</u>

Deliverables:

- ⁸ Detailed Design *draft* due to supervisor.
- ⁹ Detailed Design due.
- ¹⁰ Implementation complete.
- ¹¹ Testing completed.
- ¹² Installation completed and final project documentation draft to supervisor.
- ¹³ Final project documentation due.

Grading

Two Semester Grading

This two semester course sequence (P465-6 and P565-6 collectively) is taught and graded as one single unit. That means that you will receive an “**R**” grade for the Fall semester. The “**R**” will be automatically removed when you receive your Spring grade.

Under circumstances of demonstrated and substantial hardship, the “**R**” may be removed without completing the Spring semester. If you do not take (or complete) the Spring semester course and cannot demonstrate substantial hardship, the “**R**” will remain on your record indefinitely and the Fall course will not count toward your degree.

Weightings

The following table indicates the planned weighting for each of the graded parts of the course.

<i>Unit</i>	<i>Approximate Weight ($\pm 10\%$)</i>
Exams	20%
Assignments	10%
Project	70%

Project Grade

The project grade consists of two components, a base grade and a modifier. The base grade for a project is proportional to the quality of the *software product* developed by the team and is modified according to the individual participation of the students within a team. The software product consists of the entire system, including software and supporting documentation. Central to the evaluation of the product is the fact that the system will be used to satisfy a real client’s need over a period of time. The interfaces should be designed to make the software easy to use and effective in its purpose. The documentation should be complete, correct and of professional quality. This portion of the project’s grade is based on an evaluation of documents handed in, code (by inspection), oral presentations, and software demonstrations.

There is no set scale or weighting for the individual milestones. They are there to give you concrete intermediate objectives, valuable feedback, and metrics for evaluating your progress. An essential factor in evaluating an individual milestone is how that milestone contributes to the overall success of the project (and of course that may not be apparent for many months). For example, a team may have *appeared* to have developed a highly successful prototype, but developing that prototype provided insufficient tool experience for making good design and implementation decisions. Thus the initial good evaluation of the prototype is revised, because the prototype was in fact inadequate.

Each individual’s grade modifier is based, in part, on input from the supervisor and from other team members. The performance and participation of students in projects will be observed by the team’s supervisor and the instructors. Also, at the end of each semester, you will be asked to evaluate the relative performance of the members of your team, including yourself, by filling the form shown on page 43.

Finally, because the success of a team depends upon the contributions of each and every team member, a last-resort mechanism has been established to remove a member who does not participate in and contribute to his or her team project. This mechanism is detailed on page 12 toward the end of subsection “Supervisors and Supervisor Responsibilities.”

Assignments

The assignments include a variety of tasks, primarily practice in evaluation and modeling techniques. Traditionally, students taking the Information Systems courses underestimate the value of the assignments. It has been observed that students who do a poor job on the assignments have their final grades affected. You are strongly advised to work on the assignments.

Note that a draft of the E-R model is required as part of an assignment. As long as a reasonable effort has been made, the draft E-R Model will be graded very leniently; the purpose of requiring it is to give you good feedback on your specification.

Professional Conduct

As with other aspects of professionalism in this course, you are expected to abide by the proper standards of professional ethics and personal conduct. This includes the usual Computer Science Department standards on acknowledgment of joint work and other aspects of the Departmental Statement on Academic integrity (available on the World Wide Web in www.cs.indiana.edu/dept/integrity.html) and the University policy (*Indiana University Code of Student Ethics*, August 15, 1990, <http://dsa.indiana.edu/Code/index.html>). It also prohibits intentional misrepresentation and other misconduct.

General Writing Issues

Written communication is important in this course, as it is in the profession in general. Remember that software documentation has unique and important characteristics:

1. Technical documentation is often the result of group authorship, thus it requires preparatory planning and final melding.
2. Specificity and organization are often more important than flow, hence technical documentation is often organized around lists and tables rather than paragraphs.
3. Documentation is often the reader’s only source of information on the particular subject or product, hence it must be thorough and complete.
4. Documentation is often used to answer a specific question, hence it should facilitate finding a specific piece of information (“navigation”).
5. Documentation must bridge from general specifications to particulars of implementation and operation, hence it must make abstract concepts concrete and make concrete facts fit generalized concepts.
6. Documentation can be presented in many forms: reference manuals, tutorials, quick reference guides, *etc.* It can be represented in many media: online via html, MS Help files, or just plain text and even on paper. It is important to choose the right medium and even more important to write to fit the medium.

You should also be aware of the services provided by Writing Tutorial Services(WTS). WTS is located in Ballantine Hall 206 and is open 10-8 Monday through Thursday and 10-5 Friday. WTS tutors are available specifically for foreign students and for the IW component of this course. Call 855-6738 to schedule an appointment with a tutor. See www.indiana.edu/~wts/wts for further information on WTS.

Team Project Description

The Information Systems Course Project is unusual in that it covers two semesters, involves teams of students rather than individuals, and is intended to produce a real software product. This might be your first experience in working with others to produce a piece of software. Unless you have professional experience, it might also be your first experience in producing a real-world software product for a real user. Assignments that you have had in previous programming classes had clear goals and well-defined requirements; real-world projects require you to elucidate the goals and articulate the requirements. According to feedback from our graduates and from the people that hire them, the Information Systems project provides valuable experience; you will benefit from the sometimes painful lessons to be learned in this course.

Milestone Summary

The project is broken into objectives called “Milestones”. These milestones are summarized here and each is detailed in a subsequent section. Note that this *Information Packet* is only a high-level specification of these milestones. That is, it describes in broad terms what is required but not how the various objectives are to be realized. Suggestions about realizing the milestones are given in class, *Running the River*, and the other various references.

The first order of business in the course is for you, individually, to find a potential project and prepare a Project Proposal. Our projects are not dreamed up by the instructors, but rather come from the needs of the local academic and business communities. We have found that our students are very efficient at ferreting out likely projects. Your first assignment for the project portion of the course is to find and propose a project (see Milestone 0 below). The information in the Proposals will be used in a preliminary screening, from which approximately half of the projects will be selected for the next phase. Teams of two people will be formed.

Each team must then do a detailed Feasibility Study to examine the costs and benefits of their project and to determine the appropriateness of the project to the course (see Milestone 1). A final selection of projects will be made on the basis of the Feasibility Studies. Once again, approximately half the projects will be selected and larger teams, typically with four members, will be formed.

The project team for each selected project will then do a detailed requirements analysis, producing a Requirements Specification document and a Project Plan (see Milestone 2). The Requirements Specification describes exactly what the client needs, and the Project Plan indicates how the product will be created in the time allotted. The Requirements Review, an important step in preparing the Requirements Specification, provides feedback on your ideas, particularly of a draft of the Entity-Relationship Model.

In parallel with completing the Requirements Specification, you should be working on the Prototype (Milestone 3). The entire team should begin learning the project tools while working on the Prototype.

The final phase of the project for the Fall semester will be to produce a Preliminary

Design document and a Test Plan (Milestone 4). The Preliminary Design specifies how the finished product will look to the client, and the Test Plan indicates how the finished product will be verified and validated.

The remaining phases – Detailed Design (Milestone 5), Implementation (Milestone 6), Testing (Milestone 7), and Installation (Milestone 8) – will be completed in the second semester, but you should familiarize yourself with their goals now.

Project Schedule

	<i>Date Due</i>	<i>Supporting Reading</i>	<i>Description Goals and Activities</i>
Milestone 0	Sept 16	R I-IV; My 1; St 10	Project Proposal. Life cycle, program products, reliability. Interviewing.
Milestone 1	Sept 30	R v-VI; BH 1,2, Appx A; St 12; Pr 11.2	Feasibility Study. Systems analysis. Software management, Benefit/cost analysis. Team organization.
Requirements Reviews	Oct 28- Oct 31	RVG; BCN; Ram 2	Walkthrough of Requirements Specification. Draft E-R model. Peer feedback on requirements.
Milestone 2	Oct 31 [†] , Nov 6 [‡]	R VII-VIII; My 2; RVG; KS 2; Ram 15; St 11, 13	Requirements Specification, Project Plan, Quality Assurance Plan. Data flow diagrams, E-R model. Project scheduling and management.
Milestone 3	Nov 18	My 3; Sv 6; St 9	Prototype. User interfaces. Software evaluation. Tool experience.
Milestone 4	Nov 25 [†] , Dec 4 [‡]	R IX; BH 5; My 3, 4, 7, 8.2; St 15	Preliminary Design and Coding Standards. File organization. Pseudo-code, HIPO, Warnier-Orr diagrams. The user interface, report design. Module, interface, and documentation standards.
Presentations	Dec 4- Dec 11		Project Presentations. Project goals and general approach toward achieving these goals. Specifications, ER model, and user interface. <i>Client oriented.</i>
Design Reviews	Jan 20- Jan 30	Sv 14; R x	Design Walkthroughs. Peer feedback on designs.
Milestone 5	Feb 3 [†] , Feb 5 [‡]	My 5; R xi	Detailed Design and Test Plans. Software specification and design. System flowcharts. Relational design issues.
Milestone 6	Mar 3	My 6; Pr 12	Implementation. Programming style, readability. Documentation. Code walkthroughs. Unit testing.
Milestone 7	Mar 31	R xi; My 7; BH 10,11	Testing. Integration testing. Debugging.
Milestone 8	Apr 14 [†] , Apr 21 [‡]	R xii	Installation and Documentation. Post mortem.

† – draft due to supervisor.

‡ – final copy due to instructors.

Detailed descriptions of the reports to be produced for each milestone are given in the following pages. The deadlines for the milestones other than Milestones 0, 1, 4 and 8 are somewhat flexible and can be negotiated with your Supervisor. For Milestones 0, 1, 4 and 8, a penalty will be assessed for late reports. Milestone 0 must be turned in to Prof. Robertson or placed in his mailbox located in the CS Department's main office no later than 4:00 PM, Tuesday, Sept. 16. All other Milestone documents are to be turned in to your Supervisor on the date indicated.

Project Planning

In order to complete the project successfully, it is *necessary* to work on several tasks at the same time. The steps to the next milestone often have considerable lead-time – for contacting a client, reviewing a document, or simply careful deliberation. By analogy, a good cook often has several things in the oven and on the stove; the first course, the salad, is the last prepared because it has the shortest lead time. All of this requires that the steps be planned and that the plan be monitored.

The Project Schedule should help you get started on a Project Plan. However, your plan will require much more detail.

Project Teams

Beginning in approximately the third week of classes, you will be assigned to a *Project Team*. Team membership is assigned by the Instructors and Supervisors. Milestone 0 is done individually, Milestone 1 by teams of two, and Milestone 2 and beyond by teams of four or five. Team assignments are made with an eye toward providing each team with the necessary balance of skills and capabilities.

From time to time, it might be necessary to reassign an individual from one team to another, based on changing estimates of manpower needs and team abilities. Very rarely a team member must be “fired” using a mechanism described below.

Supervisors and Supervisor Responsibilities

Each team will have a *supervisor*. The Supervisors are graduate students taking the Software Engineering Management course (B665-6). They have already taken P565-6 and will provide valuable, and sometimes invaluable, help to the teams.

Although a supervisor will often give a team technical guidance, the supervisor's one specific responsibility is assisting the team with process. To this end, each team must hold a “supervisor's meeting”, where process-related matters are reviewed, every week. The title of these meetings does not come from the fact that the supervisor is in charge – in fact, the team is still responsible for the conduct of the supervisor meetings – but because they are the one time each week when the team is required to meet with its supervisor. The most important process matter is project planning, as reflected in the **Team Log** section below; the log thus produced is the primary deliverable of this meeting. Of course technical matters are often covered at the supervisor's meeting; it is a good occasion to hold a “walk-through” of some topic. Furthermore, the team often will need to meet more than once a week – perhaps with the supervisor, with clients, with instructors, or with other resource people.

The Supervisor has a responsibility analogous to a first line manager in a professional

situation. Of course, the Supervisor cannot literally hire and fire students, or suggest raises or salary reductions. However the Supervisor does have at least two ways to encourage performance by the team members.

First, the Supervisor will periodically evaluate each member of the team. This evaluation will be given heavy weight in determining each individual's grade for the project portion of the course. Since the project is a very significant part of the course grade, this should provide incentive for all team members to perform.

A second, more drastic mechanism is available to the Supervisor if a student is unable or unwilling to work appropriately as a team member. The Supervisor can recommend that the Instructors "fire" that student. The Instructors will carefully evaluate the student's performance in the team context and, if warranted, remove the student from his/her team, in effect "firing" him/her from the project. A "fired" student will then be given an appropriate individual project assignment by the Instructors. The grade for the replacement project will be reduced by one letter grade so that the maximum grade project is a B. A "fired" student can try to get "hired" by another Supervisor, but that student will have to make a very good case to succeed. Moreover, copies of any correspondence relating to firing will be placed in the student's departmental files.

Quality Assurance Monitor

In addition to serving as Supervisors, students in the Software Engineering Management course will also be assigned as Quality Assurance Monitors. Of course the roles of Supervisor and QA Monitor are assigned to different individuals.

The QA Monitor's role is limited but essential, being concerned with procedure rather than content. The QA Monitor must sign-off to verify that the Team has followed its QA Plan.

The assignment of QA Monitors to teams occurs toward the end of the first semester.

Team Log

The team must maintain a Team Log for the life of the project. The Team Log must record all important team decisions, task assignments (such as 'Joe agreed to provide a debugged copy of module A by Feb 15'), task completions and other task status facts, and design or process changes. All this information must be recorded in a timely manner. The Log should also record role assignments and may also include notes from team meetings.

The Team Log must be a "write-once" document. Originally the Team Log was a permanently bound volume (hence we still refer to the "Log Book"), but now it is more common to use electronic media. But even though the log is kept in a computer file, it is important that it retain its "write-once" nature. When the log was kept on paper, the "write-once" characteristic was enforced simply by keeping the log in ink, not allowing erasures. In a computer file, it can be enforced by using a file that is append-only. The reason for these constraints is that we wish to require that changes in the log be explicit. Just as a bank reflects an error in a deposit by adding a new journal entry rather than changing the original deposit, a correction or change in a log requires noting the incorrect statement and presenting the correct one.

In order to insure the “write once” characteristic, each team is required to email their log entry to their Supervisor each week. It is required that the team have one formal meeting a week during which process matters are dealt with – in particular, tasks are assigned and task status is reviewed. These process matters are the heart of the log entry.

You are asked to use a template for your log entry. This is easier for you because it indicates exactly what is required. This is easier for the Supervisors and Instructors because we know exactly what to look for. Templates are available in the files `team-log-template.*` in the course’s NFS repository. The team secretary should simply edit this file and submit it by email each week. In fact, it is a bit easier to start with a copy of last week’s log entry, since much of it carries forward.

Most of the fields in the template are self-evident, but some deserve a few words of explanation:

- **Recording Date** The date on which the secretary edited the log entry.
- **Due Date** The date given in this packet on which the milestone is due (see the Project Schedule on the following page).
- **Group Target Date** The team should set their own internal due date a few days before the milestone is to be turned in. Record that date here.
- **Deliverables Due** A list of those items to be turned in.

In certain circumstances (*e.g.* when working on the Prototype and Preliminary Design), you may need to have two “Current Milestone” entries.

At each meeting when new task assignments are made, those assignments should be logged where indicated. The next week, these assignments should be move down to the next area, as carry-forward assignments. Tasks remain as carry-forward until after they have been marked “completed” in the status field. A completed task is then removed from the next log entry.

To reiterate, the purpose of the log is to record in a consistent manner and place: what tasks must be done, when they must be begun and finished, who is responsible for accomplishing them, and where they stand with respect to their completion.

Team Roles

Roles are associated with ongoing or regularly recurring responsibilities. In some sense a role is just a continuing task, but roles are typically associated with process matters while tasks are more likely to work toward products.

Of course there are many more roles than team members, which implies that one team member may be assigned several roles. For example, the various liaison roles may be combined and assigned to one person. Sometimes a role is done collectively by the entire team and thus assigned to the team; only a few roles, such as task assignment, are suitable for collective responsibility. In any case, the party or parties responsible for a role must be recorded in the log.

The project roles are:

- *Secretary*: maintains the Team Log.
- *Supervisor Liaison*.
- *Instructor Liaison*.

- *Client Liaison*. It is important that all team members meet with client for information gathering, presentations, and related activities. But unless there is one person who schedules meetings, makes commitments, *etc.*, the client (and the team) is likely to get confused.
- *Task Assigner*: schedules and assigns tasks based on deliverables and activities identified by the team; major input to the Team Log.
- *Task Monitor*: tracks task status for the Team Log.
- *Convener*: calls meetings, organizes intra-team communication.
- *Librarian*: maintains a repository of documents and programs; monitors configuration. Particularly relevant in the Spring semester.
- *Guru*: toolkit/language/platform specialist. The guru should teach tool(s) to other team members. Occasionally this role is split to address multiple tools.

Project Grading

The project for this course actually involves the construction of a software product: a system of software and supporting documentation which will be used to satisfy a real need over a period of time. We will therefore expect careful design to protect the user from making mistakes and to allow future modifications. The interface should be designed to make the software easy to use and effective in its purpose. The documentation should be complete, correct and of professional quality. Finally, of course, the project will be evaluated as to how well it serves its original purpose.

Milestone 0

Project Proposal

The goals for this milestone are two-fold: (1) to search out suitable projects for this term, and (2) to evaluate your writing skills and capabilities. There is one single objective that meets both these goals: the preparation of a report which describes a *potential* project in sufficient detail that a decision about its viability may be made. Note that this is a *proposal* - you should not make a commitment or a final decision on whether to pursue the project, although of course you should make recommendations. To commit to a project without a thorough feasibility study would be dead wrong.

Since some of you may not find suitable projects to investigate, there are two alternative objectives that meet goal (2) only. These two alternatives, a Followup Report and a Writing Exercise, are discussed below. You are, of course, urged to attempt the assignment as given.

Where to Look

There are many potential sources for projects in and around Bloomington. The most accessible sources are probably other academic departments and administrative units. These are especially convenient because frequently student computer accounts can be obtained on the target equipment for the development stage. Local businesses, large and small, are also good sources for projects. Another possibility for projects is local government. In any case, it will be easier to approach a possible client if you already have a contact, either directly or indirectly.

What to Tell the Potential Client

You should tell the client that you are soliciting information systems projects for a Computer Science course. The client can expect to receive a professionally written software product. A team of students, supervised by faculty and advanced graduate students, will analyze, design, implement, test, and install software for an information system.

Although there is no cost for the services of developers or supervisors, the client is not without obligation. In particular, the client will be expected to provide: (i) time of individuals in the client's organization, who will help specify and evaluate the product; (ii) information about the organization and its operation, including policy and procedure guides, manuals for current systems, *etc.*; and (iii) access to any facilities that will be necessary for the development of the product. Members of the client organization must spend substantial time providing the input necessary to specify the goals and requirements of the project. Input will be required from managers as well as users of the product. The client is expected to read and sign-off on the Requirements Specification. Access might include after-hours admission to the client's premises or the purchase of additional software or hardware required for the successful completion of the project.

In order to avoid misunderstandings, the client should be aware of both what the project is and what it is not. It is not a business practice analysis, although such analysis occurs during the specification stage. It is not simply an exercise to select and install a software package, although it may be necessary to purchase software to support the project. It is not

the development of static web pages, although a web interface to the information system is increasingly common. The client must understand that the project is to be completed within two semesters and *there will be no support after the project is completed*. In no case will the Information System project team do data entry for the client (except for limited test data). However, data conversion (bulk extraction of data from existing systems for insertion into the new system) is commonly part of a project's scope. An Information Systems project can be a good opportunity for a client to experiment with or develop a prototype.

A four page handout intended to help you explain this project to the client is available in the file `clienthandout.ps` in the course directory.

Suitability

For the purposes of this assignment, you only need to find a project which can be proposed; however, most students would like to see their proposed projects designed and implemented. Your project must meet the following criteria before it will be approved.

- *Time frame.* We have a fixed time frame for designing and implementing the final product. If the client absolutely must have the working program before next April, we will not be able to do the project.
- *Content.* This is a database course as much as it is a software engineering course, so we require at least some database component for the project. This does not mean that the database has to be the primary component. However, this “database requirement” excludes purely computational projects. Projects which only require porting a system to a new computer are also not appropriate (unless considerable redesign of the software is required).
- *Size.* The size of the project is very important. It should not be so large that it is beyond the abilities of a team of four or five students, but it also should not be too small. Project size is often difficult to judge, so don't eliminate a possible project unless it is obviously too large or too small.
- *Location.* It is typically required that the project be based in Bloomington. Out-of-town projects will be approved only in special circumstances. However, you are free to do a Project Proposal on such a project if you have no local alternative.

The Report

The written report to be turned in should cover several important points. The list below should be considered as a guideline rather than a formula. The entire report should be two to three (typeset, double-spaced) pages.

1. *Client Information.* Who is the primary contact for the project? Who is motivated to support the project? Who has authority to allocate resources for the project? Discuss this in the context of the larger client organization, along with a brief description of the organization itself.
2. *Project goals.* This is the heart of the document. Briefly (one to three paragraphs should suffice) describe what the project is, what value it will be to the client, how it will be used. How are these goals currently being met, if they are?
3. *Project Scope.* Where does the proposed system fit in the client organization? What

other systems will interact with the proposed system?

4. *Computing facilities.* What computing hardware and software does the client now have? Can other available equipment (for example, campus educational computers) be used for project development? Will the client be willing to acquire additional facilities (*e.g.* a hard disk, a database package) if such facilities are required for the project. Note that you *should not*, at this point, specify additional hardware or software. That will be done during the Feasibility Study.

Try to be as clear as you can about the purpose of the proposed software and how it will be used.

Followup Report

This section is only relevant if you have been specifically asked to do a Followup Report by the Instructors. If this is the case, you will have been given the name of a client from a previous Information Systems project. In this case, your report should have different content. Your primary concern should be evaluating the success of the previous project:

- If it is being used, what are its strengths and weaknesses? Are there major extensions and enhancements that could provide another project?
- If it is not being used, why not? Are the problems with the previous system remediable with a new project?

A more detailed description of a Followup Report can be found in the file `m0-followup` in the course repository.

Milestone 1

Feasibility Study

The goal of the Feasibility Study is to provide enough information to decide whether or not the project should be pursued. In a commercial enterprise, a benefit/cost analysis is the most important deciding factor. In our case, certain other factors are more important, although a limited cost/benefit analysis should still be done.

The most important part of the analysis phase (Feasibility Study and Requirements Specification milestones) is to find out what the client wants. If there is an existing (possibly manual) system which approximates the desired product, you should try to understand the system. Read any available documentation and manuals and talk to the users of the system. (The actual users frequently know more about what the system should do than their managers.) If there is no existing system, you must carefully interview the appropriate people. *You must plan your interviews in advance.* See *Running the River* for suggestions on how to conduct an effective interview and other information gathering techniques.

Once you understand what is wanted, you should try to come up with several alternative solutions that will satisfy the client's needs. Typically you should consider "bare bones" and "full blown" versions of a customized software product. You might want to consider among several implementation "platforms", as well as customizing a special purpose software package, if appropriate. It is possible that the user's needs can be met by an off-the-shelf software package, or even by a system that does not involve a computer at all. Such possibilities must be evaluated without prejudice. If the best solution is simply a box of file cards, then a Feasibility Study that recommends otherwise is simply wrong.

The Report

The Feasibility Study should consist of five to ten typeset pages. Two things that you *must* do are identify the *key people* in the client's organization and describe the *goals and objectives* of the proposed system.

In a commercial environment, the key questions to be answered by the Feasibility Study are: "Is there a feasible solution to the problem?," and "Is the cost of the new system worth it?" Our case is somewhat different. The cost of development is likely to be low; although the client must be aware that you will need the time of some key personnel during specification and design, staff assistance for testing and evaluation, and access to the client's facilities for installation and systems testing – for the project to succeed, the client must be willing to commit these necessary resources to the team. One of the key criteria for deciding to go ahead with the project will be the commitment shown by the client toward the success of the project.

The report should also answer the following questions: *What is the problem?* – This requires a description of the current system and its associated shortcomings. *What is the scope of the problem?* – Place the problem (and its potential solutions) in context within the client's organization, including interaction with other systems, both computerized and manual. *Who will be impacted by the proposed system?* – This includes users, both inside and

outside the client organization, managers, and others indirectly effected by the system (*e.g.* all employees are effected by a payroll system). *What are the alternative solutions?* – This can be answered by describing one or more proposed new systems which would eliminate the shortcomings of the current system. *What are the benefits and costs of the proposed solutions?* – The real costs for the various solutions are the hardware and software required for their implementations (you will also be asked, below, to estimate the development cost of your system as if it were charged at professional rates). You may need to call local computer stores to get software and hardware quotes. Also, although benefits are usually less tangible than costs, you should be as specific as possible in describing the benefits. *What is the likelihood of the project's success?* – This can be answered by giving an assessment of the level of client's commitment to the project and degree of computer sophistication. If necessary, is the client able and willing to commit financial resources to the project? Are resources available so that the project team can develop, test, and install a new system? Are the client and his/her staff easily accessible for questions and interviews? *What is the recommended solution?* – If several solutions were proposed, then your report should specify which one is the best and why – remember that the best alternative might be to do nothing. Also, discuss your assessment of the general suitability of the proposal for a project. You should not force your analysis to recommend a project if that is not, in fact, the best solution for the client's needs. You will not be penalized if you recommend against the continuation of a project. Neither you nor your client will benefit by an inappropriate attempt to continue a project.

In preparing the report, you must do enough systems analysis to understand the client's problem. The report itself gives you the opportunity to clearly state the problem and demonstrate how well you did the systems analysis.

You are not limited to answering the questions mentioned above and should include others as needed. One of the main functions of this report is to enable the instructors and supervisors to select good projects with a high probability of success; so, any information that might be helpful in making this decision should be included in your report.

You may organize the report in any manner that you deem effective and appropriate, however, try to include a table of contents. Guidelines for the contents of a Feasibility Study document can be found in the class notes, *Running the River*, and in reference materials.

Cost Estimate

As an appendix to the Feasibility Study report you should attempt to estimate the development cost of your proposed system. In the "real world" these costs are, of course, the dominant factor in a benefit-cost analysis. However, our purpose is just to gain experience, since experience is most important in cost estimation.

Therefore, you should estimate the the hours your project will take. This should be done by estimating each milestone separately, further breaking down each milestone as much as possible. If you wish to translate your estimate into dollars, the current rate for administrative systems development at UITS is roughly \$50 per hour, while commercial rates can easily be twice that and more.

The paper by King and Schrems [KingS] discusses benefit–cost analysis.

Requirements Specifications Reviews

Reviews or walkthroughs are important steps in producing a system that is effective and efficient. They consist of oral presentations to an audience of “technical peers.”

There are two reasons for a team to present its work to an external audience of technical peers. The first is that, in preparing the presentation, the team is forced to once again review its analysis and design and clarify its decisions. The second is that the external peers, without the unspoken presumptions held by the team, may be free to see flaws or improvements that the team has overlooked.

Plan for about 30 minutes of presentation, with an additional 15 minutes for questions and comments.¹⁴The content of this presentation is exactly the work that your team is doing on the Requirements Specification Milestone (described in the following pages). In particular, you should not develop any special content for walkthroughs, although you may select important material and format it for the presentation. Walkthroughs should exactly reflect your team’s thinking on its current milestone. Spend a few minutes on the E-R model, but do not spend the entire time since you will be handing in a draft E-R, as discussed below. Most important are the functional requirements, which should be itemized and discussed.

Remember that you are presenting to your *technical* peers. You should not be afraid to use technical concepts, although you should avoid jargon.

Draft Entity-Relationship Model

The system design discipline appropriate for almost all Information Systems projects is information driven. Therefore, it is essential that the information model be as good as possible. To insure this, your team should hand-in a *draft* of its Entity-Relationship model diagram at its walkthrough.

Many different notations have been proposed for representing E-R models. The notation used in lecture is recommended. However, that you use a particular notation is not as important as that you use *appropriate* notation. The notation should be (i) simple, (ii) consistent, and (iii) perspicuous. Be sure to include a key for your notation.

The final version of your E-R model should probably appear in the Requirements Specification in two forms: a “stripped-down” version suitable for the client and a fully detailed version from which the system design can be derived. However, your draft model need not be given in both forms. Hand in a draft with all the details but indicate that you know what to remove for the client. Details commonly deleted when presenting an E-R model to a client include cardinality constraints, indication of weak entity types, constraints on generalization hierarchies, precise specifications of derived attributes, and complex structural constraints on entity instances. Items which clients can normally handle include structured or multivalued attributes, entity identifiers (keys), and value constraints on attributes.

Remember that the E-R model is a description of the information to be maintained by the proposed system, *not* of the representation of that information.

¹⁴ To implement the walkthroughs (both requirements and design), it may be necessary to divide the class into groups of a few teams.

Milestone 2

Milestone 2 consists of three parts: a Requirements Specification, a Project Plan, and a Quality Assurance Plan.

Requirements Specification

The Requirements Specification can be considered as a much expanded version of the Feasibility Study. Whereas the Feasibility Study gave a general picture of the project, the Requirements Specification must go into detail. In particular, the Requirements Specification must completely delineate the scope of the project, explaining what is *and is not* included. On the other hand, the Requirements Specification *should not* delve too far into the design. It should specify the needs of the client, not state how those needs will be met.

The Requirements Specification document will be given to your client. We will then ask the client to certify (i) that the requirement you specified satisfy his or her needs and (ii) that necessary resources will be provided.

Content

This Information Packet does not attempt to give a definitive itemization for the contents of a Requirements Specification, although it does stress a few of the important considerations. It is in fact impossible to give one single list that serves all requirements documents, since each project has unique factors to consider. Consequently, preparing an outline is one of the important early steps for the Requirements Specification.

In order to reflect the structure of the specifications, the document must begin with an introduction, including a section on the *goals* of the product – this is an extraction/expansion/revision of the Feasibility Study. The objectives meeting these goals are then itemized as *functionality* and *qualitative* requirements. Specifications obviously must be specific.

If an existing system (manual or computerized) is being replaced, the old system should be described thoroughly. If possible, include operating manuals and other documentation (this material will be returned if necessary).

The core of the specifications for a modern information systems is the information model, best expressed using E-R diagrams. Data-flow diagrams are almost as important.

General characteristics of the user interface must be specified. The platform used to implement the system will have control over many of the details of the user interface, so it is unnecessary and unwise to specify details over which you have no control, such as menu formats. However, you should specify consistent features which must be present on every screen. For example, specify that help will always be available with the same button and navigation will be consistent from screen to screen.

Your document should contain a description of the deliverables (source code, manuals, *etc.*) that your team will produce for the client.

Finally, requirements go two ways. Your team will require certain things of the client if the project is to be successful. Therefore you should describe the commitments that the

client is expected to make (people for interviewing, computer time for testing, *etc.*).

Presentation

It is important that your Requirements Specification be written in a way that the client can understand. Therefore, design your report to be read by someone not intimately acquainted with P465-6/P565-6. If your report is well organized and to-the-point, the client will be better able to validate your understanding of the requirements. Appropriately labeled diagrams with a brief explanation are often more expressive than pages of text.

An important factor in the presentation of the Requirements Specification (as in most other documents) is the balance between clarity and detail. Because the Requirements Specification governs the rest of the project, it is vital that all detail be included. This may be done without sacrificing readability by placing the details in appendices. A good general rule is that the main body of the Requirements Specification be readable by and meaningful to the client, while matters too technical or too complex for the client to easily read are placed in appendices. For example, the “stripped-down” E-R model should appear in the body of the document while the detailed version is an appendix. Exhaustive listings and tables are usually best put in an appendix.

Other Considerations

We have a special constraint in this course in that we have a predetermined amount of time to finish the project. The easiest way to fit a project to our time scale is to add and subtract functional features as better estimates are made about how much can be done in the allowed time. You are therefore encouraged to establish a priority ordering for the functionality requirements. There will probably be an “essential core” of requirements which are necessary if the project is to be considered at all successful. If the remaining requirements are ordered intelligently, then you can easily delineate the product scope to fit the time limit that we have.

A few pieces of advice based on past experience: Do not underestimate the importance of this milestone. In prior years, more projects failed because the specifications were wrong than because the delivered system did not meet the specifications. Therefore, take the time and put in the effort needed to do a really good job! Make sure that your new system does not merely mimic the old system. Think beyond what your client says she or he wants, considering unmet needs and anticipating future ones. Computer systems may cause great change in users perceptions and needs and you should anticipate this change even if they do not.

Additional Sources on Requirements Specifications

There are several sources of additional material on specifications. *Running the River* has a outline/discussion of the typical Requirements Specification. Class notes on E-R modeling will be available. The IEEE standard outline¹⁵ for Requirements Specifications will be distributed and critiqued. Pressman gives good pointers on preparing a Requirements Specification, including another variation of an outline. Modules D and E of Davis discuss data-flow diagrams and data dictionaries respectively, common techniques for expressing

¹⁵ ANSI/IEEE Std. 830-1984, Inst. of Electrical and Electronics Engineering, *Software Engineering Standards*, New York (May 1987).

system requirements. Although Sommerville's treatment is quite useful, his example is misleading because it displays too much detail about *how* the problem will be solved rather than *what* the product must accomplish.

Project Plan

Once a clear understanding of the project's requirements has been reached, you should be able to begin to plan for the development of the product. While this plan might change as more experience is gained, it is important to have a baseline so that slippages in schedule can be recognized before they become serious.

In this report, you should outline a preliminary schedule to meet the remaining milestones: Prototype, Preliminary Design, Detailed Design, Implementation and Installation. Parallel schedules should be defined for the development of Documentation and Test Data. The Gantt chart¹⁶ is commonly used to provide a graphic view of a project schedule. Your supervisor can also provide valuable advice in this area.

We recommend that you use Microsoft Project to prepare the Project Plan even though MS Project provides more functionality than is necessary for this task. The reason for this recommendation is that MS Project will be useful during the Implementation and Testing phases and therefore it is wise to begin learning it now. Microsoft Project is included in the Office Professional package available through the IU Bookstore for a nominal cost.

Note that a Project Plan is a living document. Even the most experienced team must revise its plans in light of wrong assumptions and changing circumstances. A team just beginning to gain experience in the software development process can expect many revisions and refinements to its Project Plan. Nonetheless, a plan allows for monitoring progress (or lack thereof), thus further allowing for corrective action before catastrophe ensues.

Team Organization

This is probably the first large, multi-person project that many of you have been involved in. You might not realize the special problems that occur when several people must work jointly toward a common goal. The following guidelines should help you achieve your goals. Feel free to ask your supervisor for advice in these matters.

There are many ways to organize a small team, but the most common is to have a single leader who makes all the key decisions and divides up the work for the other team members. A more democratic organization in which the team members share the work can also be successful, but the team must be sure that all the key responsibilities are actually met. You are free to choose your own organization, but you are advised to consult with your supervisor.

In any case, you must designate an Instructor Liaison, a Supervisor Liaison, a Client Liaison, and a Secretary (these are not necessarily different people and the identity of the individuals in these positions might change over time). The Liaisons serve as the primary coordination channel between the team and the respective targets; they are not responsible for all communications. The Secretary is responsible for recording and distributing copies of minutes of team meetings and keeping the Team Log (see page 12).

¹⁶ Davis (Dav), Module L

Quality Assurance Plan

A Quality Assurance Plan (QA Plan) is designed to insure that a project follows procedures which will yield effective, reliable, and maintainable software systems. In most organizations the general guidelines for QA are established at the top level and apply to all projects. This course follows this rule, using a QA plan which follows the IEEE standard¹⁷. A “template” QA Plan will be provided which each project team must customize to its own circumstances.

A template QA plan is provided to minimize your effort. It is not to minimize the importance of the QA Plan. If you need to work many hours for the importance of this plan to stick, that can be arranged.

Deliverables

Your three documents (the Requirements Specification itself and the two accompanying plans) must be submitted in a large three-ring binder, which is referred to hereafter as the “Project Binder”. You will be required to submit successive milestones in the same binder, so that each is in the context of the preceding and the entire history of project milestones is collected together. Therefore by the end of the project you will need a rather thick binder.¹⁸ To reduce bulk and save trees, you are strongly encourage to print all documents duplex (on both sides).

The Project Binder should also at this point contain the original (and marked) Feasibility Study.

We expect the report to be free of spelling and grammatical errors (see Strunk and White).

¹⁷ ANSI/IEEE Std. 983-1986. Inst. of Electrical and Electronics Engineering, *Software Engineering Standards*, New York (May 1987).

¹⁸ Of course you do not need to get a large binder for this milestone, but you will eventually need one 2 or 3 inches thick.

Milestone 3

Prototype

Prototype development is an essential part of the process of moving from the Requirements Specification to the Preliminary Design.

Goals

In the context of this course, there are two goals for the Prototype:

- (1) to validate the users requests and expectations and evaluate the proposed user interface, and
- (2) to gain experience with the target system.

Each team must build a prototype for its project which meets these goals, facilitating refinement of the project requirements and supporting design.

The Prototype supports the Preliminary Design by verifying that you understand the requirements in the same way that the client does. It will give the client the opportunity to evaluate your external interface before you have invested a lot of time programming it.

In industrial settings, where the tool is well-known, goal (2) is unnecessary. In such cases, the best way to demonstrate a Preliminary Design, that is to meet goal (1), is to create a “mock-up” which displays screens or produces reports similar to those that will be produced by the final product. For example, if a web-based interface is the final target, some pages can be quickly created in HTML and interaction can be simulated by merely linking to additional pages. It is therefore understood that prototypes may have little of the underlying semantics. However, goal (2) is essential in this course, perhaps even more important than goal (1), and thus a simple mock-up is inadequate.

The benefits of using a prototype system during the requirements analysis and definition phase of the software life cycle include:

- Misunderstandings between software developers and users may be identified as the system functions are demonstrated.
- Missing user services may be detected.
- Difficult-to-use or confusing user interactions may be identified and refined.
- Incomplete and/or inconsistent requirements may become evident as the prototype is developed.
- The feasibility and usefulness of the application may be demonstrated to management.
- A specification for the development of a production-quality system should result from the prototype.

User Interface Considerations

Since much of the reason for doing the Prototype involves the user interface, it is worthwhile to consider what you will want with that interface.

First, try to follow design principles for user interfaces, such as those of Foley and Van Dam¹⁹: design for consistency, structure the display, minimize memorization, give feedback,

¹⁹ Foley, J.D, and A. Van Dam. *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1982, (Chapter 6).

help the user learn the system, allow undoing, and accommodate errors. You should try out various interface “look and feel” characteristics to fit the principles (for goal (1) above) and to explore the tools available to achieve those principles (goal (2)).

Second, plan to develop systems that *dialog* with users. That is, the system should actively assist the user wherever possible rather than simply responding to a user’s explicit commands. Consider, for example, the process of entering a name in a customer database. As soon as a name is found in the database, the system should automatically move to the appropriate activities for that class of customer. On the other hand, if the name is not found, the system should automatically move to a new-customer-entry screen. The issue of dialog is emphasized because, in line with goal (2) above, you must learn how to implement such dialog; it is likely that implementing easily-flowing dialog will be the most difficult technical task you face.

Deliverables

Your team should develop a prototype which gives a clear picture of your user interface environment (goal (1)) and provides you with good understanding of the project’s platform(s) (goal (2)). This prototype should include at least, where applicable:

on-line input:

all core screens with click navigation but without functionality

example screens with functionality, down to the bottom level for one major branch of the menu selection hierarchy (*i.e.* a depth-first exploration of a menu path)

examples of data-entry screens

events that are:

explicit (*e.g.* button-click)

implicit (*e.g.* entering a field)

form related (*e.g.* switching subforms, drop-down lists)

data manipulation related

implemented by program code, including “behind the scene” communication between code modules

data:

all tables, essential attributes, not necessarily all constraints

on-line output:

sample screens generated by major categories of queries

off-line input:

copies of all forms

off-line output:

draft layouts of major reports

validation of:

at least one input value by format or domain (*e.g.* “integer from 1 to 100”)

at least one input value as foreign key constraint

- mandated interactions with other systems (where possible):
 - demonstrate ability to connect and interact
- error handling (beyond validation failures):
 - trapping at least one database interface error
 - trapping at least one user interface interface error

There are many good tools that assist in information system development; you must be aware that these tools *only* assist. The tools facilitate routine tasks, most relating to the user interface, such as placing a button on a form. However, if the action associated with such a button is at all complex, you will need to program that action. Therefore your prototype must explore the programming environment behind your tool, as indicated by the item “implemented by program code” above. You should explore how to program both interactions with the database (record access and update) and interactions with the user interface (form and subform transitions, enabling and disabling controls, ...).

The example screens with functionality should implement a substantial task. There are a good framework for accomplishing the other prototype objectives, such as demonstrating validation and error trapping. In any case, they must be more substantial than simply implementing a login functionality. However, projects on the IU campus that are using UITS Kerberos login authentication may need to implement login as an interaction with another system.

You should try as much as possible to use the computer environment where the final version of the project will run; however, if the final computer environment is not yet available, then you may use a similar environment available at the university. After an evaluation of your prototype by the team Supervisor and the Instructors you should present it to your client. This will allow you to make any necessary adjustments to your preliminary design before you turn it in.

A special case of the requirement that you try the target environment occurs when your system will operate via a web interface. In this case you must prepare for the fact that your system must behave consistently over a wide range of browsers. Of course it is not possible to try all browsers, but you must try your prototype on at least Netscape and Explorer and at least one current and one old (but not obsolete) version of each.

Each project team should work together to develop their prototype. Of course the team member(s) delegated to learn the tool in depth will perform a larger share of the prototype development. However, it is essential that all team members “get their hands dirty” at this point, in order to prepare them for future Milestones.

Milestone 4

The final milestone of the fall semester consists of two parts: a Preliminary Design and a set of Coding Standards. You should turn in these two items together in the Project Binder but you should understand they have distinct purposes.

Preliminary Design

The Preliminary Design is where “the dream meets reality” – where the goals and objectives of the Requirements Specification begin to take form in the implementation medium.

Recognizing that inadequate or inaccurate requirements are the source of most flaws in information systems, your Requirements Specification must be reviewed and validated by the client. There is not time for the client to complete this review before you begin the design stage; fortunately requirements changes from this review are almost always details that affect the second design Milestone, the Detailed Design. If changes are made to the requirements due either to the client’s wishes or to errors in the original analysis, a list of changes should be provided with the reasons for the changes.

The requirements describe what the product will do for the client. The designs describe how the product will accomplish the requirements. The design stage is divided into two milestones in recognition of several facts: design is the most important phase of the system development cycle, the early design steps explain and validate the requirements, design has a natural iteration in the determine–evaluate cycle, and design has a natural iteration in the successive refinement cycle.

Architecture

A unifying architecture is essential for a successful system implementation; careful presentation of this architecture is therefore the starting point of the design documentation. The **webster** utility provides several definitions for “architecture”; the one relevant here is “a unifying or coherent form or structure.”

In an information system, the architecture is specified by identifying the tables, code modules, and forms that comprise the system and by characterizing the flows between these components. Typically, the most general aspects of this architecture are determined by the software platform(s) used. For example, Microsoft Access places code modules subsidiary to user interface forms.

After specifying the architecture, the design documents should then describe each component, typically by category.

Database Schema

The Preliminary Design should include the relational design for the project, indicating how this relational design is derived from your Entity-Relationship Model. A data dictionary should probably be included, at least in a preliminary form. For example, you might know that a date field is required in a table but not know its exact representation. In the odd circumstance that you will be using something other than a relational database, discuss comparable relevant file organization.

Modules

Both input/output structure and functional behavior can be expressed using HIPO. However, you may also wish to consider using just a hierarchy chart along with a brief explanation of each module's functions, inputs, and outputs. The modules should be related to the data-flow diagram of the Requirements Specification.

If the client has specified that certain complex functions be done in a particular way (*e.g.*, a scheduling algorithm), then you should describe the corresponding algorithms using a high-level pseudo-code specification. However, it is premature to describe or specify other algorithmic details – that is for the next milestone.

User Interfaces

The Preliminary Design must describe the functionality of the user interface, that is, what the user must do to use the system. Much of this description will come directly from the Prototype.

The Preliminary Design should be easy to convert to a user's guide during the documentation and installation phase. One effective way to explain how to use the system is to give a sample script of a session. Ordinarily, you will also need a table-like listing of all the features of your software and how to use them. If you are using a menu driven system, you should describe the general menu conventions and give a representative sample of the menus that the user will see.

The goal is to specify how the user will experience the system (both appearance and functionality), *without* giving any more detail than necessary about how the implementation will be done. For each form, itemize the functionality provided by this form (operations, navigation, *etc.*). In addition, give a “style sheet” that describes the common characteristics of the forms (*e.g.* “top banner with descriptive title”, “back button at lower right”, ...).

Describe the major screens, forms, menus, *etc.* of your application, as well as the relationship between these forms. Initially it will appear that navigation between forms can be presented as a tree structure, but (as always) things are rarely so simple. For example, sometimes the user will need to go from the Order Entry form to the Customer form; at other times, the user will need to go from Customer to Order Entry. It is important to recognize that the user interface structure is distinct from, although related to, the module structure. The chapter on Design in *Running the River* contains more details on this matter.

Most software products will produce reports. You should give examples of a representative set of the formats that you will use (if this is not already part of the carry over from the Requirements Specification). Note that reports are a part of the user interface, even though they may not be part of the on-line side of that interface.

System Interfaces

For the batch-type components of the system, you must still explain how the software is to be used. This will ordinarily be an explanation of the input files which the program will require. If your software is designed to be used with other software, you must specify the interface precisely. That is, explain how your routines are to be called and what all of the parameters are.

Exception Handling

Error-handling and recovery is frequently a major part of a software product. It is not necessary to specify how every error event will be handled in detail. However, you should describe your error handling strategy. Provide a few examples of how typical errors (for example, mistyped input in an interactive system) will be treated. Investigate and discuss what standard mechanisms your tool provides (for example, alphabetic characters in a numeric field).

QA: Coding Standards

The QA Plan requires specification of Coding Standards. Coding-specific matters normally included are: (i) module decomposition rules, (ii) module header formats, (iii) in-line commenting, and (iv) indentation.

Specific hardware and software systems sometimes require additional standards. For example: (v) many PC database systems keep variables in a single, global namespace, requiring variable name standards so modules don't interfere with each other.

User interface matters also require standardization, including: (vi) form layout (color highlighting, button position, *etc.*), (vii) form navigation, (viii) subform characteristics, (ix) appearance and placement of informative messages, and (x) key bindings ("help" should always be the same key, for example).

In the "real world," an organization will commonly have one coding standard for all projects. Because we use so many different platforms, we do not have that luxury.

Deliverables

Because your team already has the Project Binder, which now includes the Requirements Specification, you should not include context information in your Preliminary Design document. This means, for example, that sections 1–5 of the sample design outline in *Running the River* should not be included. That material does belong in a stand-alone design document, but that material is provided by your Project Binder.

Your Preliminary Design report and Coding Standards comprise the final document to be handed in for the first semester. Your Preliminary Design *must* have relevant cross-references to the Requirements Specifications.

Your project grade for this Milestone will be determined by how well your design meets the stated goals of the Requirements Specification, by how well designed it is, and by how professionally the report is prepared.

Project Presentation

The last few days of the Fall semester will be devoted to project presentations. Each team will have approximately twenty minutes to describe the product that they intend to produce. There are several reasons for giving presentations. First of all, they give you a chance to exercise your communication skills. You will find that whatever kind of job you choose, you will have to present your ideas to others at one time or another. Secondly, they help the class as a whole since everyone learns about a large number of projects. Finally, and perhaps least importantly, they provide a basis for comparing the teams for grading purposes.

Your presentation should be based mostly on the Requirements Specification. It should explain *(i)* who the client is, *(ii)* why the product is desirable and useful, *(iii)* what requirements the client has, and *(iv)* how those requirements are met by your design. You should plan to do a professional job. You should prepare the presentation in PowerPoint®, some equivalent Open Source tool, or overhead transparencies. Remember that you only have a limited time – you will be cut off if you run long (and this will hurt your score). It is a good idea to practice your presentation several times before you get up in front of the class.

In targeting the audience for your presentation, imagining that you are speaking to managers who are savvy but not necessarily familiar with the intimate details of technology. You are encouraged to invite your clients to your presentation or to repeat it for them at their site.

For pedagogical reasons, every member of the team is required to participate in the presentation. It is not necessary that every member talk for exactly the same length of time, but everyone should have something meaningful to say, and everyone should have a designated part of the product that they will be responsible for answering questions about. No written report is required for the project presentation.

Design Review

As with the Requirements Specification Review in the fall, the Design Review is an important step on the way to producing the current milestone. As with the Requirements Specification Review, this Review is intended to help the team organize its analysis and design and to have “fresh eyes” look at these. As with the Requirements Specification Review, all material presented at the review should come from the current milestone.

Note that this presentation is called the “Design Review”, without focusing on either the Preliminary or Detailed versions. Thus the Design Review should walk through the overall architecture; indeed, the “big picture” issues at the architecture level fit the limited time of this walkthrough better than module or form details.

While the design documents may rely on the reader having the Requirements Specification at hand, the walkthrough should provide a *limited* view of the requirements in order to place the design in perspective. For example, the E-R model is a good place to start discussing the information requirements, without delving into the analysis that went into developing that model, leading to discussion of the data dictionary and table design. Similarly, an itemization of functionality requirements should lead to the flow through the modules and forms of the design.

Although the Information System sequence attempts to make the project development reflect the “real world,” the design walkthrough is one place where we cut corners (as was noted above, there is not time to cover all details). It is not unusual to have two walkthroughs, each a full day long, for a project of this size. Nevertheless, during their Post Partum Presentations, many past teams have said that the walkthroughs were the single most valuable step in their project cycle.

While discussing the real world, it is worthwhile to recognize that being a reviewer is considered a substantial responsibility by most employers. Thus giving you experience as a reviewer is yet another pedagogic objective in this course.

Milestone 5

Milestone 5 has two components: a Detailed Design and a Test Plan.

Detailed Design

The Detailed Design describes how to break the problem into pieces (modules), what each module does, and how each module is related to the others.

For traditional (imperative) programming language code, a module usually consists of a set of one or more subprocedures which, taken as a whole, constitute a solution to a subproblem of the system. Each module should be documented to show which other modules it calls and is called by, exactly what the module does, and the details of the interface through which the module is called. A system structure chart showing the module relationships graphically is helpful. Although the function of the module must be specified exactly, it is not necessary to give coding details. The specification of module interfaces is probably the most important part of the detailed design. Great care should be taken to insure that the interfaces are complete and unambiguous.

Modules in modern system development toolkits are more difficult to characterize. A module may correspond to a form, an encapsulated database interaction, a report generator, *etc.*²⁰In web-based systems, modules are even generated “on the fly”. In any case, clearly identify the type of each module. In the hierarchy chart, this can be done using different shapes (rounded *verses* square corners, for example) or borders (dashed *verses* solid).

The most obvious way to break a system into modules is to follow the temporal flow of the problem. That is, observe the transformations that an input transaction goes through while being processed by the system and define a module for each transformation. However, this approach has many problems from a software engineering point of view, as shown by Parnas²¹. A better idea is to define modules so that they have small, simple interfaces. This will make it more likely that module integration will not be too difficult. One of the main ideas to come out of Parnas’s work is “information hiding.” The concept of information hiding is to enclose each decision made during the design process within a single module. If the decision requires change later, only one module needs to be modified.

You should plan ahead for the next phase of the development of your system by providing an implementation schedule. For each module you should designate a primary author, first reader, and a scheduled date for (i) all code written, (ii) compilation without error, (iii) acceptance by first reader and passing preliminary testing, and (iv) integration testing complete. Of course, test data generation should be proceeding in parallel to assist in integration testing.

²⁰ Do not be misled because MS Access uses the term “Module” only for code written in their programming language Visual Basic for Access. In fact Access’s Queries, Forms, Reports, and Macros also correspond to our notion of modules.

²¹ Parnas, D.L., “On the Criteria to be Used in Decomposing Systems into Modules.” *Communications of the ACM*, Vol. 15, No. 12, Dec. 1972, pp. 1053–1058.

QA: the Test Plan

As soon as the specifications are firm, you should begin formulating a Test Plan. You should consult your textbook for some ideas on testing programs. Development of the Test Plan should be done in parallel with the development of the Detailed Design.

Your Test Plan should define the steps that you will go through to verify and validate your software. You should include module testing, integration testing and acceptance testing in your plan. Define what criteria you will use to decide that the software is correct. A test plan should describe sets of test data. The client might wish to supply some or all of the data for the final acceptance tests.

There are two main methods for integration testing: top-down and bottom-up. In the top-down approach, the highest level modules are completed first and tested with the aid of “stubs,” which are simply skeleton programs which accept calls from higher level modules and do something “reasonable.” Bottom up testing builds the lowest level modules first and tests them with the help of “driver” programs which supply appropriate test data (perhaps read from a file) to the lower level modules. You may use either approach, but be sure to choose one. Do not try to test your complete system by pasting together all of your independent modules and trying to test the system all at once. Any bugs that are discovered will be very hard to trace. If you do a good job on the detailed design, the implementation phase should be easy.

The Test Plan is a separate document from the Design, but they must be turned in together.

Test themselves follow the Test Plan. But their development (scripts, stub and scaffold definition, test data sets, *etc.*) should begin during the Detailed Design.

Milestone 6

Implementation

If you have done a good job on the earlier milestones, this should be the easiest phase. You should already know a lot about your target hardware/software system. If you didn't learn about your target platform during the development of the Prototype, you may find that you have a lot of redesign work ahead of you. If you did learn, it should be a simple matter to translate the detailed design into coded modules. You are expected to code all the modules using a consistent and well-structured style.

The Milestone for this phase is met by producing a *complete* set of modules. All modules must be functionally complete and without syntax errors. On the other hand, we do not expect all modules to have passed initial unit testing. In fact, the next phase, Testing, should proceed in parallel with the implementation phase. The Milestone should be viewed as a concrete, intermediate goal. It is not a barrier to be surmounted before going on to testing. In fact, the deliverables for the Implementation and Testing Milestones are integrated.

Deliverables

The only deliverable is a form that indicates when each module has been developed and tested, where the developer of each module signs-off on that module. Printable versions of the form are in the files `impl-test-form.p*` in the repository. Those who know \TeX may wish to customize the form similar to the example in `impl-test-exmpl.tex`. Teams may wish to prepare a version in their favorite word processor.

Note that the form should be validated by your Supervisor.

Milestone 7

Testing

This milestone completes the testing activities which began in the first semester with your initial Test Plan. In parallel with the Detailed Design and Implementation phases, you have developed a comprehensive set of tests. Unit testing began, during the Implementation phase, as soon as the first modules were completed. Integration and acceptance testing are the final activities of this process, although the Testing milestone is probably completed before acceptance tests occur.

Unit Testing

Preliminary verification of a module can begin even before it is committed to code. Once a specific implementation is chosen by the author, the author should explain the design to the first reader. If this test is passed, the module can be coded. The author then tests the module by itself (unit testing) by providing it with an appropriate environment and input. After this debugging, the first reader should also try out the module, preferably with new test data. When they are both satisfied that the module is bug-free, the module can be presented to the entire team (see Module A of Davis for a discussion of the Structured Walkthrough technique).

Integration Testing

Although there may be high confidence that each module is independently correct, the system cannot be said to be tested until all the parts are working together. Integration testing involves putting the modules together (using either top-down or bottom up techniques) and testing the integrated system. Errors found during integration testing will frequently point out misunderstandings in the Requirements Specification or Preliminary Design. It is also possible that some interfaces were not accurately specified. All errors must be corrected, and the entire system must be retested.

The Testing Milestone will be met when the complete system has been integrated and tested without error. Your Quality Assurance Monitor will then test the system independently and sign-off on successful completion of all tests. If integration testing has been done adequately, this should not uncover any new errors.

Deliverables

Because unit testing is recorded concurrently with implementation, it uses the same form as described at the end of the previous section.

Integration testing is completed when your team successfully demonstrates your project to the course instructors.

Milestone 8

The final Milestone will have been completed when the Supervisor and QA Monitor have verified that all the Installation items (below) have been satisfied and when a copy of all documentation has been delivered to the Instructors.

Installation

The ultimate goal of the Course Project is to successfully install your product in the client's organization. Installation means that the software must be ported to client's system, acceptance testing must be completed successfully, data must be converted from previous systems, user training must be done, and all documentation must be delivered.

Porting

If you do your final development on your client's system, the "porting" portion of installation is, of course, trivial. However, if you do have to port your software from some other site, be sure to do preliminary trials and leave extra time to straighten out any problems that occur.

Acceptance Testing

Acceptance testing might simply require rerunning your final integration tests for your client. In some cases, however, the client might wish to design his own set of test data for the completed product. If bugs are discovered at this late date, you will be in trouble!

Conversion

If your client had a computerized system already, it may be necessary to convert the data stored in that system. The most common way to accomplish this task is to export the data as ASCII text files and then read those files into the new system. Most modern tools use a standard format with commas delimiting fields in an exported text file, so writing and reading requires no special programming. However, specially written programs may be required, typically when the data needs to be restructured; Perl is recommended for this task. You are not expected to do data entry for the client.

Training

The amount of training required is strongly dependent on the product. However, it is your duty to be sure that the ultimate users of your product can make effective use of it. In addition to the direct training that you provide, you should also produce training documentation for future use.

Project Demo and Walkthrough

The successful installation of your project will be evidenced by a demonstration in its final working environment.

Documentation

Once your team disbands at the end of the semester, the success of your product will depend heavily on the quality of the documentation that you leave behind. Your project

grade will therefore be strongly correlated with how good your documentation is. The minimum documentation required for your project includes:

1. a User's Manual (giving all the details necessary for using the system, what error messages mean, how to use the product, *etc.*) and
2. a Programmer's Reference Manual (linking the design and implementation, giving program structure charts, *etc.*).

The user documentation should include an executive summary (explaining the purpose of the product, when it was developed, for whom it was developed, *etc.*), either as a separate document or as the preface to the User's Manual. You might also want to provide a training manual to help a new user get started with your product. Much of this documentation should come from the documents prepared for the other Milestones.

The goal of the Programmer's Reference Manual is to provide understanding of the internals of the software. Since it is impossible to understand an artifact without understanding its design, the Programmer's Reference Manual must incorporate or refer to relevant parts of the design documents and even the Requirements Specifications.

Post Partum Presentation

The Post Partum Presentation²² gives everyone a chance to benefit from from what you have learned during this course and you a chance to learn from others too. We learn most effectively by facing and (hopefully) overcoming problems. If, upon attempting something for the first time, everything goes well, we are likely to assume that the task is easy or that we are especially skilled. This may lead to a casual attitude which could be costly later. On the other hand, if we have trouble, we are more likely to be aware of pitfalls and avoid them the next time. The post mortem analysis enhances this affect by calling our attention to the problems we faced (and overcame?). The intent of the post mortem analysis is then NOT to find fault but rather to avoid faults in the future. It is an opportunity for your team to sit down and analyze its mistakes and accomplishments.

Many kinds of problems can arise in a software development project. Most of these stem from overconfidence: “Of course we know what the user wants,” “All Foobar compilers are the same, so conversion to another machine will be easy,” “I never have bugs in my programs, and anyway I can fix them within half-an-hour.” Try to recall as many trouble spots in your development process as you can. (Your project log should help here.)

Difficulties can be classified as anticipated or unanticipated and avoidable or unavoidable. An example of a anticipated problem is having bugs in your programs. We expect bugs to occur in programming and so plan to test and debug. Do not dwell on such anticipated problems in the Post Partum, but methods of preparing for and coping with problems are always of interest. On the other hand, when a scheduled milestone is missed or a costly redesign is needed, there is usually an unanticipated problem as the cause. In these cases, you should not only mention the problem, but also what could have been done to anticipate or avoid the problem.

Unavoidable problems are caused by factors outside of the team’s control. Sometimes these can be anticipated (“If the hard disk is not delivered by Jan. 15th then ...”) and backup plans made. Even if there was no reasonable way to foresee an unavoidable problem, it should be described in the Post Partum. By becoming more aware of the kind of things that can go wrong we can become better at estimating the risks associated with projects.

Your class presentation should describe your most significant unanticipated problems, both avoidable and unavoidable. Explain how you overcame the problems (assuming that you did!) and how costly they were (in terms of your time). Suggest how to avoid the avoidable problems and assess the risk of the unavoidable problems. Your written post mortem analysis should be similar to the oral presentation, but should be more detailed. The written report should be turned in with your other system documentation.

²² This presentation is often labeled “post mortem” (“after death” in Latin), reflecting the fact that the project process has completed; however, “post partum” (“after birth”) reflects the fact that the project has been delivered, about nine months after conception.

Reporting: Forms and Logs

There are a number of forms which you individually must fill out and others which must be filled out by each team. The individual forms are given here. An ASCII file template for the Team Log file is available in the NFS repository directory (see page 12). Additional team forms will be made available during the second semester, since they apply to milestones 5 and 6.

These instructions should be carefully read before attempting to fill out the forms on the following pages. If you fail to follow instructions accurately, your grade could be affected.

Supervisor Evaluation Form

The Supervisor Evaluation Form should be self explanatory. These will be kept confidential. Your comments will be taken into account in determining the Supervisor's grade in B665 and B666.

Team Member Evaluation Form

This form is to be filled out at the end of the semester and turned into the Project Instructor. The information is confidential and will not be revealed to the other team members.

You are to evaluate the performance of all of your team's members (including yourself).

The primary means of evaluation is a numeric score from 1 to 9, with 1 meaning terrible, 5 meaning average, and 9 meaning terrific. THE TOTAL POINTS AWARDED MUST BE EXACTLY FIVE TIMES THE NUMBER OF MEMBERS IN YOUR TEAM. For example, if you have four members and you give yourself 7 points and two other members 5 and 4 points, then you must give the remaining member exactly $4 \times 5 - 7 - 5 - 4 = 4$ points. Scores that deviate very far from 5 require strong reasons. For example, a score of 1 would be appropriate for someone who refused to help with the project, and a score of 9 might be deserved by someone who did almost all the work by themselves.

The "check-off" boxes on the front of the form and the comment space on the back should be used to justify your scores. Do not spend excessive time filling out the check-off boxes; they are there to help provide an overall picture of each team member. The comment for a team member should begin with an itemization of that member's contribution.

Remember, be candid and make sure that your scores total correctly.

Team Log

A Team Log file must be emailed each week to your team supervisor (see page 12).

The Team Log template assumes one formal meeting a week at which project status is evaluated. You will be asked to log the assignment, status, and completion of various project components.

Other Forms

Other forms will be made available on the NFS repository. In particular, the PostScript file eval.ps contains the forms used by the course instructors to evaluate each of the milestones.

Supervisor Evaluation Form

Team: _____

Supervisor: _____

Your Name: _____

This side asks you to give information about specific issues concerning your supervisor's interaction with your team. The other side asks more open-ended questions. Please answer these questions thoughtfully, completely and honestly.

Indicate whether you agree or disagree with the following statements as applied to your supervisor:

	strongly agree		neutral			strongly disagree	
met at least weekly with the team	7	6	5	4	3	2	1
responded adequately to email	7	6	5	4	3	2	1
followed up adequately	7	6	5	4	3	2	1
provided leads to information	7	6	5	4	3	2	1
monitored team log effectively	7	6	5	4	3	2	1
was supportive of the team	7	6	5	4	3	2	1
provided adequate technical guidance	7	6	5	4	3	2	1
provided adequate process guidance	7	6	5	4	3	2	1
prevented wasted time	7	6	5	4	3	2	1

Our supervisor helped manage the team with (circle *all* that apply):

role assignment	group dynamics	task delegation
task monitoring	quality control	<i>other:</i> _____

Our supervisor gave us guidance on the indicated tasks (circle *all* that apply):

client interaction	project planning	ER modeling
other modeling	other analysis	database design
U.I. design	architectural design	document design
document prep	document review	<i>other:</i> _____

Our supervisor's interaction with the team was (circle all that apply):

enthusiastic	supportive	positive
distant	confused	erratic
weak	heavy-handed	<i>other:</i> _____

Our supervisor's management style was (circle *one*):

facilitator	meddler	nit-picker
mother-hen	dictator	ghost
<i>other:</i> _____		

In what ways did your supervisor assist you in completing your project on time this semester? Was the supervision an overall benefit in improving the quality and/or timeliness of your work?

Was your supervisor as helpful as he/she could have been, given that the supervisor had only a limited amount of time to devote to this project? What should the supervisor have done differently?

Did your supervisor provide you personally with adequate feedback as to your performance? Comment.

Did your supervisor provide adequate communications to and from the “course management” (instructors and AI’s)? Comment.
