Object-Oriented Software Development: Requirements elicitation and analysis

CS 4354
Fall 2012

Jill Seaman

Progress Report

- So far we have learned about the tools used in object-oriented design and implementation
  - ✦UML Models
  - ✦Java programming language
- Next we will learn how to use them in the Object-oriented software development process.
  - ✦How to analyze a problem, design a solution using models, and implement it as a Java program.

Object-oriented analysis, design, implementation

- **Object-oriented analysis**: finding and describing the objects (or concepts) in the problem domain.
- **Object-oriented design**: defining software objects and how they collaborate to fulfill the requirements.
- **Object-oriented implementation**: implementing the designs in an object-oriented language such as Java or C++.

Object-oriented software development

- During **requirements elicitation**, the client and developers define the purpose of the system. (Develop use cases)
- During **analysis**, developers aim to produce a model of the system that is correct, complete, consistent, and unambiguous.
- During **system design**, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams.
- During **object design**, developers define solution domain objects to bridge the gap between the analysis model and the hardware/ software platform defined during system design.
- During **implementation**, developers translate the solution domain model into source code.
- During **testing**, developers find differences between the system and its models by executing the system with sample input data.

Ch 4: Requirements Elicitation

- During requirements elicitation, the client and developers define the purpose of the system.
- The result of this phase is a Requirements Specification.
  - ✦Written in natural language
- The Requirements Specification contains
  - ✦Nonfunctional Requirements
  - ✦Functional model
    - – In object oriented development, this tends to be represented by use cases and scenarios

Requirements

- Functional Requirements:
  - ✦What the system must do: features, functionality it must have.
  - ✦describe the interaction between the system and its environment independent of its implementation.
- Nonfunctional Requirements:
  - ✦How the system must function: constraints over the functions offered by the system.
  - ✦describe aspects of the system that are not directly related to the functional behavior of the system.
    - – Usability, Reliability, Performance, Supportability

Requirements: Quality

- The Requirements Specification must have the following qualities:
  - ✦**Completeness**: all scenarios included
  - ✦**Consistency**: does not contradict itself
  - ✦**Clarity**: free from ambiguity
  - ✦**Correctness**: exactly what the customer needs
  - ✦**Realistic**: the system as described is feasible
  - ✦**Verifiable**: tests can be written to validate the requirements
  - ✦**Traceable**: requirements can be traced to system functions and vice versa

Requirements Elicitation Activities

- Identifying actors.
- Identifying scenarios.
- Identifying use cases.
- Refining use cases.
- Identifying relationships among use cases.
- Identifying nonfunctional requirements.

Identifying actors

- Identifying actors:
  - ✦all external entities that interact with the system
  - ✦humans (roles) or systems (software, databases)
  - ✦defines system boundaries
  - ✦defines perspectives from which analysts need to consider the system

Questions for identifying actors:
- Which user groups are supported by the system to perform their work?
- Which user groups execute the system's main functions?
- Which user groups perform secondary functions (maintenance/admin)?
- With what external hardware of software system will the system interact?

Identifying scenarios

- Identifying scenarios:
  - ✦a narrative description of what people do and experience as they try to make use of the system
  - ✦a specific instance of concrete events
  - ✦understandable to users and customers

Questions for identifying scenarios:
- What are the tasks that the actor wants the system to perform?
- What information does the actor access? Who creates that data? Can it be modified or removed? by whom?
- Which external changes does the actor need to inform the system about?
- Which events does the system need to inform the actor about?

Identifying use cases

- Identifying use cases:
  - ✦specifies all possible scenarios for a given piece of functionality
  - ✦generalizes scenarios, describes a flow of events
  - ✦attach to the initiating actor

Guidelines for writing use cases:
- Name with a verb phrase (ReportEmergency).
- Steps in the flow of events should be phrased in the active voice, so it is clear who does what.
- The boundary should be clear, what the system does, what actors do.
- Causal relationship between successive steps should be clear.

Refining use cases, Identifying relationships among use cases, actors

- Refining use cases:
  - ✦Rewriting, adding missing cases, dropping unneeded ones
  - ✦Add more details, constraints
  - ✦Describe exceptional cases
- Identifying relationships:
  - ✦start drawing use case diagrams with actors/ellipses for use cases
  - ✦use different kinds of relationships: communication, extend, include
  - ✦For communication relationship, indicate if that actor initiates or participates in the interaction.

Chapter 5: Analysis
Products of Requirements Elicitation and Analysis

Products of Requirements Elicitation

- Requirements specification: Understood by users/customer
  - ✦nonfunctional requirements
  - ✦functional model
    - represented by use cases and scenarios

Products of Analysis

- **Analysis model**: Understood by developers
  - ✦functional model (use cases developed in requirements elicitation)
  - ✦**analysis object model** (class diagram, domain concepts)
  - ✦**dynamic model** (state machine and sequence diagrams)

Entity, Boundary, and Control Objects

- **Entity objects** represent the persistent information tracked by the system.
  - ✦Year, Month, and Day
- **Boundary objects** represent the interface between the actors and the system.
  - ✦Button, LCDDisplay
- **Control objects** are in charge of realizing use cases.
  - ✦ChangeDateControl represents activity of changing the date by pressing combinations of buttons

Separates interface objects from objects that are less likely to change

Analysis Activities: From Use Cases to Objects

- The activities that transform the use cases and scenarios produced during requirements elicitation into an analysis model.
  - ✦Identifying Entity Objects, Boundary Objects, Control Objects
  - ✦Mapping Use Cases to Objects with Sequence Diagrams
  - ✦Identifying Associations, Aggregations, Attributes
  - ✦Modeling Inheritance Relationships
  - ✦Modeling State-Dependent Behavior of Individual Objects
  - ✦Reviewing the Analysis Model

Identifying entity objects

- Identifying entity objects
  - ✦find the actors that participate in the use case
  - ✦as objects are found, record their names, attributes, and responsibilities
  - ✦use names used by the user/customer/domain specialists

Heuristics for identifying entity objects
- Terms that developers of users need to clarify in order to understand the use case.
- Recurring nouns in the use case.
- Real-world entities that the system needs to track.
- Real-world activities that the system needs to track.
- Data sources or sinks (e.g., Printer, Database)

The ReportEmergency use case

Use Case Name	ReportEmergency
Entry condition	1. The FieldOfficer activates the Report Emergency function of her terminal.
Flow of events	2. FRIEND responds by presenting a form to the officer. The form includes an emergency type menu (general emergency, fire, transportation), a location, incident description, resource request, and hazardous material fields. 3. The FieldOfficer completes the form by specifying minimally the emergency type and description fields. The FieldOfficer may also describe possible responses to the emergency situation and request specific resources. Once the form is completed, the FieldOfficer submits the form by pressing the Send Report button, at which point, the Dispatcher is notified. 4. The Dispatcher reviews the information submitted by the FieldOfficer and creates an Incident in the database by invoking the OpenIncident use case. All the information contained in the FieldOfficer's form is automatically included in the incident. The Dispatcher selects a response by allocating resources to the incident (with the AllocateResources use case) and acknowledges the emergency report by sending a FRIENDgram to the FieldOfficer.
Exit condition	5. The FieldOfficer receives the acknowledgment and the selected response.

Entity objects for the ReportEmergency use case

Dispatcher	Police officer who manages Incidents. A Dispatcher opens, documents, and closes Incidents in response to Emergency Reports and other communication with FieldOfficers. Dispatchers are identified by badge numbers.
EmergencyReport	Initial report about an Incident from a FieldOfficer to a Dispatcher. An EmergencyReport usually triggers the creation of an Incident by the Dispatcher. An EmergencyReport is composed of an emergency level, a type (fire, road accident, other), a location, and a description.
FieldOfficer	Police or file officer on duty. A FiledOfficer can be allocated to, at most, one Incident at a time. FieldOfficers are identified by badge numbers.
Incident	Situation requiring attention from a FieldOfficer. An Incident may be reported in the system by a FieldOfficer or anybody else external to the system. An Incident is composed of a description, a response, a status (open, closed, documented), a location, and a number of FieldOfficers.

Identifying boundary objects

- Identifying boundary objects
  - ✦in each use case, each actor interacts with at least one boundary object
  - ✦boundary object collects info from actor, displays info to actor
  - ✦translates information between entity and control objects

Heuristics for identifying boundary objects
- Basic user interface controls needed to initiate the use case. (Button)
- Forms the users need to enter data into the system (EmergencyReportForm).
- Notices and messages the system uses to respond to the user
- When multiple actors are involved in a use case, identify actor terminals (DispatcherStation) to refer to the user interface under consideration.
- Do not model the visual aspects of the interface with boundary objects

Entity objects for the ReportEmergency use case

AcknowledgmentNotice	Notice used for displaying the Dispatcher's acknowledgment to the FieldOfficer.
DispatcherStation	Computer used by the Dispatcher.
ReportEmergencyButton	Button used by a FieldOfficer to initiate the ReportEmergency use case.
EmergencyReportForm	Form used for the input of the ReportEmergency. This form is presented to the FieldOfficer on the FieldOfficerStation when the Report Emergency function is selected. The EmergencyReportForm contains fields for specifying all attributes of an emergency report and a button (or other control) for submitting the completed form.
FieldOfficerStation	Mobile computer used by the FieldOfficer.
IncidentForm	Form used for the creation of Incidents. This form is presented to the Dispatcher on the DispatcherStation when the EmergencyReport is received. The Dispatcher also uses this form to allocate resources and to acknowledge the FieldOfficer's report.

Identifying control objects

- Identifying control objects
  - ✦coordinate boundary and entity objects
  - ✦do not have concrete counterpart in the real world
  - ✦collects information from boundary objects and dispatches to entity objects

Heuristics for identifying control objects
- Identify one control object per use case.
- Identify one control object per actor in the use case.
- The life span of a control object should cover the extent of the use case or the extent of a user session.

Control objects for the ReportEmergency use case

ReportEmergencyControl	Manages the ReportEmergency reporting function on the FieldOfficerStation. This object is created when the FieldOfficer selects the â€œReport Emergencyâ€ button. It then creates an EmergencyReportForm and presents it to the FieldOfficer. After submitting the form, this object then collects the information from the form, creates an EmergencyReport, and forwards it to the Dispatcher. The control object then waits for an acknowledgement to come back from the DispatcherStation. When the acknowledgment is received, the ReportEmergencyControl object creates an AcknowledgmentNotice and displays it to the FieldOfficer.
ManageEmergencyControl	Manages the ReportEmergency reporting function on the DispatcherStation. This object is created when an EmergencyReport is received. It then creates an IncidentForm and displays it to the Dispatcher. Once the Dispatcher has created an Incident, allocated Resources, and submitted an acknowledgment, ManageEmergencyControl forwards the acknowledgment to the FieldOfficerStation.

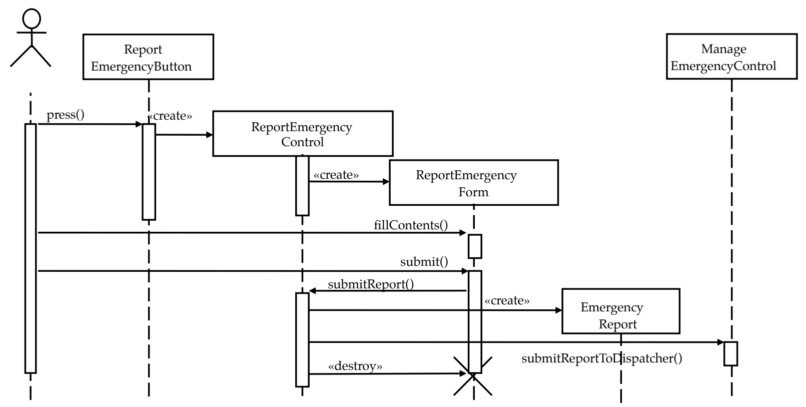
Mapping use cases to objects with sequence diagrams

- Sequence diagrams
  - ✦show how behavior of a use case is distributed among participating objects
  - ✦allow developers to find missing objects and clarify behavior
  - ✦assigns responsibilities to each object as a set of operations (identifies the operations)

Heuristics for drawing sequence diagrams
- The first column should correspond to the actor who initiated the use case.
- The second column should be a boundary object (that the actor used to initiate the use case).
- The third column should be the control object that manages the rest of the use case.
- Control objects are created by boundary objects initiating use cases.
- Secondary boundary objects are created by control objects.
- Entity objects are accessed by control and boundary objects.

Sequence diagram for ReportEmergency use case

Part 1



Identifying attributes

- Attributes:
  - ✦properties of individual objects
  - ✦note names and data types of each
  - ✦properties represented by objects are NOT attributes (ie address)

Heuristics for identifying attributes
- Examine possessive phrases (______ of <an object>)
- Represent stored state as an attribute of the entity object.
- Describe each attribute.
- Do not waste time describing fine details before the object structure is stable.

Identifying associations

- Associations:
  - ✦show relationship between two or more classes
  - ✦name, multiplicity, roles
  - ✦assigns responsibilities to each object as a set of operations

Heuristics for identifying associations
- Examine verb phrases.
- Name associations and roles precisely.
- Use qualifiers as often as possible to identify namespaces and key attributes.
- Eliminate any association that can be derived from other associations.
- Do not worry about multiplicity until the set of associations is stable.
- Too many associations make a model unreadable.

Identifying aggregates, Identifying Inheritance

- Aggregations:
  - ✦denote whole-part relationships
  - ✦composition, special case of aggregation, when the existence of the parts depend on the existence of the whole.
- Inheritance:
  - ✦Generalization is used to eliminate redundancy from the analysis model. (put shared attributes and behavior in superclass).

Modeling State-Dependent Behavior of Individual Objects

- State machine diagrams:
  - ✦represent behavior of the system from the perspective of a single object.
  - ✦helps identify missing use cases, new behavior
  - ✦not necessary to build for each object in model (often for control objects).

Reviewing the Analysis model

- Analysis model is built incrementally and iteratively.
- Reviewed by developers, then jointly with the customer.
- Certain questions should be asked to ensure the model is correct, complete, consistent, realistic.
  - ✦Are all entity objects understandable to the user?
  - ✦For each object: Is it needed by some use case? In which use case is it created? modified? destroyed?
  - ✦Are there multiple classes with the same name?
  - ✦Are there any novel features in the system, that the developers have never experienced before?