CSSE 374 – Software Architecture and Design I

Rubric for Milestone 4

1. The teams should update their Domain Model, System Sequence Diagrams, Operation Contracts, Interaction Diagrams, Logical Architecture, and Design Class Diagrams from Milestone 3 based on the feedback provided on the paper. There is an implicit expectation that they would continue to refine their analysis and design models to reflect emerging design decisions. This represents 30% of the grade for the overall assignment.
    a. 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, you should 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. The System Diagrams will describe these operations with classes within the :System as part of #4 below.
    b. Operation Contracts (OC) – the teams detail key operations more formally in their analysis model (Chapter 11) using OC. There should be the title, short description, cross references (for 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.
    c. 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. 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 (e.g., incorporated appointment in schedule package since a schedule consists of appointments).

d. 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 <u>showing the detailed messages</u> and <u>objects</u> involved in implementing the operations (Chapter 15). Again, each diagram should have some <u>textual description or embedded notes</u>.

e. 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, 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.

2. The team should identify as many of the 9 GRASP principles (Low Coupling, High Cohesion, Information Expert, Creator, Controller, Polymorphism, Indirection, Pure Fabrication, and Protected Variations) as possible in their design and describe how they are used to arrive at their design. If they have made the design changes based on the tradeoffs presented they get full credit. If they just describe what they would do, they get 80% credit. They should reference their DCDs and interaction diagrams. The objective of this task is to compare their design to alternatives and reason for a selection that improves their solution. This represents 40% of the grade for the overall assignment.

   Iteration 2 (a 70-90% functional working version of the system) – the teams must build upon their work from Milestone 3 by implementing classes for their domain layer. They should follow the guidelines from Ch. 20 to transition their designs into code. Note that there is no requirement to build out the user interface here – again the focus is on the domain layer functionality of their architecture.

   The students have two options for testing their domain layer code:

   1. Test-driven development to create unit tests for their domain layer code as they develop it, or

   2. Implement a rudimentary UI layer to exercise their domain layer code.

   We strongly encourage the team to choose the first option if it is reasonable for their project. However, we recognize the each project is unique.

3. The teams will demonstrate their software for this second iteration at their first project meeting on or after Friday of 7th week or shortly there after (determined by team and instructor). 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. This represents 30% of the grade for the overall assignment.

As always, they should provide accompanying text and/or embedded notes indicating what they did in your modeling where it is not clear what they have conveyed using the diagram. 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.

The assignment with all of the above should be turned in as a single **pdf** file [named *RefinedDesign.pdf*]. Ten points (10%) of the grade can be deducted if they did not name or submit the file correctly to SVN.

Excellent work (A) would include a large segment of the things listed above. Major points are taken for one of the key task items missing or largely incomplete. **Use the table below for grading the overall document after leaving comments in document for recommended improvements.**

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),
70-79 points earned for a C (reasonable work),
60-69 points earned for a D (poor work), and
0-59 points earned for an F (unacceptable or very poor work).

Criteria (weight)	5 Exemplary	3 Satisfactory	1 Needs Improvement	Weighted Score
Professional-ism (x 2)	Document is neatly drawn. (Apart from any problems with the formalism) it could be shared with a stakeholder without changes.	Document is somewhat sloppy, but could be shared with a “real-world” stakeholder after some revisions.	Document is largely unprofessional. It would have to be largely reworked before sharing the document with a savvy stakeholder.	
Clarity of Formalism (x 3)	Diagrams are well-labeled and at an appropriate level of abstraction so that stakeholders fam-iliar with the prob-lem domain could readily understand the design.	Diagrams are mostly well-labeled, with 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 the design.	Labels are often cryptic or abstraction is used to the point that the actual design implications would be obscured to all but an expert in the notation.	
Conciseness of Formalism (x 3)	Design uses appro-priately the abstrac-tion features of the notation to minimize useless redundancy	Design may include some unhelpful redundancy, but the general design representations are still readily comprehensible	Design is highly redundant to the point that compre-hension of the design very difficult.	
Effective-ness of Design Models (x 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.	
Effective use of GRASP (x 3)	All 9 GRASPs are used correctly along with cogent descriptions outlining the tradeoffs and rationale.	Most of the 9 GRASPs are used correctly along with reasonable descriptions outlining the tradeoffs and rationale.	Some of the 9 GRASPs are used correctly along with questionable descriptions outlining the tradeoffs and rationale.	
Correctness of Solution (x 3)	The design is viable within assumptions and rationale presented. Key tradeoffs are suc-cessfully analyzed and defended.	The design is largely viable within assumptions and rationale presented. Key tradeoffs are presented, but may be fully or clearly analyzed.	The viability of the design is question-able. Some assum-ptions and rationale lacking. Key trade-offs are missing, and may be poorly analyzed.	
Correct Use of Notation (x 3)	All notation used in the diagram is appropriate to the diagram type and is used correctly.	All notation used in the diagram is appropriate to the diagram type. At most two sorts of errors are made in the application of the notation.	Diagram uses notation inappro-priate to the diagram type or contains a large variety of errors in the application of the notation.	

Subtotal Score (Sum of above / 10):	
(Subtotal Score) X (% of Assignment Completed):	
Total Score:	