Jeff Palmerino

SWEN 745 Midterm Review

**I. Software Modeling**

- Why?

* Schedule and divide tasks
* Collaboration and communication
* Decompose complexity and check for software quality
* What?
  + Overall architecture
  + System dependencies
  + Complexity, flow of information
  + Database organization and structure
* How?
  + Manually construct
  + Automatically transform into code
* Abstraction
  + The essential characteristics
    - Only show what is important, leaves our unnecessary information
    - Highlight the important stuff
    - Domain/perspective dependent
      * What’s important in one domain might not be in another

**II. UML**

* Diagrams
  + Structural – relations of objects
    - Class, component, deployment diagrams
    - Deployment Diagram
      * Capture computing hardware (nodes, devices, comm) and which elements are deployed
  + Behavioral – sequence of actions
    - Activity, sequence, state diagrams
* Views
  + Different concepts can be expressed in diagrams, i.e. *views*
    - Design – class, structure
    - Deployment – configure, install
    - Implementation – state, interaction, sequence
    - Process – performance, run-time behavior
    - Use case – requirements, system flow
* Extensibility Mechanisms
  + Allow modelers to refine semantics of UML for specific domain
    - Extensions can’t violate standard UML semantics, prevents meta-model explosion
  + Basic types
    - Stereotypes – used to refine meta-classes by redefining supplemental semantics
    - Constraints – predicates that reduce the semantic variation
    - Tagged values – individual modifiers with user defined semantics
* UML English
  + Dependency -> uses
  + Association -> is associated with
  + Aggregation -> has part of, is part of
  + Composition -> is composed of, is a necessary part of
  + Generalization -> is a, is a kind of

**III. Modeling**

* Domain model
  + Conceptual model of all the topics related to a specific problem
    - Considered effective if its models make sense to user who is familiar with the domain
* Maturity Levels
  + Level 0 – specification of software is not written down
  + Level 1 – textual specific, software is specified by a natural language
  + Level 2 – text with models
  + Level 3 – models with text, written down in one or more models, text is used to explain details
  + Level 4 – precise models, enhanced level 3
  + Level 5 – models only, price and detailed enough to allow code generation
* Meta-model
  + Typically defines the language and process from which to form a model
    - Model of another descriptive model
* Encapsulation
  + The physical localization of features in a single black-box abstraction that hides their implementation (i.e. private variables)
    - Two kinds of protection
      * Protects client code from changes
      * Protects internal state from corruption by its clients
* Polymorphism
  + The ability to hide may different implementations behind a single interface
  + Interface
    - A named set of operations that characterize behavior of a class, component or subsystem
      * Specify, not implement behavior
      * Different classes can realize (implement) same interface
* Package
  + General purpose mechanism for organizing elements into groups
* Subsystem
  + A grouping of elements that represent behavioral units in a physical system
  + Realizes one or more interfaces which define its behavior
* Component
  + A physical packaging of elements
    - A modular, deployable, replaceable part of a system that encapsulates implementation and exposes a set of interfaces
    - Physical realization of an abstraction in the design
  + Standard stereotypes
    - <<executable>> - program that can run on any code
    - <<file>>
    - <<library>>
    - <<document>>

**IV. Relationships**

* Collaborations and Interactions
  + Description of behavior has two aspects:
    - Structural decomposition of the participants playing roles in the task, *collaboration*
    - Description of communication patterns among participants, *interaction*
* Use cases
  + Capture the intended behavior of the system
  + Concepts:
    - Actor – primary, secondary, external
    - Generalize, include, extend
      * Include – base use case explicitly incorporates the behavior of another at a location specified in base use case
      * Extend – base use case implicitly incorporates behavior of another at a location specified indirectly by extending use case
      * Generalize – child inherits behavior and meaning of parent

**V. Task Analysis**

* Task specific description of a complete action that a specific user performs
* Process of analyzing and documenting how people perform their jobs
* Used to understand:
  + Workflow
  + Division of responsibility
  + Objects user will manipulate
  + Tasks user will perform
* Represented by:
  + Annotated task lists
  + UML activity diagrams
  + Task description template