Chapter 11
Object Design

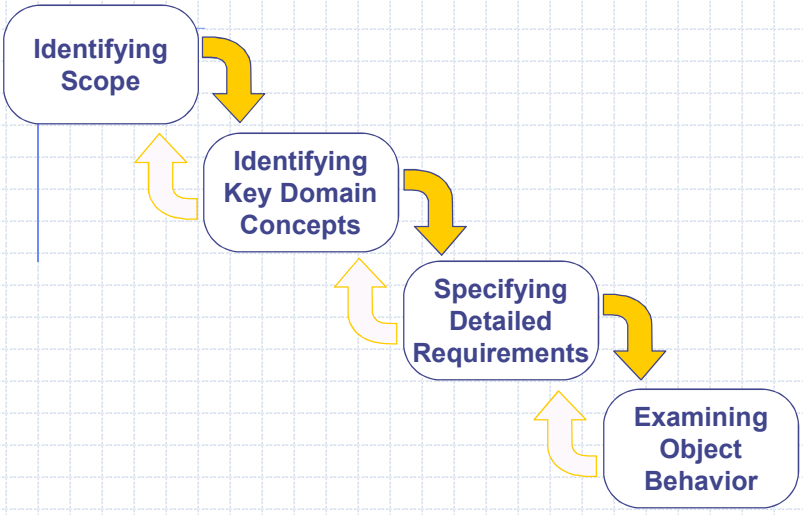
What You Will Learn

- Review Analysis
- Highlight Design topics
- Reinforce the Differences between Analysis and Design

Analysis is . . .

- The 'what'
- Defining the problem space
- A conceptual model to facilitate understanding
- Unaffected by software considerations
- Unaffected by performance considerations

Analysis Summary



Identifying Scope

- Create Use Cases
  - Identify:
    - Actors
    - System behavior (high-level)
    - System boundary
  - Purpose:
    - Define scope.
    - Highlight key domain concepts.
    - Provide requirements framework.

Identifying Key Domain Concepts

- Create a class diagram.
  - Begin data dictionary.
  - Identify:
    - Classes
    - Class definitions
  - Purpose:
    - Provide framework for detailed requirements.
    - Create repository of all system information.

Specifying Detailed Requirements

- Create scenarios and sequence diagrams.
  - Identify:
    - Logical system interaction
    - Attributes & Operations
    - Associations
  - Purpose:
    - Define system behavior.
  - Update:
    - Use cases
    - Class Diagram
    - Data dictionary

Examining Object Behavior

- Create state diagrams.
  - Identify:
    - Operation definition
    - Attributes
  - Purpose:
    - Ensure cohesion of classes.
    - Check sequence diagrams for consistency.
  - Update:
    - Use Cases
    - Class diagram
    - Sequence diagrams
    - Data Dictionary

Completeness?

- Is the increment complete?
- Have all Use Cases for this increment been explored?
  - Class diagram includes all key domain concepts.
  - Illuminating scenarios and sequence diagrams covered.
  - Dynamic objects' states understood.
- What if it is not complete?
  - Change plans or schedules for the iteration accordingly.
  - Or iterate through this increment, completing unfinished areas.

Consistency?

- Class diagram includes all domain objects identified in use cases, scenarios and sequence diagrams.
- Class diagram includes all attributes, operations and associations discovered from sequence diagrams and state diagrams.
- Use cases are updated to reflect changes made while creating scenarios and sequence diagrams.
- Data dictionary includes all information uncovered during this iteration.

Test Analysis Model

- Step through sequence diagrams
  - Check that for each event:
    - Class diagram includes operation in the receiving class.
    - Class diagram identifies associations between sender and receiver classes.
    - Class diagram has sufficient knowledge for operations.
      - By attributes, parameters, or associations
- Step through state diagrams
  - Check that for each transition:
    - Class diagram includes the operation.
    - Guard condition can be tested.
  - Check that for each state:
    - Class diagram identifies how a class will know its state.
    - Class diagram includes operations action-expressions.

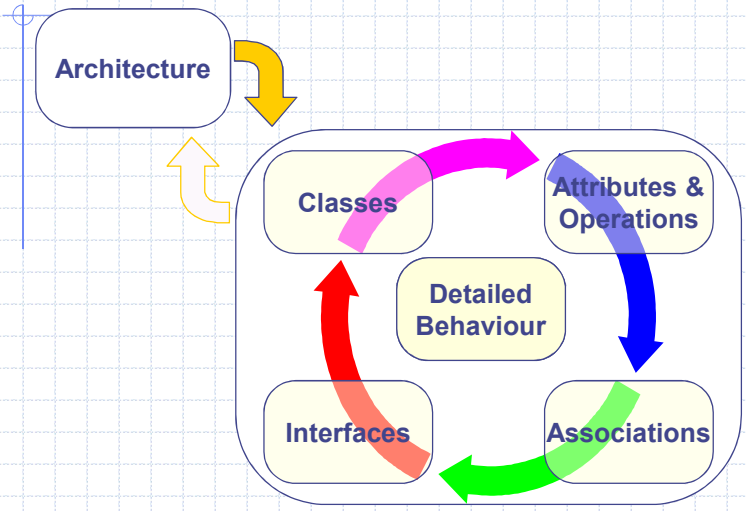
What's Missing?

- Brainstorm related operations.
  - Deposit cash.
    - Transfer between accounts.
    - Payments to credit line.
    - Cancel a deposit.
- CRUD
  - Create, Read, Update, Delete operations.
  - Read and update attributes.

Design

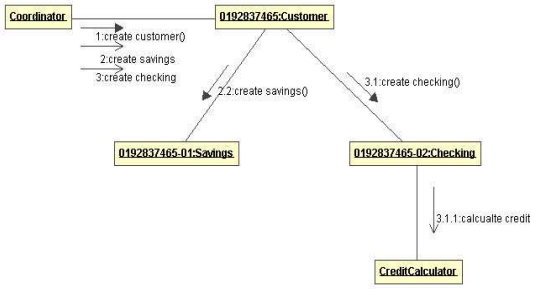
- Turn focus to software and construction, instead of understanding system requirements.
  - Minimize future maintenance effort.
  - Apply design constraints such as space, time and complexity.
  - Capture omitted detail.
  - Apply design principles, encapsulation, reuse and inheritance.
  - Architecture design.

Design Map



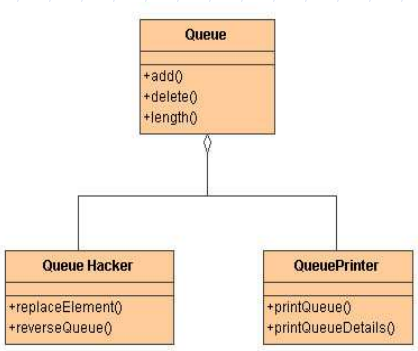
Detailed Behavior

- Collaboration diagrams are used to provide perspective of system behavior per object.
- Accommodate great detail.
- Alternative designs are investigated.



Classes

- Complete
- Sufficient
- Primitive
- High cohesion
- Low coupling

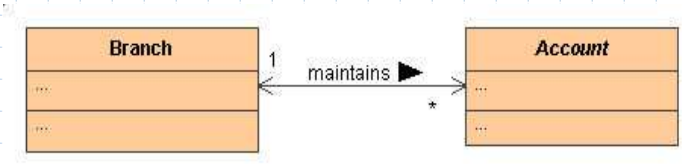


Attributes and Operations

- Attribute or association or operation
- Class vs. object attributes
- Derived attributes
- Visibility
- Operation Details
  - Arguments
  - Argument type
  - Return type

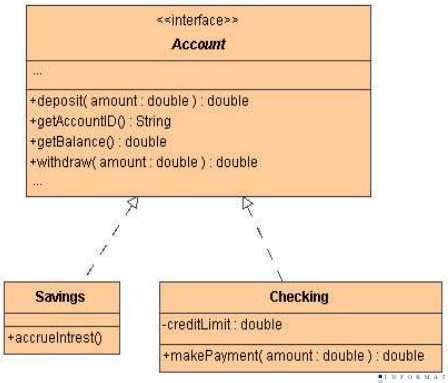
Associations

- Navigation
  - Single direction
  - Bi-directional
- Implementation
  - number of references
  - Location(s) of references
- Collections
  - Capturing constraints
  - Choosing classes
- Aggregation
  - By-reference
  - By-value



Interfaces

- Maintain separation between interface and implementation.
- Optimize design with use of interfaces.



Architecture

- Logical software architecture
- Physical software architecture
- Hardware architecture
- Software deployment