SOFTWARE ENGINEERING LABORATORY

Projects and Guides

Table of Contents

Course Syllabus

Class time and location: TThF 3:30-5:00, Braunstein 207
Instructor: Prof. Hal Carter, Rhodes 814,
Teaching assistant: Darren Insko, Rhodes 805D

Course Summary and Objectives:

This course is the laboratory component of the ECE 493 Software Engineering course. During the laboratory course the student will apply formal software engineering techniques to the development of a software product. Each group of three students will create the project plan, requirements specifications, design specifications, code, and test plans for a software program to be coded in Ada. Maximum re-use of software components will be emphasized to ensure timely development of a significant product in a short period of time. Each group will also prepare five formal documents and participate in five formal reviews. Each software project will be demonstrated and evaluated at the end of the course.

Prerequisites:

ECE 226 - Intro to Programming, ECE 328 - Data Structures and Algorithms.

Co-requisites

ECE 493 - Software Engineering

Required Texts:

Naiditch, David J., **Rendevous with Ada: A Programmer's Introduction**, John Wiley and Sons, 1989.

Useful Documentation: (Sections to be passed out during the course):

Software Engineering Laboratory: Projects and Guides, Winter 1992.

Grading:

The grades for this course are based upon the following weighted basis:

Item	% report/review	% total grade
Reports		70
Software Project Plan	15	
Software Requirements Spec	25	
Software Design Spec	25	
Software Code	15	
Software Test Plan	15	
Course Evaluation	0	
Final reports on floppy disk	5	
Reviews		30
Project Review	15	
Requirements Review	20	
Design Review	20	
Code Review	15	
Test Plan Review	15	
Demonstration	15	
Total		100

The criteria for grading reports and reviews are:

Criteria	weight (%)
Documents:	
Content scope	20
Content correctness	25
Content completeness	25
Spelling and Grammar	10
Composition	10
Code:	
Completeness of internal documentation	20
Adequate unit test programs	20
Complete code	20
Correct code	20
Coding style	20
Reviews:	
Content scope	20
Content correctness	25
Content completeness	25
Presentation overheads	10
Presentation style	10

Project Organization:

Each team of three students will select a project from the list of projects provided later in this guide. Each team will consist of a team leader, analyst, and test engineer all elected by popular consensus of the group. All team activities will be mamaged by the team leader. Further, the team leader will be responsible for interfacing with the project customer (i.e., lab instructor).

The development of the sofware product will take place in the standard software engineering phases. Work will *not* proceed to the next phase until the customer has officially signed off the documentation for the previous phase.

All coding shall use only the Ada programming language, and will make maximum use re-usable components.

Course Schedule

The overall schedule for the course is:

Wk	Day	Date	Description	Time	Reading	Report due
1	Tu	Jan 7	Intro to lab Project descriptions Project plan	2:30-3:00 3:00-3:30 3:30-4:00	Syllabi, schedule P&G[1]: projects P&G: doc guides	
2	Tu	Jan 14	Requirements spec Intro to Interleaf Ada programming	2:30-3:00 3:00-3:30 3:30-5:00	P&G: doc guides N[2]:1-5	
3	Tu	Jan 21	Ada programming Project review (20 mins/group)	2:30-3:30 3:30-5:10	N: 6,7	Project Plan
4	Tu	Jan 28	Design spec Ada programming	2:30-3:00 3:00-4:30	P&G: doc guides N:8-12	
5	Tu	Feb 4	Requirements review (30 mins/group)	2:30-5:00		Reqs Spec
6	Tu	Feb 11	Ada programming	2:30-4:30	N:13-15	
7	Tu	Feb 18	DesignReview (30 mins/group)	2:30-5:00		Design Spec
8	Tu	Feb 25	Coding practices Test Plan	2:30-3:00 3:00-3:30	 P&G: doc guides	
9	Tu	Mar 2	Code Review (30 mins/group)	2:30-5:00		Code
10	Tu	Mar 9	Test Plan review (30 mins/group)	2:30-5:00		Test Plan
		Mar 9-13	Project demos in Rhodes 800 (30 mins/group)	Exam schedule		Course eval, All rpts on disk

[1] Software Engineering laboratory: Projects and Guides
[2] Naiditch, David J., *Rendezvous with Ada: A Programmers Introduction*, Wiley, 1989.

Documentation Outlines

This section contains the documentation outlines along with statements of required content. The following guides are included:

1. Software Project Plan
2. Software Requirements Specifications
3. Software Design Specifications
4. Coding Style
5. Software Test Plan
6. Course Evaluation Form

The plans and specification reports given here are taken from Pressman, 3rd Ed. while the content descriptions are in part derived from ANSI/MIL STD-2167A. The outlines are provided in three formats: Word for Windows, V. 2.0 (*.mww), ASCII text, (*.txt), and Interleaf.

Software Project Plan

SOFTWARE PROJECT PLAN
Outline and Requirements

***General comments**: The following guidelines are given in italics and represent the bare minimum required to complete this plan. Non-italicized text is to be included in your plan verbatim. All sections must be included in your plan, even if you do not feel it applies.*

1. Introduction

One or two paragraphs which introduce the document..Several sentences on the software to be developed can be included.

1.1. Scope and Purpose of Document

This document outlines the necessary time and cost estimates, risks and risk abatement, and resources required to carry out the development of the software project. This document is primarily for use by the development group, and may be used by the project sponsor as information relating to the management and resource estimates of the project.

1.2. Project Objectives

1.2.1. Objectives

Itemize the objectives of the software development. Characteristics to consider are:

1. *Intended use of the software and scope of user base*
2. *Environmental constraints in which the software will be executing*
3. *Lifetime of the software product*

1.2.2. Major Functions

Identify all the major functions of the software to be produced at a high level. For example, consider a compiled simulator project. The major functions could be:

1. *Product will provide a capability to compile a source command language into an intermediate format and store the resulting modules in a user-controllable library.*
2. *Integrate pre-compiled modules stored in the user-controllable library into an executable simulator program.*
3. *Load and execute the executable simulation program during which event times and values are stored in a file for subsequent diagnostics (diagnostic software is not to be a part of this product).*

1.2.3. Performance Issues

One or two paragraphs detailing all issues relating to the speed at which the software needs to execute. If realtime needs are important, identify them here. Even if timeliness of software execution is not an issue, say so and clearly state why there is no need to be concerned about performance.

1.2.4. Management and Technical Constraints

At a minimum provide one paragraph detailing mangagement constraints associated with the software development, and another paragraph identifying all known impacting technical constraints. If there are a number of constraints, itemize them in lists. Constraints include computers the software is to operate on, cross compilation constraints, development time and cost constraints, longevity constraints, memory limitations, and processor limitations (e.g., math coprocessor, peripherals).

2. Project Estimates

2.1. Historical Data Used for Estimates

Show sources and organized data obtained from prior projects that you used to in producing your estimates. Such information is available in journals and publications if direct knowledge of prior projects is not known.

2.2. Estimation Techniques

Outline all estimation techiques you used to develop your estimates in the next section.

2.3. Estimates

This is the primary section in this part of the plan. Show in detail all estimates for manpower loading, hours per task to be accomplished, and costs (yes, even for student projects). Show all supporting data. A complete spreadsheet form of presentation is preferred.

3. Project Risks

3.1. Risk Analysis

3.1.1. Identification

Identify all risks associated with this project. Potential risks are 1) limited time to develop software, 2) lack of software development experience of personnel, 3) lack of complete understanding of requirements, 4) improper motivation of developers, 5) lack of knowledge of implementation language.

3.1.2. Risk Estimation

Give an estimate of the importance of each risk identified above. Translate these importance factors into weighting factors to be used in the next section.

3.1.3. Evaluation

Evaluate the impact of all risks on the development and quality of the software you will be developing.

3.2. Risk Management

3.2.1. Risk Aversion Options

Identify all options available to your group to avoid or control risk. Give a table reflecting how each risk aversion technique will reduce each risk.

3.2.2. Risk Monitoring Procedures

Give several procedures in a list of step format that you will implement to reduce all risks.

4. Schedule

4.1. Project Work Breakdown Structure

Give a complete work breakdown structure for your project. This requires you to identify the major subsystems to be developed.

4.2. Task Network

Provide a task network diagram with sufficient explanation which corresponds with the tasks and breakdown structure.

4.3. Timeline Chart

Provide a Gantt chart for the schedule of effort for your project. The schedule should be detailed to the day. Identify timelines for all tasks shown in the breakdown structure.

4.4. Resource Table

Give a table of resources to be used by task.

5. Project Resources

5.1. People

Identify the people participating in the development of the project. Provide a brief (e.g., one or two paragraphs each) biography of each person as it relates to their skill and qualifications to carry out the project.

5.2. Hardware and Software

Identify all hardware and software resources that will be used to develop the project items. Describe when during the development process they will be used. Project items include reports, source code, executable code, and any other deliverables associated with the project.

5.3. Special Resources

Identify any special or unique resources, if required, that are needed. These resources are usually unique because of limited availability or high useage cost.

6. Staff Organization

6.1. Team Structure

Give an organization chart and description of the organization of the group developing the project.

6.2. Management Reporting

Describe how the members of the group will report to the team leader (and possibly to each other). Identify how the group will be keeping the sponsor informed. Describe how the leader will report to the group.

7. Tracking and Control Mechanisms

Describe the policies, processes, and enforcement methods to be used to assure the development process is carried out with the lowest possible risk.

8. Appendices

Software Requirements Document

SOFTWARE REQUIREMENTS SPECIFICATION
Outline and Requirements

***General comments**: The following guidelines are given in italics and represent the bare minimum required to complete this plan. Non-italicized text is to be included in your plan verbatim. All sections must be included in your plan, even if you do not feel it applies.*

1. Introduction

One or two paragraphs which introduce the document.

1.1. System reference

One or two paragraphs which briefly describes the overall system within which the software product executes. This description identifies the context for the software product.

1.2. Overall description

One or two paragraphs which briefly describes the software product. This description is an overview of the product itself which introduces it to the reader.

1.3. Software project constraints

One or two paragraphs which briefly identifies the major constraints imposed upon the product. Use a list format if there are more than two constraints.

2. Information Description

2.1. Information flow representation

One or two paragraphs which introduce the major information flow of the software product. Also identify nomenclature and symbols to be used in this section.

2.1.1. Data flow

Using a hierarchical decomposition of data flow diagrams, show, in detail, the data flow of the software product and all its subsystems. Clearly describe the dataflow diagrams. Use consistent naming conventions for all data. This section is usually quite lengthy and care must be taken to organize the presentation well.

2.1.2. Control flow

Using finite state machine models or other similar methods, show, in detail, the flow of control in the software product and all its subsystems. Clearly describe all diagrams. Ensure all names are consistent with the data flow diagrams in the previous section. This section is usually quite lengthy and care must be taken to organize the presentation well.

2.2. Information content representation

A complete description of all data identified in the Information flow representation section above is given in this section. The data type, size, description, and other attributes of each named data item is given, usually in tabular form.

2.3. Standard interface description

Clearly describe the hardware, software, and human interfaces to external system elements and internal software functions. Use drawings as necessary to show where the interfaces exist and the contents of the interfaces.

3. Functional Description

Give one or two paragraphs which summarizes the functional description of the software product.

3.1. Functional partitioning

Provide a partitioned functional representation of the software product in a hierarchical fashion. Use a formal representation, use text to clarify the figures, carry the decomposition to the module unit level.. Ensure all named data or control items are consistent with the data and control-flow models described above.

3.2. Functional description

For each module identified above:

3.2.1. Processing narrative

Describe the operation of the module.

3.2.2. Restrictions and limitations

Identify all restrictions and limitations imposed upon the module.

3.2.3. Performance requirements

Identify all timing, delay, or other performance constraints for the module. Note that module performance requirements are usually derived from system and subsystem constraints given in the Functional Partitioning section above. For time constrained software, a flowdown of module delays is documented either in this section or in the Functional Partitioning section above.

3.2.4. Design constraints

Identify and justify all design constraints for each module.

3.2.5. Supporting diagrams

Show and describe all data-flow and control flow diagrams for the module.

3.3. Control description

For each module:

3.3.1. Control specification

Identify the control inputs and outputs for the module.

3.3.2. Design constraints

Identify all design constraints for the module.

4. Behavioral description

4.1. System states

Decompose, show, and describe all finite state diagrams and/or decision charts reflecting the control of the system, subsystems, and significant control-intensive modules. Clearly describe the inputs, output ID's, and states for all control state transitions.

4.2. Events and actions

Clearly identify and describe all events creating the inputs, actions performed by the output ID's.

5. Validation criteria

5.1. Performance bounds

Identify the performance expectations and bounds for the software product and its major subsystems. Testing will validate against these bounds.

5.2. Classes of tests

Give a table and description of all major classes fo tests to be applied to the software once constructed to assure compliance with requirements specs.

5.3. Expected software response

For each test class, define the expected software response.

5.4. Special considerations

Claify any special issues relating to validation testing such as special setup conditions, additional resources necessary to conduct the tests, etc.

6. Bibliography

Give references to all documentation used to create this specification including:

- *other software engineering documentation*
- *technical references,*
- *vendor literature,*
- *standards*

7. Appendix

Software Design Specification

SOFTWARE DESIGN SPECIFICATION
Outline and Requirements

General Comments: *The following guidelines are given in italics and reporesent the base minimim required to complete this plan. Non-italicized text is to be included in your plan verbatim. All sections must be included in your plan, even if you do not feel it applies.*

1. Scope
The general scope of the project is given in this section. Give one or two introductory paragraphs here.

1.1. System Objectives
Clearly state the overall objectives and subobjectives of the system of which this software project is a part. These objectives must be consistent with the objectives given in the Systems Requirements Document and the Software Requirements Document.

1.2. Hardware, Software and Human Interfaces
Clearly identify the scope of all interfaces between the software project described by this document and the external software programs, hardware ports, and human functions such as display screens and mouse inputs. Include a context diagram or figure to place your descriptions in context. Note that this section describes the scope of interfaces, not the interfaces themselves.

1.3. Major Software Functions
Using applicable diagrams and text, describe all major software functions that are a part of the software project described in this document. These functions should be described only to the extent that the functional scope of the project is made clear.

1.4. Externally Defined Database
Define and characterize any significant files, external databases or database systems used by the program. Level of description should be enough to establish the scope of interaction between the program and the database or database system.

1.5. Major Design Constraints and Limitations
Identify all design constraints and limitations against which the software will be defined. Repeat the constraints and limitations from the SRS and further refine their description at a design level.

2. Reference Documents
Identify all documents used throughout the design process. Reference documents include:

- *Mandatory compliance and guidance standards*
- *Documentation on all prior software, components, and libraries used in this project*
- *Significant CASE tools documents*
- *Sources of algorithms, methods, or significant processes used in developing or within the software itself*

Use a standard bibliography style of reference. Separate the listings of documents into the following categories:

2.1. Existing Software Documentation

3. Design Description

This section contains the actual design descriptions of the program, subsystems, and each module. Use graphics and tabular representations augmented by clear and consise text. Cross-reference liberally.

3.1. Data Description

3.1.1. Review of Data Flow

Copy significant data flow diagrams from the SRS to this section. Explain the diagrams with text to ensure section stands alone (i.e., do not make references back to the SRS). At a minimum the level 0 and level 1 diagram must be shown and described.

3.1.2. Review of Data Structure

Copy from the SRS the data descriptions of all data id's shown in the figures, table, or text given in Section 3.1.1 above. Explain all data as necessary to ensure section stands alone without reference to the SRS.

3.2. Derived Program Structure

Give a sequence of heirarchical structure chart figures in decomposeable order each augmented with tables and text for added clarity. Note that the hierachical charts do not necessary reflect the same organization as the data flow diagrams in the SRS, but rather reflect a module-level structure of the program.

3.3. Interfaces within Structure

Completely identify all aspects of the major interfaces between structure elements in the program. Such interfaces may include data structures, timing diagrams, lists of data item id's, or other form of explanation. Note that each interface will include explanatory text, and should include graphical representations wherever possible. The depth of description should be enough that a complete module design can be given in the next section where the level of abstraction of the interface is no higher than that of the module design.

4. Module Design

This section should be organized by module. For each module present in detail the design of the module. The sections to be given for each module are:

Processing Narrative	*Describe the behavior of the module in text.*
Interface Description	*First give the calling syntax as **proc (arg, arg, ...)** then define each arg, proc type, and return type.*
Design Language Description	*Code the module using pseudo-code or formal design language.*
Modules Used	*Give a list of modules called by this module. Give a short phrase description of each module name on the list.*
Data Organization	*Completely describe each internal data item in the module using a formal data method (e.g., C types, ADA types, etc.).*
Comments	*Give in text all special notes or other items not included in the Processing Narrative section.*

This section can be quite large. Make up each section in a form-like format to make the task of preparing this section more structured.

5. File Structures and Global Data

Give a brief textual description of each file and global data items as a means of introducing this section.

5.1. External File Structures

For each file, give the following:

Logical Structure	*Describe the logical structure of the entire file graphically or tabularly. Use text as necessary to clarify the structure.*
Logical Record Description	*For each record type in the file, describe its strucuture and meanings of each field.*
Access Method	*Identify the method of access to the file (e.g., hash, indexed. pile, sequential, etc.).*

The use of formal data modelling methods is helpful, but not strictly necessary.

5.3 Global Data

Describe each global data item in turn, giving sufficient information for direct coding into data types in the implementation phase. The use of formal data modelling methods is helpful, but not required.

5.3. File and Data Cross Reference

In this section, give in tabular and/or graphical form the association between data items in the program and data items in each file. For instance if you have an input file where each line is read in turn and the data is parsed into a data structure called inline, then show how the line structure and fields maps into the data structure items. Also give any enumeration or range limits.

6. Requirements Cross Reference

This section of several tabular lists such as

- *Module versus requirement in the SRS*
- *Module data item versus data dictionary entry in the SRS*
- *File name versus file name in the SRS*
- *... and any other major association between the contents of the SRS and the design*

7. Test Provisions

This section refines the test plan information given in the SRS. Note that the tests themselves are not specified here (they are given in the Software Test Plan document).

7.1. Test Guidelines

Identify all guidelines and suggestions for test setup, environment, and application. This section applies primarily to the unit testing of each module or collection of units performing a function.

7.2. Integration Strategy

Give an outline of the procedure or sequence of tasks (including testing tasks) for integrating the modules into a working program. Ususally, each unit (i.e., module) is unit tested, then the modules are integrated into functional collections of modules and each is tested. Then these units combined into subsystems each of which are tested followed by a complete all-up program test. This section should give a "cookbook" step-by-step presentation of this process.

7.3. Special Considerations

This section includes further clarification on test development or application. For instance, if some special stubs are needed they can be further described here.

8. Packaging

This section describes how the executing segments of the program go together. If the program is segmented by overlays than section 8.1 applies. Other forms of program segmentation are allocation of modules or objects to nodes in a parallel processor, assignment of program sections to parts of memory (such as the TPA, high memory, or extended memory on a PC). If there are no special packaging requirements, so state.

8.1. Special Program Overlay Provisions

Graphically show how the program is partitioned for memory or processor allocation purposes. Use text for clarification.

8.2. Transfer Considerations

If the program is dynamically allocated during operation (e.g., VROOM by Borland Int.) describe its operation here. This section generally applies to programs operating over a suite of processors on a network.

9. Special Notes

Anything that doesn't fit above goes here.

10. Appendices

The Appendix contains copies of vendor or internal documents, large graphics that do not fit well in the body of the document, or other information gemain to the document but to unwieldy or unimportant to put in the body of the document.

Coding Style

Coding Style

General Comments: *The following requirements are the minimum required to ensure correct coding style for support and maintenance.*

These guidelines have been adapted from a coding style guide authored by Prof. Karen Davis, University of CIncinnati.

1. Identifiers
Identifiers shall be descriptive of their purpose or content. Single letter or non-meaningful names are not acceptable. Use correct English spelling, do not haphazardly eliminate vowels. Use abbreviations or acronyms only if they are commonly known.

2. Named Constants
Named constants shall be used rather than literal constants.

3. Modularity
The code shall be written in a modular style. Each module should have only one well-defined task.

4. Variable References
No non-local references to variables shall be made. Variables used in a module are either local or are parameters.

5. Data Abstraction
Good data abstraction shall be employed. The only access to an abstract data type shall be through the procedures or functions that define the behavior of the abstract data type.

6. Partitioning
A complete program shall be partitioned into one or more modules each of which are represented as source code in a separate named file. Each module shall have a module description written as comments at its beginning. The main or root module shall also contain the program code documentation written as comments at the beginning of the file before the module description.

6.1. Program Code Documentation
The format of the program code documentation is:

Purpose:	A brief statement of the purpose of the program
Invocation Syntax:	The specific command syntax for executing the program and definition of each command line option.
Description:	A description of what the program does.
Copyright:	A statement of the copyright including limitations and owner.
Notes:	Any additional notes about the program.

6.2. Module Description

The format of the module description is:

Module name:	Name of the module.
Pupose:	Brief purpose of the module relative to the program as a whole.
Calling syntax:	The specific calling syntax of the module in the form `proc(a,b,c,...) return x` where `return x` applies only to functions.
Inputs:	List of procedure inputs and their definitions.
Outputs:	List of prodedure outputs and their definitions.
Files used:	List and definition of files read and/or written.
Description:	A brief and complete description of the module. If module implements a published algorithm, give reference.
Author:	Name of primary author of the module.
Date:	Date of last update to the module.
Revisions:	List of revision descriptions with latest revision first.

Note that both program and module descriptions are a high-level description of what task the code accomplishes. It should not mention data structures, program modules, or variables. It should be understandable by an intelligent non-programmer.

Correct spelling and punctuation shall be used in all textual descriptions including comments written throughout the code.

The description of program input and output contains the source/destination of all input and output, as well as the formats used for each. BNF may be used for describing formats. A brief example of input and output is often useful.

7. Indention

Adequate use of whitespace shall be employed for indentation and separation of modules. Four spaces for indentation is suggested, and uniform spacing to indicate levels of control flow is required.

8. Line Length

Long lines shall be broken and indented so that they may be easily read on a screen (no longer than 80 characters).

9. File Protection

Unless several students are working on a module, the files containing the module sources shall be set so that no one but the individual programmer may have access to it. For example `chmod 600 filename` gives read/write access to you and not the rest of the world under UNIX.

Software Test Plan

SOFTWARE TEST PLAN
Outline and Requirements

General Comments: *The following guidelines are given in italics and reporesent the base minimim required to complete this plan. Non-italicized text is to be included in your plan verbatim. All sections must be included in your plan, even if you do not feel it applies.*

1. Scope of Testing
The general scope of the test plan is given in this section. Summarize the specific functional, performance, and internal design characteristics that are to be tested. Briefly describe the bounds of the testing, criteria for completion of each test, and schedule constraints.

2. Test Plan
Testing is divided into phases and builds that address specific functional and behavioral characteristics of the software. Each of these builds is a group of modules which are created in a phase.

2.1. Test Phases and Builds
Describe each software development and test phase, and describe the build process to implements that phase.

2.2. Schedule
Provide estimated start and completion dates for integration, overhead software development to support test (see next section), test application periods, and test analysis periods (if applicable) for each test phase. Note that each of these development of test periods imposes an availability of unit tested modules to integrate and test.

2.3. Overhead Software
Overhead software consists of software test benches, stubs, and drivers necessary to carry out the testing. Describe these overhead items.

2.4. Environment and Resources
Define the normal test environment including hardware platforms or external resources, and software tools necessary to conduct the tests. Also define any unusual hardware configurations, exotic simulators, special test tools or techniques.

3. Test Procedure: Build *n*
This section prsents the detailed testing procedure required to accompish the test plan (Section 2 above). Separate section 3's containing the unit and integration tests are defined for each build.

3.1. Order of Integratiion
Briefly describe the order of unit and subsystem integration to be carried out to create the build.

3.1.1. Purpose
State the purpose of the build and the testing of the build.

3.1.2. Modules to be Tested
Identify the modules in this build to be tested and integrated in the order they will be tested.

3.2. Unit Tests for Modules in Build

This section describes the actual test for each module m. Briefly introduce the test here. There are m Section 3.2's, one for each module being tested.

3.2.1. Description of Tests for Module *m*

Describe each test using a formal point-by-point statement of each test.

3.2.2. Overhead Software Description

Describe all software items such as stubs, drivers, and test benches to check out this module.

3.2.3. Expected Results

Identify the expected results of the test. Write this section in a list fashion such that each test motivates one or more output descriptions.

3.3. Test Environment

3.3.1. Special Tools or Techniques

Identify all special hardware or software tools or techniques necessary to carry out this test.

3.3.2. Overhead Software Description

Describe the overhead software (.i.e., stubs, drivers, test benches) at a level detailed enough to support coding the item.

3.4. Test Case Data

Identify all test data to be used to conduct the test.

3.5. Expected Results for Build *n*

Identify the expected results for each input test data case. Note test case data and expected results can be described in a singe section mainly consisting of large table where test case data is presented on the left side of the table, and expected results shown on the right side of the table.

4. Actual Test Results

This section consists of forms with blank spaces to record the actual test results along with the date of the test, who the tester is, etc. Again, this section can be integrated with Section 3.4 and 3.5 for each build as an option.

5. References

Identify all documentation references where supporting material has been used.

6. Appendices

Presentation Evaluation Form

This form will be completed by the Professor (and the Teaching Assistant as a cross-check) during each presentation by each team.

Review Grading Sheet for _____ Presentation.
Date:_____
(e.g., Project Plan, Requirements Spec, etc.)

Team	Criteria	Wt	Grade	Comments
1	Scope	20		
	Correctness	25		
	Completeness	25		
	Overheads	10		
	Presentation	10		
	Grade:	100		
2	Scope	20		
	Correctness	25		
	Completeness	25		
	Overheads	10		
	Presentation	10		
	Grade	100		
3	Scope	20		
	Correctness	25		
	Completeness	25		
	Overheads	10		
	Presentation	10		
	Grade	100		
4	Scope	20		
	Correctness	25		
	Completeness	25		
	Overheads	10		
	Presentation	10		
	Grade	100		
5	Scope	20		
	Correctness	25		
	Completeness	25		
	Overheads	10		
	Presentation	10		
	Grade	100		

Scope:	Is content of presentation within the scope of the intent of the review?
Correctness:	Is material presented correct?
Completeness:	Is presentation complete?
Overheads:	Were overheads prepared in a professional manner?
Presentation:	Was the presentation given in a professional manner?

Project Descriptions

Project 1: File Folder Database Manager

A File Folder Database Manager maintains a database of the folders in a filing cabinet. It allows the user to organize the folders by a taxonomy, specify their locations, and print reports on them.

1. For each folder (record), the following information is maintained:
2. Top-level taxonomy entry (40 characters maximum)
3. 2nd-level taxonomy entry (40 characters maximum)
4. 3rd-level taxonomy entry (40 characters maximum)
5. Description of folder contents (3 lines of 60 characters/line maximum)
6. ID of drawer containing folder (5 characters maximum)
7. ID of folder in drawer (5 characters maximum)

The database manager should perform the conventional database functions (create, import text files, export text files, add records, delete records, sort records, etc.) and generate the following reports:

1. A listing of the database ordered by the three taxonomy entries
2. A listing of the unused drawer/folder IDs
3. A listing showing the last folder ID for each drawer ID

Project 2: Cruise Control System

A cruise control system maintains a car's speed, even over varying terrain. The basic idea is that the driver engages the cruise control system, the car reaches a desired speed, and the driver tells the cruise control system to start controlling the car's speed.

If the cruise control system is controlling the car and the driver touches the brake pedal, the cruise control system stops controlling the car's speed but remains engaged and remembers the last speed set by the driver. After touching the brake pedal, if the driver presses the resume switch, the cruise control system resumes controlling the car at the last desired speed.

If the cruise control system is controlling the car and the driver depresses the accelerator pedal to speed up, the cruise control system stops controlling the car's speed until the driver releases the accelerator pedal. After the driver releases the accelerator pedal, the cruise control system resumes controlling the car at the last desired speed.

This problem is the design a cruise control system and a screen-oriented user interface from the driver's perspective. Through this interface the driver (at the keyboard) should be able to do the following:

1. Start the engine
2. Depress the accelerator pedal
3. Depress the brake pedal
4. See the speed of the car (a speedometer reading is displayed)
5. See if the cruise control system is engaged and if it is currently controlling the car
6. Engage the cruise control system

7. Set the current speed as the desired speed, allowing the cruise control system to control the car's speed
8. Press the resume switch, allowing the cruise control system to resume control of the car's speed
9. Disengage the cruise control system
10. Stop the engine

For the purpose of this problem, an interface to the automobile hardware is provided by an automobile interface package. If this exercise were to be extended into a real application, the only difference would be a change to the package body of this automobilie interface from a simulation to an actual hardware-level interface to the sensors and actuators in the automobile itself. The following is the package specification of the automobile interface (tentative, subject to change):

```
-- ****************************************
-- *                                      *
-- *  Automobile_Interface                *  SPEC
-- *                                      *
-- ****************************************
package Automobile_Interface is

  Maximum_Speed : constant := 120.0;
  type SPEED is new FLOAT
      range 0.0 .. Maximum_Speed;  -- MPH

  -- ......................................
  -- .                                    .
  -- .  Automobile_Interface.Turn_On_Engine    .  SPEC
  -- .                                    .
  -- ......................................
  procedure Turn_On_Engine;
  --| Purpose
  --|    Turn on the automobile engine.  The Brake
  --| Pedal and Accelerator Pedal routines are
  --| activated.

  -- ......................................
  -- .                                    .
  -- .  Automobile_Interface.Turn_Off_Engine   .  SPEC
  -- .                                    .
  -- ......................................
  procedure Turn_Off_Engine;
  --| Purpose
  --|    Turn off the automobile engine.  The Brake
  --| Pedal and Accelerator Pedal routines are
  --| deactivated and the car comes to a stop.

  -- ................................................
  -- .                                              .
  -- .  Automobile_Interface.Depress_Accelerator_Pedal  .
SPEC
  -- .                                              .
  -- ................................................
  procedure Depress_Accelerator_Pedal;
  --| Purpose
  --|    The car accelerates.
  --|
  --| Notes
  --|    Turn_On_Engine must first be called.

  -- ................................................
  -- .                                              .
```

```
  -- .  Automobile_Interface.Hold_Accelerator_Pedal     .
SPEC
  -- .                                                  .
  -- ..................................................
  procedure Hold_Accelerator_Pedal;
  --| Purpose
  --|    The car stops accelerating and holds a steady
speed.
  --|
  --| Notes
  --|    Turn_On_Engine must first be called.

  -- ..................................................
  -- .                                                  .
  -- .  Automobile_Interface.Release_Accelerator_Pedal  .
SPEC
  -- .                                                  .
  -- ..................................................
  procedure Release_Accelerator_Pedal;
  --| Purpose
  --|    The car stops accelerating and starts to
  --| decelerate.
  --|
  --| Notes
  --|    Turn_On_Engine must first be called.

  -- ..........................................
  -- .                                        .
  -- .  Automobile_Interface.Depress_Brake_Pedal  .  SPEC
  -- .                                        .
  -- ..........................................
  procedure Depress_Brake_Pedal;
  --| Purpose
  --|    The car decelerates quickly.
  --|
  --| Notes
  --|    Turn_On_Engine must first be called.

  -- ..........................................
  -- .                                        .
  -- .  Automobile_Interface.Release_Brake_Pedal  .  SPEC
  -- .                                        .
  -- ..........................................
  procedure Release_Brake_Pedal;
  --| Purpose
  --|    The car decelerates at the same speed as
  --| it does if Release_Accelerator_Pedal is called.
  --|
  --| Notes
  --|    Turn_On_Engine must first be called.

  -- ....................................
  -- .                                  .
  -- .  Automobile_Interface.Sensed_Speed      .  SPEC
  -- .                                  .
  -- ....................................
  function Sensed_Speed return SPEED;
  --| Purpose
  --|    Return the speed of the car.

  -- ....................................
  -- .                                  .
  -- .  Automobile_Interface.Update            .  SPEC
```

```
   -- .                                          .
   -- ..........................................
   procedure Update;
   --| Purpose
   --|    Update the status of the car.  This routine
   --| should be called periodically, and it advances
   --| the state of the car to the next second.

end Automobile_Interface;
```

Project 3: Spacecraft Environment Monitoring System

A spacecraft contains a number of sensors which continuously sample the internal ambient temperature, pressure, and radiation levels within the spacecraft. The problem is to design and implement a monitor system in software which reads the values of these sensors, displays them (using character graphics and numeric values) on a CRT screen, and checks to ensure that the sensor values always fall within desired constraints. If any one sensor or combination of sensors return values which fall outside their constraints, an alarm indication will be displayed on the screen along with a beeping sound to attract attention.

For the purpose of this problem, an interface to the sensor hardware is provided by a sensor interface package. If this exercise were to be extended into a real application, the only difference would be a change to the package body of this sensor interface from a simulation to an actual hardware-level interface to the sensors themselves. The following is the package specification of the sensor interface (tentative, subject to change):

```
-- ************************************************
-- *                                              *
-- *  Spacecraft_Sensor_Interface                 *  SPEC
-- *                                              *
-- ************************************************
package Spacecraft_Sensor_Interface is

  type SPACECRAFT_SECTION is (BRIDGE, AUXILIARY_BRIDGE,
                              CREW_QUARTERS, GALLEY,
                              LAB1, LAB2, LAB3,
                              AIRLOCK1, AIRLOCK2,
                              EXPERIMENT_BAY);

  type PRESSURE is new FLOAT
      range 0.0 .. 400.0;          -- psi
  subtype TOLERATED_PRESSURE is PRESSURE
      range 20.0 .. 80.0;

  type TEMPERATURE is new FLOAT
      range -400.0 .. 2_000.0;     -- Fahrenheit
  subtype TOLERATED_TEMPERATURE is TEMPERATURE
      range -20.0 .. 120.0;

  type RADIATION_LEVEL is new FLOAT
      range 0.0 .. 8_000.0;        -- Roentgens
  subtype TOLERATED_RADIATION_LEVEL is RADIATION_LEVEL
      range 0.0 .. 400.0;

  -- ..............................................
  -- .                                            .
  -- .  Spacecraft_Sensor_Interface.Sensed_Value  .  SPEC
  -- .                                            .
  -- ..............................................
  function Sensed_Value
      (Location : in SPACECRAFT_SECTION) return PRESSURE;
```

```
  function Sensed_Value
      (Location : in SPACECRAFT_SECTION) return TEMPERATURE;
  function Sensed_Value
      (Location : in SPACECRAFT_SECTION) return
RADIATION_LEVEL;
  --| Purpose
  --|    Return sensor readings from different parts of the
  --| spacecraft.
  --|
  --| Exceptions (none)
  --| Notes
  --|     If values exceed the limits set for the different
  --| types, the corresponding maximum or minimum values are
  --| returned.  Assume that sensor validation is performed
  --| internally to these routines.

  -- ..............................................
  -- .                                            .
  -- .  Spacecraft_Sensor_Interface.Update        .  SPEC
  -- .                                            .
  -- ..............................................
  procedure Update;
  --| Purpose
  --|     Examine each sensor and update its status.  This
  --| update includes sensor input validation.
  --|
  --| Exceptions (none)
  --| Notes
  --|     Update must be called before calling any of the
  --| Sensed_Value routines.  Update checks all sensors, so
  --| the scenario is to call Update and then call all the
  --| permutations of the Sensed_Value routines before
  --| calling Update again.

end Spacecraft_Sensor_Interface;
```

Project 4: Document Concordance Generator

A concordance is an alphabetical index that shows the places in a document where each word may be found. For example, a concordance of this paragraph might appear as:

Word	Line Number
a	1, 2, 3
alphabetical	1
an	1
appear	3
...	
word	2

Concordances are typically used as an aid in the study of massive works, such as the Bible or the collected works of Shakespeare. In a slightly different form, a system that creates a concordance might be used to provide the functionality of a cross-reference generator for programs or to create an index.

This problem is to develop a program that, given the name of a file containing a document and the name of an output file, produces a concordance of the document in the output file. The concordance output file is to contain a heading at the top of each page, a footer at the bottom of each page with page numbers, and a title page showing the name of the input file. Articles (the words "a", "an", and "the") are to be

omitted from the concordance. Connectives (the words "and", "or", "then", and "else") are also to be omitted. Finally, other common words (including "of", "but", "is", "are", "not", "to", "that", "from", "it", "its", "itself", "in", "out", "very", "most", "it's", and "also") are to be omitted. Design the concordance software so this list of omitted words can be easily changed and the program recompiled.

Project 5: "Host at Sea" Buoy System

The "Host at Sea" buoy system is a group of free-floating buoys that provide navigation and weather data to air and ship traffic at sea. The buoys collect data on air and water temperature, wind speed, and their location through a variety of sensors. Each buoy is equipped with a radio transmitter (to broadcast weather information, location information, and an SOS signal) and a radio receiver (to receive requests from passing vessels). Each buoy is equipped with a yellow light, which can be activated by a passing vessel during sea-search operations. A sailor reaching the buoy can flip a switch on the side of the buoy which causes the buoy to send out an SOS broadcast and flash a red SOS light.

Software for each buoy must do the following:

1. Obtain air and water temperature and wind speed from the buoy's sensors and broadcast this information.
2. Obtain location information from the buoy's sensors and broadcast this information.
3. Activate or deactivate the yellow light based on requests from passing vessels.
4. Continuously broadcast an SOS signal and the red light after a sailor engages the emergency switch. This continues untilsomeone disengages the emergency switch. All other activity is taking place while the SOS broadcast is taking place.

You are to write a monitor system for all buoys in the world for this problem. This monitor system will display information about all buoys that are sending out SOS signals or flashing Yellow. Additionally, if the temperature of the air or the water exceeds 200 degrees Fahrenheit, this is an indication of a nuclear explosion, and such events must appear on the monitor with a warning alarm (beeping sound). Finally, if the wind speed at a buoy exceeds 50 MPH, a hurricane is assumed to be present, and this condition should also set off a warning alarm.

For the purpose of this problem, an interface to the buoy hardware is provided by a sensor interface package. If this exercise were to be extended into a real application, the only difference would be a change to the package body from a simulation to an actual hardware-level interface. The following is a listing of the package specification (tentative, subject to change):

```
-- ******************************************
-- *                                        *
-- *  Buoy_Sensor_Interface                 *  SPEC
-- *                                        *
-- ******************************************
package Buoy_Sensor_Interface is

  type BUOY_ID is (B01, B02, B03, B04, B05,
                   B06, B07, B08, B09, B10,
                   B11, B12, B13, B14, B15,
                   B16, B17, B18, B19, B20);

  type OFF_ON is (OFF, ON);

  type TEMPERATURE is new FLOAT
      range -100.0 .. 300.0;     -- Fahrenheit

  type SPEED is new FLOAT
      range 0.0 .. 200.0;        -- MPH
```

```
type NSEW is (NORTH, SOUTH, EAST, WEST);
type DEGREE is new FLOAT
    range 0.0 .. 90.0;         -- Degrees
type COORDINATE is record
  Direction : NSEW;
  Offset    : DEGREE;
end record;
type LOCATION is record
  Latitude  : COORDINATE;
  Longitude : COORDINATE;
end record;

-- ..........................................
-- .                                        .
-- .  Buoy_Sensor_Interface.Air_Temperature    .  SPEC
-- .                                        .
-- ..........................................
function Air_Temperature (ID : in BUOY_ID)
    return TEMPERATURE;
--| Purpose
--|    Return the Air Temperature around the buoy.
--|
--| Notes
--|    Update must be called to update the status
--| of the buoy sensors first.

-- ..........................................
-- .                                        .
-- .  Buoy_Sensor_Interface.Water_Temperature  .  SPEC
-- .                                        .
-- ..........................................
function Water_Temperature (ID : in BUOY_ID)
    return TEMPERATURE;
--| Purpose
--|    Return the Water Temperature around the buoy.
--|
--| Notes
--|    Update must be called to update the status
--| of the buoy sensors first.

-- ..........................................
-- .                                        .
-- .  Buoy_Sensor_Interface.Wind_Speed         .  SPEC
-- .                                        .
-- ..........................................
function Wind_Speed (ID : in BUOY_ID)
    return SPEED;
--| Purpose
--|    Return the Wind Speed around the buoy.
--|
--| Notes
--|    Update must be called to update the status
--| of the buoy sensors first.

-- ..........................................
-- .                                        .
-- .  Buoy_Sensor_Interface.Global_Position    .  SPEC
-- .                                        .
-- ..........................................
function Global_Position (ID : in BUOY_ID)
    return LOCATION;
--| Purpose
```

```
   --|    Return the location of the buoy.
   --|
   --| Notes
   --|    Update must be called to update the status
   --| of the buoy sensors first.

   -- ..........................................
   -- .                                        .
   -- .  Buoy_Sensor_Interface.Red_Light       .  SPEC
   -- .                                        .
   -- ..........................................
   function Red_Light (ID : in BUOY_ID) return OFF_ON;
   --| Purpose
   --|    Indicate if the red light is on or off.
   --|
   --| Notes
   --|    Update must be called to update the status
   --| of the buoy sensors first.

   -- ..........................................
   -- .                                        .
   -- .  Buoy_Sensor_Interface.Yellow_Light    .  SPEC
   -- .                                        .
   -- ..........................................
   function Yellow_Light (ID : in BUOY_ID) return OFF_ON;
   --| Purpose
   --|    Indicate if the yellow light is on or off.
   --|
   --| Notes
   --|    Update must be called to update the status
   --| of the buoy sensors first.

   -- ..........................................
   -- .                                        .
   -- .  Buoy_Sensor_Interface.Update          .  SPEC
   -- .                                        .
   -- ..........................................
   procedure Update (ID : in BUOY_ID);
   --| Purpose
   --|    Update the buoy sensors' status.

end Buoy_Sensor_Interface;
```

Project 6: PC Board Router

This project consists of developing a simple two-layer printed circuit board router to automatically determine shortest wiring paths between IC and component pads on the board. Assume the board component assignment has already been done, and a netlist consisting of a list of nets is given where each net *i* consists of n_i (x,y) points.

The following board physical constraints hold:

1. The maximum board size is six inches by six inches.
2. Wire width is 25 mils and pitch is 50 mils.
3. All pads have a diameter of 25 mils.
4. All component pads lie on a virtual grid with 100 mil spacing in x and y.
5. Wire direction is in either the veritcal or horizontal direction (manhattan corrdinate system).

The software product must:

1. Automatically route all valid wires.
2. Check for all errors in the netlist and board definition.
3. Display the routed board on the terminal screen as the board is routed.
4. Operate in one of two modes:
   a. fully automatic mode - read netlist and fully route displaying as execution progresses
   b. interactive mode - request two-point net from user, route it and display results on the screen. The interactive mode should also permit deletion of routed nets or any subnet, and display routing statistics such as % nets routed, sum of routes, etc.

Check Lists

Use these check lists to ensure your activities throughout the quarter are done properly and efficiently. There's a lot of good advice here!

Meeting Diagnosis

This checklist is used to assess the planning, execution and effectiveness of meetings. Source: Barbara C. and Kenneth R. Palmer, *The Successful Meeting Master Guide,*Prentice-Hall, Inc., 1983.

1. Meeting Goals and Objectives
   - Were the issues or topics of the meeting clear to all in attendance?
   - Did everyone know what decisions or actions were to be taken?
   - Were the meeting's objectives reasonable, given available time and resources?
   - Was a good sense of priorities established to guide deliberations?
2. Notification
   - Was the agenda made clear by the meeting organizer before the meeting (verbally or in writing)?
   - Were participants told of the meeting sufficiently in advance to allow adequate preparation?
   - Were they prepared to make decisions or act?
   - Were all necessary materials distributed sufficiently in advance to allow review?
   - Was the distribution of unnecessary materials avoided?
   - Was the time of the meeting convenient?
3. Participation
   - Were all key players present?
   - Were the interests of everyone impacted by the meeting outcomes epresented?
   - Was your contribution to or benefit from the meeting sufficient to warrantyour attendance?
   - Was sufficient time allocated to each item on the agenda?
   - Were all people given an opportunity and encouraged to participate?
   - Was a spirit of cooperation fostered?
   - Was domination of the group by one or two persons avoided?
   - Were people presenting information well prepared and organized?
   - Did the group adhere to the agenda or get side-tracked?
   - Was the meeting leader effective in instilling a sense of common purpose, eeping the group on track, creating an atmosphere conducive to the free exchange of ideas and information, resolving conflict, providing positive feedback when warranted, maintaining interest and enthusiasm, summarizing periodically and/or at the end of the meeting, and moving the group toward closure on agenda items?
   - Was the meeting concluded reasonably close to the scheduled ending time?
4. Meeting Follow-Up
   - At the close of the meeting, was there consensus on decisions or actions?
   - Were follow-up assignments and responsibilities clear?
   - Were minutes, follow-up correspondence, or actions executed in a timely manner?

Project Startup

This checklist is used to get the project off to a good start. These actions should be taken upon the first meeting of the project group. Source: Barbee Teasley Mynatt, *Software Engineering with Student Project Guidance,* Prentice-Hall, Inc., 1990.

1. Meet with your project group and decide on a team organization.
2. Select a leader for the group.
3. Set up a formal communication network for the group. For example, a telephone chain might be set up where each member is responsible for contacting the next member on the list. Also, a common electronic mailing list might be set up on theRm 800 SUN server and group members may agree to read their electronic mail every day.
4. Decide on meeting procedures: who will call the meetings, where the meetings will be held, who the meeting leader will be (for at least the next meeting), what the format of the meetings will be, and how long typical meetings will last.
5. Consider establishing a regular meeting time so that people can build their schedules around the regular time.
6. Choose a recorder to take minutes of meetings.

Requirements Analysis Activities

This checklist is used to ensure that all activities performed during requirements analysis are completed. Source: Barbee Teasley Mynatt, *Software Engineering with Student Project Guidance*, Prentice-Hall, Inc., 1990.

1. Set up a schedule of analysis activities and assign personnel.
2. Itemize the requirements of the system.
3. Each individual should review and make sure he understands the Project Description. If there are any questions, discuss them in the group before the meeting with the sponsor.
4. Meet with the sponsor. Make sure the meeting covers the following:
   - A meeting with the sponsor IS a meeting. Follow the procedures for a good meeting given earlier.
   - Be sure to take written notes and give the sponsor a copy of those notes after the meeting.
   - Review and discuss the Project Description.
   - Review and discuss the requirements of the system that your project group has identified.
   - Determine the who, what, where, when, why, and how of the project. Write down the answers.
   - Get copies of any forms, data, printouts, and the like that the sponsor uses in the current system.
   - If not already prescheduled, set up regular meeting times with the sponsor.
5. If the sponsor is not the user of the current system, interview or observe the activities and responsibilities of the people who are.
6. If the sponsor is not the user of the proposed system, interview the people who will be. Be sure to assess their knowledge and skill in using computers, and analyze their tasks and responsibilities in the system.
7. Write down an initial description of the system in the form of a statement of system scope.
8. Make a list of the objects in the system and identify their type.
9. Make a list of the operations in the system, and associate each operation with an object.
10. Create one or more Entity-Relationship Diagrams (ERDs) of the system.
11. Create one or more State Transition Diagrams (STDs) of the system.
12. Create one or more Flow Charts of the system.
13. Create one or more Data Flow Diagrams (DFDs) of the system.
14. Create a Data Dictionary (DD) of the system.
15. Study the Interleaf publishing system on the Rm 800 SUNs (or other sophisticated formating editor like Word for Windows or Wordperfect) and learn how to use it in to create the Software Requirements Specification (SRS).
16. Make sure you understand the stylistic guidelines and standards for the DFDs, ERDs, DD, Flow Charts, and SRS.

17. When your model (DFDs, ERDs, DD, Flow Charts, others) of the system is well-along, have the sponsor and other people from the current system examine and comment on your model.
18. Create one or more model versions of the proposed system and conduct reviews of these within the project group.
19. Choose the best model from among those proposed.
20. Assess the feasibility of the proposed system. Identify the areas of risk.
21. Determine the qualification criteria for the proposed system. Make sure to include definitions for the "success" and "failure" of the system.
22. Assemble and review the Software Requirements Specification document.
23. Review your requirements analysis within the project group before the formal Software Requirements Review (SRR) with the sponsor and the class. Plan the formal SRR (including the preparation of transparencies), allowing 5 minutes for questions and discussion.
24. Hand in the SRS to the sponsor.

Design Activities

This checklist is used to ensure that all activities performed during preliminary design are completed. Source: Barbee Teasley Mynatt, *Software Engineering with Student Project Guidance*, Prentice-Hall, Inc., 1990.

1. Schedule user interface design activities and assign personnel.
2. Assess the current system from the standpoint of the user interface. What aspects of the current system's interface are good or poor? What aspects of the proposed system's interface are good or poor?
3. Obtain information about the potential users of the system. Will they be novice, intermittent, or frequent users, or possibly a mixture? Will they be adults or children?
4. If possible, observe the potential users of the system. How do they approach their tasks? Interview them concerning the pros and cons of the current system.
5. Try out software systems similar to the proposed system and assess the user interfaces.
6. Define the semantics of the proposed system. Organize the tasks into categories. Consider task flow sequencing alternatives. Make a distinction between objects in the task environment and actions in the task.
7. Decide on the style or combinations of interface styles to be used. Consider alternative styles before selecting one. Make prototypes to help in reaching a decision.
8. Decide what other user aids might be needed, such as online help, tutorial systems, or offline tutorials. Begin planning their design and implementation.
9. Write a draft Software User's Manual (SUM) and review it within the project group.
10. Begin filling out the Software Design Document (SDD) and review it within the project group.
11. Plan a schedule or design activities and set deadlines.
12. Assign personnel to be responsible for the different activities.
13. Using the results of the requirements analysis activity, identify the essential objects of the system. Document their attributes, the operations performed by them, and the operations performed on them.
14. Create one or more Object Interaction Diagrams (OIDs) of the system.
15. Design the major data structures for the system.
16. Evaluate the modules of the system for coupling and cohesion.
17. Review your preliminary design within the project group before the formal Preliminary Design Review (PDR) with the sponsor and the class. Plan the formal PDR (including the preparation of transparencies), allowing 5 minutes for questions and discussion.
18. Update all diagrams and the Data Dictionary. Complete the design.
19. Evaluate the detailed designs by holding a number of walk-throughs and/or informal reviews within the project group. Have the recorder document the proceedings of these reviews and distribute them to group members.

20. Review your final design within the project group before the formal Critical Design Review (CDR) with the sponsor and the class. Plan the formal CDR (including the preparation of transparencies), allowing 10 minutes for questions and discussion.
21. Hand in the SDD to the sponsor.

Coding and Testing Activities

This checklist is used to ensure that all activities performed during coding and testing are completed. Source: Barbee Teasley Mynatt, *Software Engineering with Student Project Guidance*, Prentice-Hall, Inc., 1990.

1. Assess and study your tools. Is an Ada pretty printer available? Is a language-sensitive editor available? What other tools are available to help you?
2. Make sure you code according to MIL-HDBK-1804.
3. Assign personnel to coding and testing duties. Make sure that the person who codes a part of the design does not test that part of the design.
4. Set up a schedule of coding, integration, and testing activities.
5. Conduct code walkthroughs with the project group. Document the results of these walkthroughs.
6. Keep records of the unit testing done. Review those records for completeness.
7. Prepare the presentation for the delivery and live demonstration of the software.
8. Give a live demonstration of your system to the sponsor and the class. Be sure to have presentation aids ready to aid in the demo.
9. Hand in your code, SUM, and Version Description Document (VDD) to the sponsor. Also hand in a copy of all records kept on this project.