



## Performance Objectives:

1. **Learn and appreciate the basic issues involved in the development of complex engineering software systems**
2. **Learn tools and techniques for requirements analysis and design and apply it within the context of your chosen paradigm.**
3. Use automated tools (CASE tools), if necessary, for your class projects.
4. Develop a Software Requirements Specification (SRS) and a detailed Design Specification for a chosen project.
5. Learn to work in a group and contribute effectively as a team member.

## Course Policies / Guidelines:

1. Copying / cheating of any kind will result in a letter grade of F to everyone involved. However coordination and communication between all the students in class will be encouraged.
2. No unexcused absences will be accepted. Test dates will not be changed and make up tests will not be conducted for such reasons. However, if you had an emergency, I will make an exception with proper supporting documents.
3. You may have up to 3 students per group for projects &/or assignments.
4. All tests / assignments / projects should be typed, and the documents must be well prepared / documented. **You will not be graded for any loose sheets (including documents with clips).**
5. The deadlines indicate the amount of time and effort required both towards the completion of your assignment / project and its documentation. Do not anticipate to complete your assignments / projects and related documentation 2-3 days prior to the final deadlines. **I expect the group members to meet as often as necessary** (once in 2-3 days) and keep informed about the progress of other members in the group. Group projects will be evaluated on the functioning of the entire group and not based on individual accomplishments. Therefore, **be aware** of persons who might have a heavy course load and/or students who are not willing to spend the appropriate amount of time. **I will not entertain complaints about irresponsible group members just before the due dates!**
6. If appropriate at a later time, the instructor reserves the right to modify any of the dates / criterion for this course.
7. The course involves a lot of writing. **You are required to be responsible** and make all efforts to organize your thoughts and communicate effectively (within the team and in your reports). **If you do not take responsibility, your grades may be adversely affected.** Note that there are different organization styles for your documents and there is not a unique "correct" style. I suggest that you and your team discuss and agree on some standard document formats. If you seek timely help, I will make all efforts to comment on your style and organization of your documents.
8. Class notes / out of class reading assignments are essential for the class. You are required to read all such reading material. **Read ahead, not after every class / week, to fully understand and finish your work in time!**
9. All groups will present their design in front of the entire class. All students in class (includes your group's evaluation of your work) will evaluate the final presentation. You do not have to complete your documentation / coding, but it should provide you a chance to gather meaningful inputs. Any good suggestions must be included in your final implementation and the design document.

### Guidelines for Program Documentation

This guideline is for all files in your final implementation, not just the main file.

1. Program header: The program header must contain information about the program group, the scope and function of the entire program. It must identify the program inputs, the format of these inputs and the parameters that are acceptable as inputs with examples whenever appropriate. It must also provide information about the program outputs, their expected values / formats and their interpretation, with examples wherever appropriate. The locations of all include files and their use in the program must also be documented. Also, list all the input variables and their relationship with actual real world data - write what they represent in the real world.
  - Function: identify the functionality of the program and what it will accomplish
  - Variables: List all global and main variables, their necessity and what they represent
  - Inputs: List and provide examples of acceptable input formats and type for every input to the program
  - Outputs: List and provide examples of expected output formats.
  - Algorithm: Briefly discuss the algorithm.
2. Module documentation: All subroutines / procedures/modules in the program must also be documented using the guidelines above. Procedure names must be meaningful. Discuss any exception conditions for failures.
3. Testing: All programs must be thoroughly tested for any errors and all flaws must be documented and corrected.

## Essentials in Design Documentation: Guidelines / Criteria for Grading

The following is a tentative list of criterion to be used for the evaluation of your final project design documentation. If you are unsure of any of the following, **please get it clarified as soon as possible**. **This design documentation does not mean that your program will not be documented with appropriate comments as required**. An outline for design documentation is in your book (page 364) to guide you through the important sections of a design document. The test provisions section in the outline must be substituted with the outline in page 504. A modified and more detailed version will be given out and this must be included as part of your design document. You will also create a software requirements specification (or SRS - page 290) during the first half of the course.

1. Specification (20%): The program must meet all your specifications. This includes all modifications in functionality that we will discuss early in the quarter for every project group.
  - 20: If the project design and implementation meets all specifications correctly and completely
  - 10: If the design and implementation does not meet some (1 - 3) of the specifications
  - 0: If none of the specifications or a majority of the specifications (> 3) are not met.
2. Coding and Design (40%): The program solution / implementation strategy must be well thought of and not be just a quick implementation. (This includes proof of all implementation options, design tools / techniques to be used and other issues discussed in your group meetings and your choice for the implementation strategy). Each function / procedure in your design must be carefully documented with its inputs, outputs, input and output calling procedures, a process specification and a brief explanation of its functionality. If you choose to use an automatic code generation tool, you must be able to explain the code in your presentation and your final design document.
  - 40: If the project implementation (design solution and the code) is well documented, formatted and understandable both in the document and in the code - use of appropriate identifiers, indentation, etc.
  - 20: If the design and implementation (comments on the code is included here) is incomplete.
  - 0: If the code is not commented appropriately as required, or the program does not execute and has errors, or the documentation of the design is bad.
3. Testing (40%): The code must be fully executable and functional. The program must execute correctly, with no warnings / errors. It must be designed to be user-friendly in error messages thereby indicating possible UI errors. The code should use the language capabilities completely. (Note: A smaller code is not essentially a badly written code.). Any interfaces with the user must be well thought of and should be completely meaningful. Any possible user errors and logical errors must be carefully thought of in your design. Testing documentation should include evidence of individual function tests for meaningful inputs and outputs as well as integrated testing (testing interfaces with other modules and checking if the modules perform well when combined together)
  - 40: A complete testing procedure was done and appropriately documented. All errors found during the testing was integrated and the code modified appropriately.
  - 20: Incomplete testing / testing process, or the results of the testing process is not well documented.
  - 0: No testing was done and/or it was not reported in the design document.
4. Final design documentation due: **9:00 a.m. on the first day of final examination week.**

*Note: Any student who has special needs should contact Sammie Young, Director of the Disability Services in Room 307 at the University Center. Phone: (931)-372-6119.*

**Software Engineering Class Summary: Relevant Topics and their Interrelationships (as in CSC 370)**

**In addition, in this class, we'll look at a detailed introduction to formal methods, OO software analysis, design and testing concepts as and where appropriate.**

<b>Documents</b>	<b>Major Topics</b>	<b>Minor Topics</b>	<b>Additions</b>
	Paradigms	Lifecycle / Waterfall	
		Spiral	
		Prototyping	
		Formal Methods	
		CASE	
Project Proposal			
	Metrics	Size Oriented	KLOC
		Function Oriented	Function points
	Project Management	Timelines	Gantt Charts
		Scheduling	Pert/CPM, Cost Projections
	Requirements Analysis	Data Collection	Interviews, Questionnaires, Hard data collection
		Problem Analysis	What vs How, Want vs Need
		State based tools	Flowcharts, FSA, StateCharts, Petri Nets, OO methods
		Data Flow tools	DFDs, CFDs
		Data Oriented tools	ER models,
		View Point analysis	Warnier Orr method
Software Requirements Specification			
	Design	Software Architecture	Structure Charts, OO notations
		Program Design	PDL, Decision trees, decision tables, Other graphical tools
		Data Structure Design	Data dictionaries
		Issues	Coupling and Cohesion, Fan-in and Fan-out, OO issues
		Formal Specification	Algebraic Specification
Software Design Specification			
	Testing and Complexity	McCabe's Measure	Cyclomatic complexity, Simple OO measures
		McCall's Quality factors	Product operation, revision and transition factors
		White-box testing	Path testing using Cyclomatic complexity, logic testing of loops and conditions
		Black Box testing	Equivalence partitioning, Boundary value analysis, IO testing
		Other issues	GUI testing, verification and validation
		Test integration	Top-down, bottom-up and sandwich integration
		Debugging	Brute force, backtracking, induction and deduction approaches, Regression testing
Software Test Document			