CSE/ISE 308	**Software Engineering**	**Handout E1**
Spring 2010	**Annie Liu**	**April. 16, 2010**
Stony Brook	**Preparation for Exam**	

When and where: Friday, April. 23, 2010, in class (80 min).

What to bring: You may bring a one-page, hand-written, personal "crib sheet"; you may write on both sides of the sheet if you like.

Materials covered: You are responsible for materials covered in lectures and assignments before the exam. They are best summarized as on the course webpage.

Course goals: (1) Introduce models of software development and methodologies for project planning, requirements analysis, and system/test design. (2) Provide experience in working as a team to produce software systems that meet specifications while satisfying an implementation schedule. (3) Train students to produce professional quality oral/written presentations of system designs, reviews, and project demonstrations. (4) Expose students to ethical issues in software design and computing in general.

Sample Problems

1. All problems in quiz 1, quiz 2, quiz 3, and quiz 4. Make sure you can do all of them.

2. What is common underlying very different development process concepts: feature-driven development, test-driven development, and use-case-driven development?

3. What is UML? Name two advantages of using it.

4. What diagrams are for modeling structures? What diagrams are for modeling behaviors?

5. How can one make use of nouns in the requirement description?

6. What is a template class? Give an example.

7. How is a transition labeled with `event[guard]/action` taken?

8. Give three examples of design patterns. Give three example types of software architectures.

9. Contrast black box testing with white box testing.

10. Similar as above, you will need to know how to answer questions about other basic concepts, as listed below.

11. Consider developing a campus classroom facilities management system.

    The campus has a collection of classrooms. Each classroom has a collection of equipments, which may include blackboard, microphone, overhead projector, Internet connection, etc. Each equipment has information such as history, condition, available forms (built in or for picking up somewhere), as well as information specific to the equipment, such as size for blackboard, mobile or not for microphone, and speed of connection to Internet.

    Each classroom has a list of courses scheduled to take place in it, the instructor for each course, requested equipments and setups for each course, and a manager in charge of setting up the equipments.

    The system should allow instructors to look up the list of equipments in a classroom, request equipments and setups, and query the setup status. The system should send request for equipments and setups in a classroom to the classroom manager. The manager can update equipment information and setup status. In case of an equipment can not be setup according to the request, the manager should notify the instructor as soon as possible. The system should also allow anyone to send comments to the manager.

    Draw class diagrams for this system. Draw use case diagrams for this system.

12. Similar as above, you will need to know how to draw other UML diagrams from more detailed descriptions. You will also need to know how to write invariants and pre and post conditions in OCL from given descriptions.

13. Give three of the most important classes in your group project and show how they are related in a class diagram.

14. Similar as above, you will need to know other aspects of your group project.

15. Given an example of an ethics requirement in software development.

Basic concepts and methods

```
Overview:
  software engineering, state of the art, special difficulty;
  modular design and reuse, incremental and iterative development, use of tools

Software process models:
  development phases: requirement specification, design,
  implementation and testing, delivery and maintenance;
  process models: sequential, iterative, modeling and specification driven;
  waterfall, agile, extreme programming, unified process

Project planning:
  people involved, system to build, development tasks, development team;
  scheduling, effort estimation, product quality;
```

Requirement specification:
  requirement gathering and use case modeling: capture ‘‘what’’;
  use cases, use case diagrams, use case text, system sequence diagrams;
  requirement analysis and domain modeling:
  concepts for responsibilities, associations and attributes.

Design:
  decompose responsibilities, refine interactions: introduce ‘‘how’’;
  dynamic interactions, design sequence diagrams;
  static structures, class diagrams;
  design principles: nothing too big, maximizing sharing, caching;
  design patterns: structure and behavior, observer pattern, other patterns;
  patterns for concurrency, distribution, security;
  frameworks and architectures: MVC

Implementation:
  coding: clarity, efficiency; generate code from models; reuse existing code;
  forward, backward, and round-trip engineering;
  object relational mapping, persistent objects;
  object queries and updates, expensive queries;
  optimization by incrementalization, maintaining invariants

Testing:
  faults; test stub, test driver; black-box, white-box;
  unit test: choosing test cases, test coverage; code review, proving code;
  integration test: bottom-up, top-down, others;
  system test: functional and non-functional; acceptance test;
  regression testing; testing tools; when to stop

Delivery:
  deployment diagram, training and manuals

Modeling and specification:
  individuals and relations, states and state variables, events;
  state diagrams, composite states; activity diagrams;
  interface specifications, design by contracts,
  class invariants, method pre and post conditions;
  object constraint language (OCL), collections and comprehension

Measurements and estimation.
  measurements: use case points; SLOC count, cyclomatic complexity;
  effort estimation; believe it or not

Main UML and OCL concepts:

use case diagrams
actor, use cases, main flow of events, exceptional flow of events
relationships: include, extend, generalization, initiate, participate

class diagrams, object diagrams
objects, classes, attributes, operations, responsibilities
attributes and operations: type, signature, visibility
relationships: association, generalization, dependency
association: role, multiplicity, aggregation, composition
abstract classes, interfaces, template classes

interaction diagrams, sequence diagrams
object lifeline, focus of control,
create and destroy, call and return, branching and iteration

state diagrams
events, states, transitions
states: initial, final, activities
transitions: source and target states, event[gruad condition]/action

activity diagrams
activities, sequencing (transition)
branching (branch and merge, guard condition), concurrency (fork and join)
swimlane

other
stereotypes, notes, packages, component diagrams, deployment diagrams

constraints
contracts, invariants, pre and post conditions
collections: sets, sequences, bags
operations on collections: comprehension, quantifiers