

RUP Design Workflow

Michael Fourman

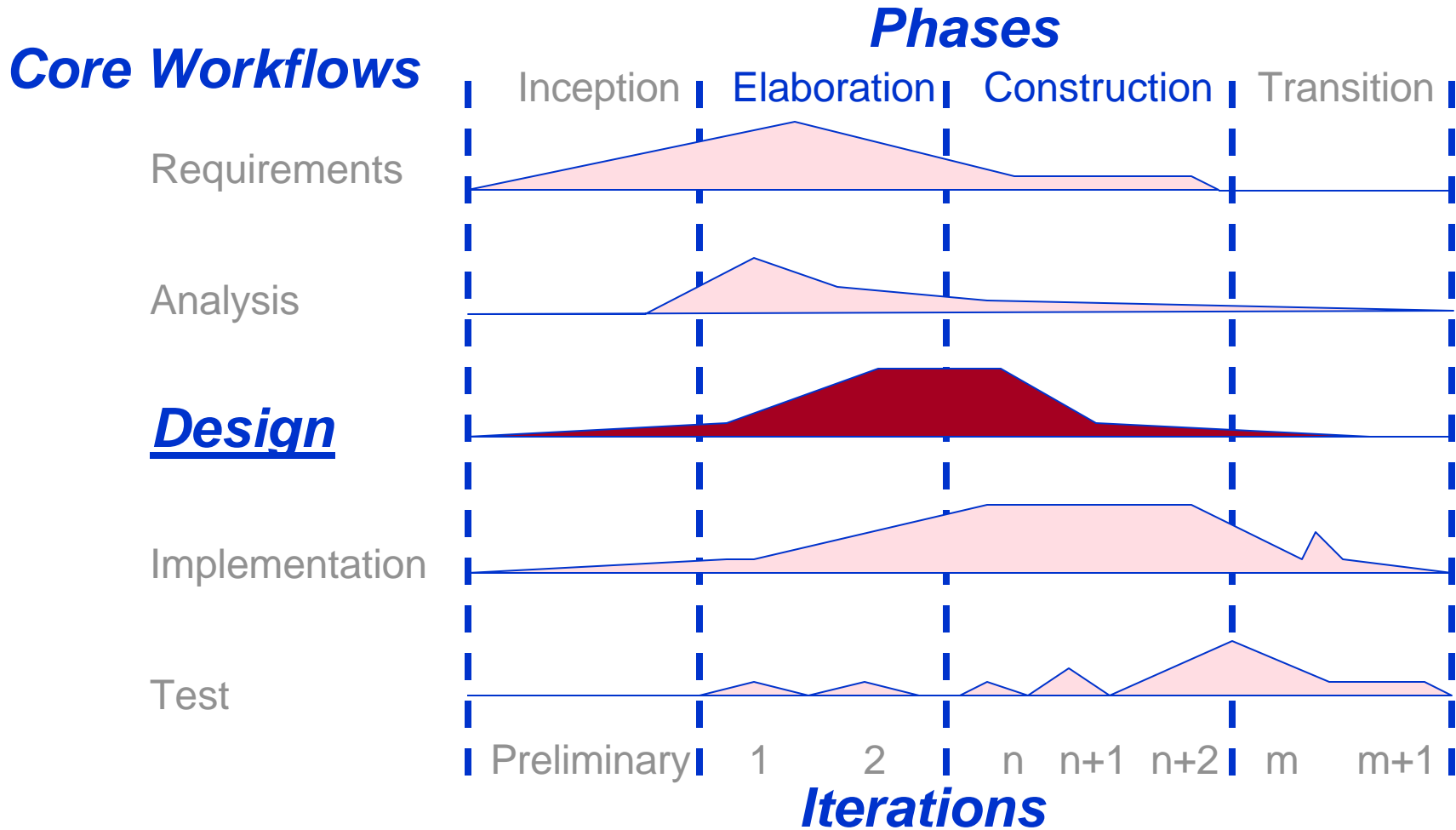
2001-10-27

Cs2 Software Engineering

Introduction

- Design architecture that can meet all requirements
- Understand non-functional requirements and constraints related to technologies
- Identify subsystems (overall structure, requirements, interfaces, classes) enabling concurrent implementation
- Create a seamless abstraction of the system implementation
 - Implementation adds content to a stable architecture
- Provides a visualization of the implementation

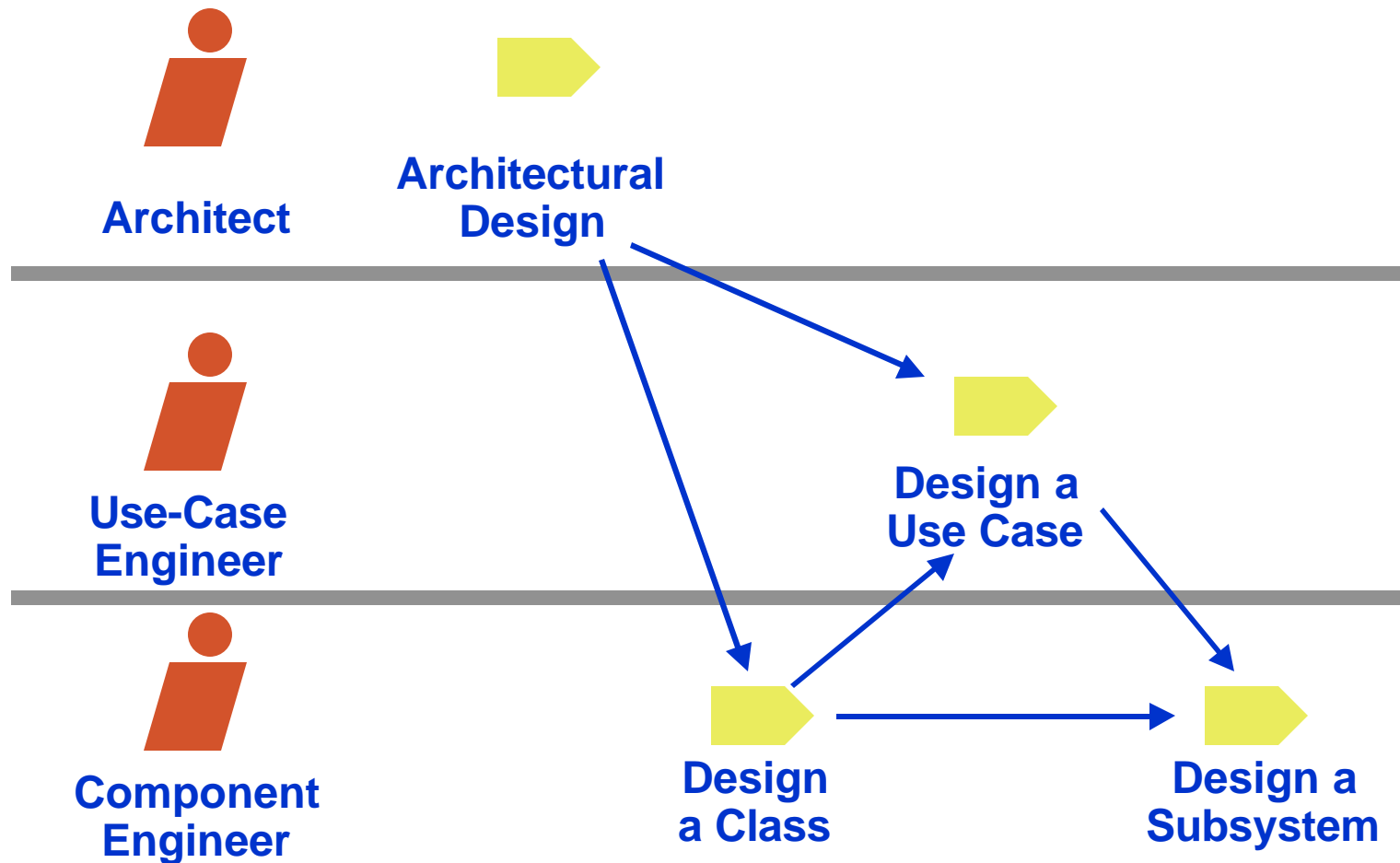
Design Focus



Artifacts

- Design Model
 - Design Classes
 - Use-Case Realization - Design
 - Class Diagrams
 - Interaction Diagrams
 - Flow of Events - Design
 - Implementation Requirements
 - Design Subsystem
 - Service subsystem
 - Interface
- Deployment model
- Process model
- Architecture Views
 - Design Model
 - Deployment Model
 - Process Model

Design Workflow Activities and Workers



2001-10-27

Cs2 Software Engineering

Design Model

- Hierarchy of design (sub)Systems containing Design Classes, Interfaces, and Use-Case Realizations - Design
- Object model describing physical realization of use cases
 - Design system and subsystems and classes that are abstractions of implementations
 - Use-case realizations (design-level)
 - Class and subsystem interface definitions
 - (these are described, next)
- Distinguish implementation technology-oriented design model from requirements-oriented analysis model

Class Design 1

- Specification language is the programming language
- Visibility (e.g., *public*, *protected*, *private*) of attributes and operations
- Type and signature of attributes and operations
- Methods (realization of operations) are in natural language or pseudo-code
 - Typically Specified only when implementer is not the same person as developer
 - Postpone the handling of some requirements by noting them as implementation requirements

Class Design 2

- Relationships mapped to implementation mechanisms
 - e.g., generalization --> inheritance or delegation
 - e.g., associations and aggregations --> variables and references
 - select one-way or two-way navigation
- Design class stereotype mapped to implementation
 - e.g. <<module>>, <<class>>, <<form>>, <<user control>>, etc.
- Interface classes and classes that realize the interfaces (e.g. in Java, CORBA, DCOM)

Realizing Use-Cases 1

- Collaboration of design objects that realize a use case (traced through use-case realization -- analysis)
- Class diagram: object operations, attributes and associations only as relevant to the use-case realization
- Interaction diagrams: sequence of internal subsystem/object actions in response to actor invocation of use case
 - Realize a specific flow or scenario in a given context (initial state)
 - Typically use sequence diagrams (instead of collaboration diagrams) for detailed timing and flow of control
 - Messages between object or subsystem lifelines
 - Level of detail at sub-system interface level and/or object operation level
 - Allows high-level design early, detailed design later

Realizing Use Cases 2

- Flow of events: textual description that explains relationship between multiple or complex interaction diagrams
 - Can describe how multiple interaction diagrams are related
 - Explain in terms of objects but not operations, attributes or associations (too much detail to maintain)
 - Provide details that are referenced (via labels) in interaction diagrams
 - Put the clutter in the textual description, not the diagram
- Implementation requirements: new or derived requirements discovered in design and addressed in implementation

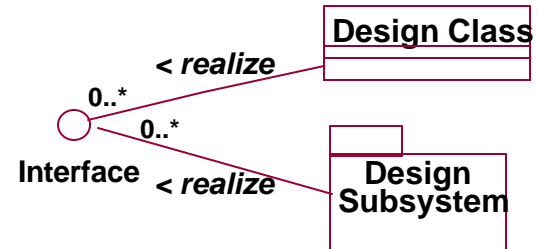
Subsystem Design

- Organize design model artifacts into more manageable pieces
 - A cohesive organization of classes, use-case realizations, interfaces and other subsystems
- Loosely coupled with other subsystems
- Separation of concerns
- Application layers often map to analysis packages
- Infrastructure layers provide technology services
 - Communications, persistence, security, etc.
- Can represent large-grained components and multiple interfaces that manifest themselves as executables deployed on different nodes
- Can represent and wrap reused software, legacy software or purchased third-party software

Application (Domain)

Infrastructure

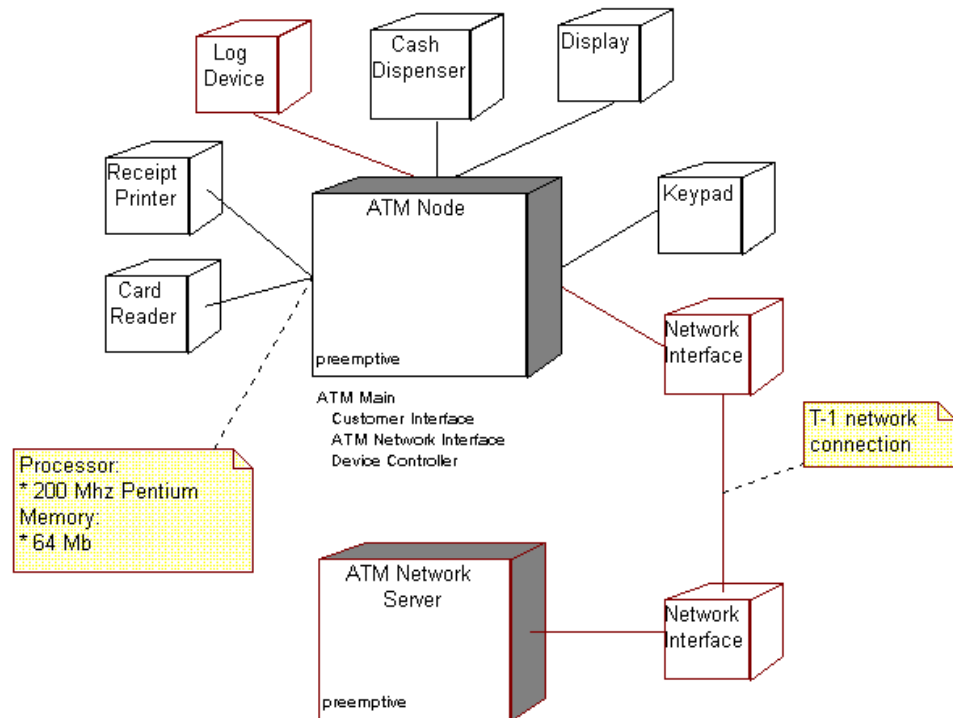
Interface



- Interfaces specify the operations provided by design classes and subsystems
 - A design class must have methods that realize interface's operations
 - A design subsystem must contain design classes or other subsystems that provide the interface
- Interfaces separate functional specification (operations) from implementation (methods)
 - Clients refer to interfaces
 - Can substitute implementations without changing client
 - Allows interface design early (architecture), implementation design later

Deployment Model

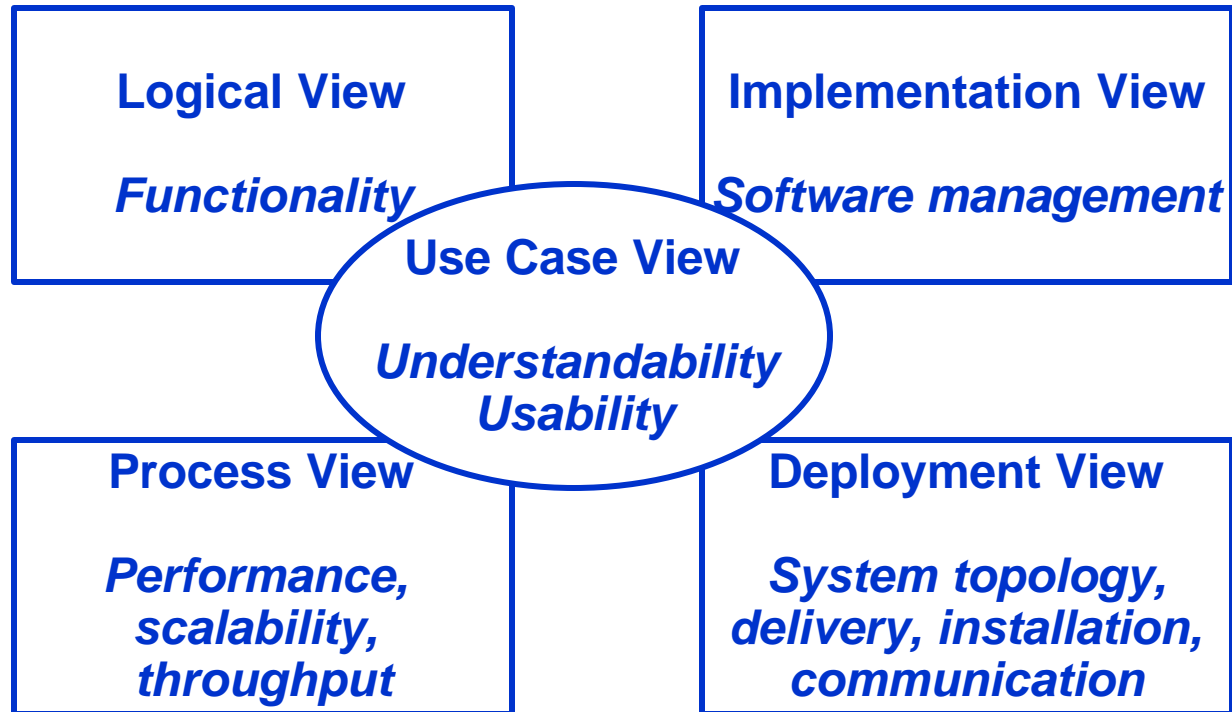
- Object model describing physical distribution of system functionality to computation nodes
- Node: computation resource
- Relationships: communication
- Describe alternate network configurations
- Map functionality to nodes (subsystems, components, processes)
 - Maps between software architecture and hardware architecture



Architecture Description

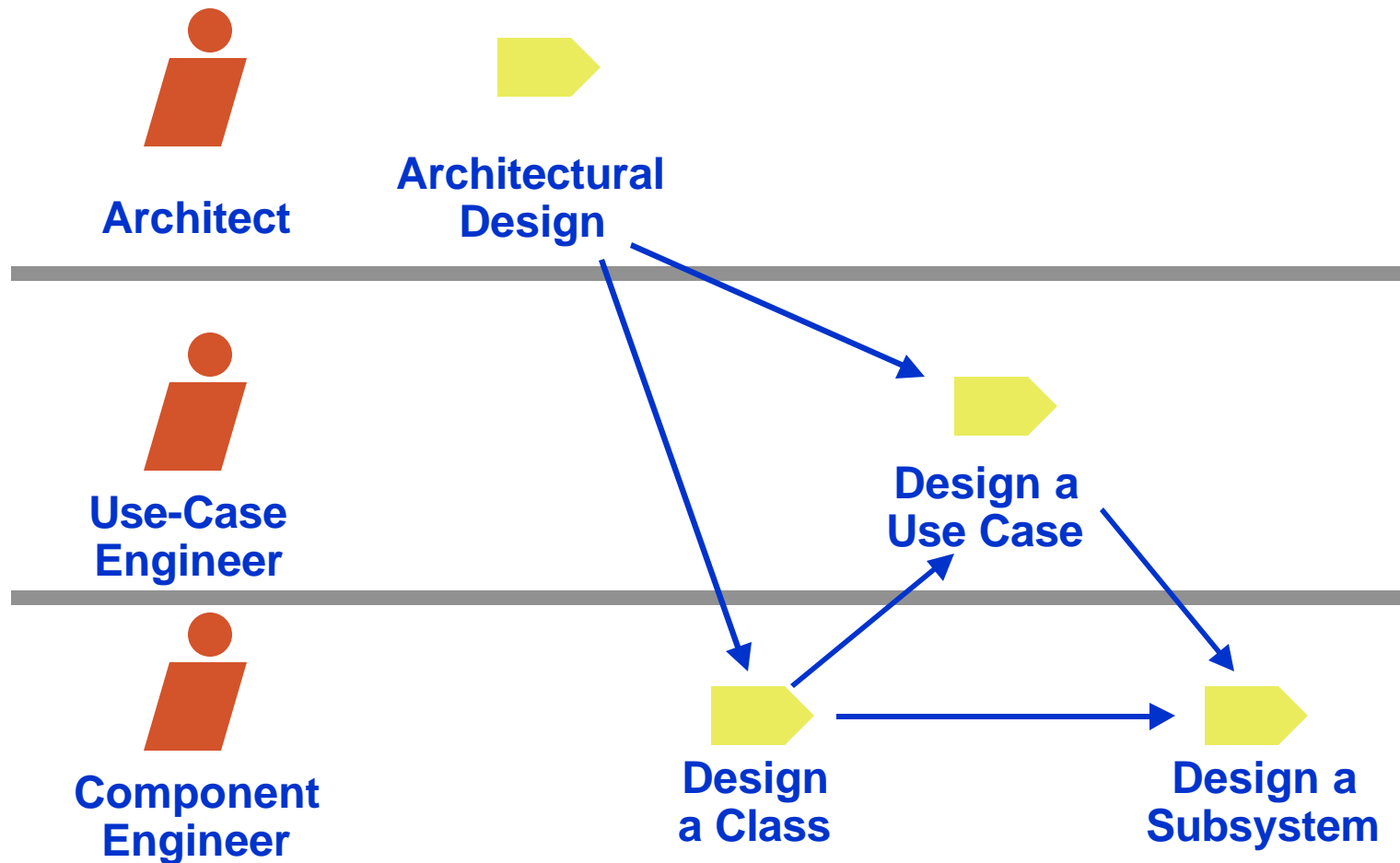
- View of design model (Logical View)
 - Decomposition of design model into subsystems, interfaces and dependencies
 - Fundamental system structure
 - Key design classes
 - Trace to architecturally-significant analysis classes
 - Active classes
 - Design classes that are general or central
 - Use-case realizations that are important, involve many design classes or key design classes, or cross subsystems
- View of deployment model
- View of process model (multiple active design objects)

4+1 View of Architecture



- See Phillippe Kruchten, *The 4+1 View Model of Architecture*, <http://www.rational.com/products/whitepapers/350.jsp>

Design Workflow Activities and Workers



Workers

- Architect, Responsible for:
 - integrity of design and deployment models (correct, consistent, readable)
 - architecture views of design and deployment model
- Use-case engineer
 - Responsible for integrity of one or more use-case realizations, assuring they realize the behavior that fulfills the requirements
- Component engineer
 - Defines and maintains operations, methods, attributes, relationships and implementation requirements of one or more design classes
 - May maintain the integrity of one or more subsystems
 - Often the implementer in later workflow

Workflow Details

- The following slides highlight some of the activities and practical guidelines to carry out the workflow

Architectural Design (1 of 2)

- Identify
 - Nodes and their network configurations
 - Subsystems and their interfaces
 - Architecturally significant design classes
 - Especially active classes
 - Generic design mechanisms that handle common requirements
 - Infrastructure services
 - “Middleware”
 - Common domain services
 - Application frameworks
- Consider styles, patterns and implementation reuse

Architectural Design (2 of 2)

- Identify nodes and network configurations
 - Nodes: number and processing power
 - Connections: protocols, bandwidth, quality, etc.
 - Fault tolerance: redundancy, fail-over, migration, data backup, etc.
- Identify subsystems and their interfaces
 - First iteration is analysis package --> subsystem
 - Refine for shared functionality, reused/wrapped legacy software, load balancing, etc.
 - Define dependencies and layers
 - Define interfaces to serve dependencies
 - When designing use cases in terms of subsystems and interfaces, identify operations on each interface
- Identify generic design mechanisms: persistence, transparent object distribution, security, error detection and recovery, transaction management, etc.

Use case design

- Identify participating design classes: derived from analysis classes in analysis-level use-case realization, derived from special requirements on a use-case realization, or discovered during design
- Describe design object interactions as one or more diagrams of sequences of messages among actor instances and design objects
 - Use case invoked by a message from an actor to a design object
 - Each participating design class should have an object in at least one diagram
 - Messages should be (or become) operations on the receiving object's class
 - Use labels and textual flow-of-events to complement the diagrams
 - Sequence diagram should handle all use case relationships (generalizes, extends, etc.)

Use case design (continued)

- Identify all alternative paths, especially new ones
 - Time-outs when nodes or connections fail
 - Erroneous input
 - Error messages and exceptions
- Identify participating subsystems and interfaces and show interactions at sub-system level
 - more coarse granularity than classes/objects
 - Identify bottom-up and/or top-down
 - Fine-grained \leftrightarrow Coarse-grained
- Capture implementation requirements

Class design (1 of 3)

- Class operations, attributes, relationships, methods, states, dependencies, implementation requirements, interface realizations
- Outline design classes
 - boundary classes mapped to user interface technology (e.g. forms, controls, charts, ...)
 - screen design tools implicitly define boundary classes
 - entity classes mapped to persistent technology
 - object-relational mapping, database design, data modeling, etc.
 - control classes must consider distribution (separate design classes on different nodes), performance, and transaction issues.

Class design (2 of 3)

- Identify operations and visibility in language syntax
 - responsibilities, inputs and outputs
 - special operations
 - interfaces provided
 - flow of events descriptions
 - activity diagrams
 - state diagrams
 - Support and coordinate all roles the class plays in different use-case realizations
- Identify attributes
 - type based on programming language or design class
 - cannot share a single attribute among design classes - make it a separate class
 - refactor complex classes - separate concerns

Class design (3 of 3)

- Identify associations and aggregations
 - see interaction diagrams
 - multiplicities, role names, association classes, ordered roles, qualified roles, n-ary associations, navigability
- Identify generalizations
- Describe methods, as necessary (defer to implementation)
- Describe states and state chart
- Handle special requirements
- Refactor mercilessly during iterations with Implementation Workflow

Subsystem Design

- Ensure that the subsystem...
 - is as independent as possible of other subsystems and/or interfaces
 - provides the right interfaces
 - fulfills its purpose: correctly realizes operations of the interfaces it provides
- Depend on interfaces, where possible, rather than the subsystem internals
- Relocate classes to other subsystems to reduce dependencies
- Define collaborations to show how a subsystem realizes any of its interfaces

Summary

- Design model
 - Added technology details and structure that provide a blueprint of the implementation while preserving the structure imposed by the analysis model
 - Design subsystems and their dependencies (including layering), interfaces, contents
 - Design classes, including operations, attributes, relationships and implementation requirements
 - Process model for active classes
 - Use-case realizations - design
 - Architectural views
- Deployment model
 - Nodes, connections, mapping of functions