CSSE 374 – Software Architecture and Design I

Scoring Rubric for Milestone 3

There are for areas that will need focus in grading this assignment:

1. System Sequence Diagrams (SSD) – the teams should provide SSDs describing the behaviors and events between the junior project "system" and key actors in the application domain. For some projects this will be more challenging than others. To grade this effectively, we look at how these diagrams capture key operations with relevant parameters. The event arrows should go to and from the :System with the appropriate arrow heads and line types (solid arrow heads and lines for synchronous events and stick arrow heads and dashed lines for asynchronous events). Note that there can be more than one actor working with a :System in the middle so long as the actors are outside the system. Sequence Diagrams (SD) describe these operations with classes within the :System as part of #4 below.
2. Operation Contracts (OC) – the teams detail key operations more formally in their analysis model (Chapter 11) using OC. Note that operations that need more detail (or can/have not be detailed in the use case) or have complex more constraints and interaction will have an OC. There should be the title, short description, cross references (optional, but typically include relevant Use Cases and SSDs), Preconditions, and Post-Conditions. Post-Conditions must be stated in the past tense and Chapter 11 and my class slides outline how these post-conditions should create/delete/update class instantiations and/or associations, and the like.
3. Logical Architecture – Using the analysis model elements (DM, SSDs, and OCs), the team should formulate the allocation of classes to packages based on guidelines from Chapters 12 & 13 of the text (allocate the packages to appropriate layers and partitions). Note that the primary focus is on the Domain Layer, but other layers like the UI and Technical Services layers should be present with typical affordances (e.g., persistence, directory, security, etc.). The team should indicate "key" dependencies between packages and/or classes in packages, and describe why they are there (either through note tags embedded in the model and/or in a textual description that follows the diagram). The textual description presents rationale and assumptions for the elements in the model as well as explanations where the model may be ambiguous or complex.

4. Interaction Diagrams (ID) – using relevant system operations the team should develop Sequence Diagrams (SD) and/or Communications Diagrams (CD) as appropriate, that model the key behaviors for Iteration 1 functionality showing the detailed messages and objects involved in implementing the operations (Chapter 15). Again, each diagram should have some textual description or embedded notes presenting rationale and assumptions for the elements in the model as well as explanations where the model may be ambiguous or complex.
5. Design Class Diagram (DCD) – the team should produce a set of DCD for Iteration 1 following the guidelines in the book (Chapter 16) and discussed in class. Note that this means progressing from the Domain Model classes into more detailed design classes that contain attributes and their respective type, operations, and have relationships between classes for dependencies, various associations, aggregations/compositions, generalizations, and the like. While aggregations/compositions and generalizations need not have labels, most of the others should have some labels indicating the association or dependencies.
6. Iteration 1 (initial working version of the system) –This core implementation should provide the basic infrastructure on which the teams build functionality in future iterations. For example, they may need user interface and database technologies in place, or may be using some open source components in their design that they will need to analyze and begin programming against. The teams need to identify a few basic elements of their domain to implement first and to demonstrate the use of the infrastructure.

   They will demonstrate their software for this first iteration at their first project meeting on or before Friday of 4th week. They may use their team SVN repository for source code control or some other version control system (e.g., git on github or Mercurial on code.google.com). If they choose another system, they must make sure the instructor and project manager are able to access the code.
7. The models and information should be communicated in a way that a reasonably knowledgeable software engineer could understand what the models are communicating. Hence, presentation or polish is important – not necessarily pretty, but complete, unambiguous, and comprehendible. Further, the information between the models should be relatively conflict free.

Excellent work (A) would include a large segment of the things listed above. Major points are taken for one of the first six items missing or largely

incomplete. Single points are taken for somewhat incomplete models, misunderstanding the use of UML in modeling over multiple situations, or sloppy representations. Fractions of points (½ , ¼) are taken for individual or minor problems found. The eighth element listed above on polish and comprehensibility can have a deduction of 2 to 5 points depending on how egregious the infraction.

While the absence of the above will result in deductions, so will lack-luster performance on presenting cogent work. If the descriptions are haphazard, then there should be assessed accordingly.

The homework is graded from 0 to 100, with:

90-100 points earned for an	A (superior or excellent work),
80-89 points earned for a	B (very good work),
25-29 points earned for a	C (reasonable work),
20-24 points earned for a	D (poor work), and
0-59 points earned for an	F (unacceptable or very poor work).

Scoring Rubric for Milestone 5

Criteria (weight)	5 Exemplary	3 Satisfactory	1 Needs Improvement	Weighted Score
Professionalism (×2)	Document is neatly drawn and formatted. (Apart from any problems with the notation) it could be shared with a stakeholder without changes. Document is free of errors in spelling, grammar and punctuation.	Document is somewhat sloppy, but could be shared with a "real-world" stakeholder after some revisions. Document has a small number of errors in spelling, grammar, or punctuation.	Document is largely unprofessional. It would have to be largely reworked before sharing the document with a savvy stakeholder. Document has many errors in spelling, grammar, and punctuation.	
Cohesiveness (×1)	The parts of the document reinforce each other. Each piece is consistent with the others and the document as a whole tells a story.	The parts of the document mostly reinforce each other. Each piece is generally consistent with the others with just a few minor differences.	The parts of the document are disjointed. They are largely inconsistent, to the point that it is unclear whether they describe the same system.	
Clarity of Diagrams (×2)	Diagrams are well labeled and at an appropriate level of abstraction so that stakeholders familiar with the problem domain could readily understand them.	Diagrams are mostly well labeled, with no more than 15% cryptic labels. Diagrams are generally at an appropriate level of abstraction, though a stakeholder familiar with the problem domain might need some guidance to understand them.	Labels are often cryptic or abstraction is used to the point that the actual analysis and design implications would be obscured to all but an expert in both the notation and the domain.	

Conciseness of Diagrams (×1)	Diagrams appropriately use the abstraction features of the notation to minimize useless redundancy	Diagrams may include some unhelpful redundancy, but the general representations are still readily comprehensible	Diagrams are highly redundant to the point that they are difficult to comprehend.	
Effectiveness of Analysis (×3)	Analysis artifacts identify all important domain concepts and clearly define the system interface. They demonstrate a deep understanding of the problem domain.	Analysis artifacts identify many important domain concepts and define the system interface. They demonstrate a reasonable understanding of the problem domain.	Analysis artifacts identify only a few of the domain concepts or only cursorily define the system interface. They betray a superficial understanding of the problem domain.	
Effectiveness of Design Models (×3)	Design conveys all important elements, constructs, and behaviors. It demonstrates a deep understanding of the solution to the problem.	Design conveys many key elements, constructs, and behaviors. Some situations might be treated in an unusual manner, but such treatment is documented.	Design minimally conveys key elements, constructs, and behaviors. It shows a superficial understanding of the problem and its solution.	
Correctness of Solution (×3)	The design is viable within assumptions and rationale presented. Key tradeoffs are successfully analyzed and defended.	The design is largely viable within assumptions and rationale presented. Key tradeoffs are presented, but may not be fully or clearly analyzed.	The viability of the design is questionable. Some assumptions and rationale lacking. Key tradeoffs are missing or may be poorly analyzed.	
Correct Use of Notation (×2)	All notation used in the diagrams is appropriate to the diagram type and is used correctly.	All notation used in the diagrams is appropriate to the diagram type. At most two sorts of errors are made in the application of each diagram type.	Diagrams use notation inappropriate to the diagram type or contain a large variety of errors in the application of the notation.	
Software Demonstration (×3)	Software demonstration illustrates key foundational infrastructure elements like database, GUI, and security features as they might pertain to the system under development. The few selected features of the system were covered in a compelling way that made clear how the problem was being solved from the user's perspective.	Software demonstration illustrates some foundational infrastructure elements like database, GUI, and security features as they might pertain to the system under development. At least two selected features of the system were covered indicating reasonably well how the problem was being solved from the user's perspective.	Software demonstration illustrates only a few foundational infrastructure elements like database, GUI, and security features as they might pertain to the system under development. The selected features of the system were not covered well enough to indicate how the problem was being solved from the user's perspective.	
			Subtotal Score (Sum of above):	
			× (% of Assignment Completed):	
			= Total Score:	