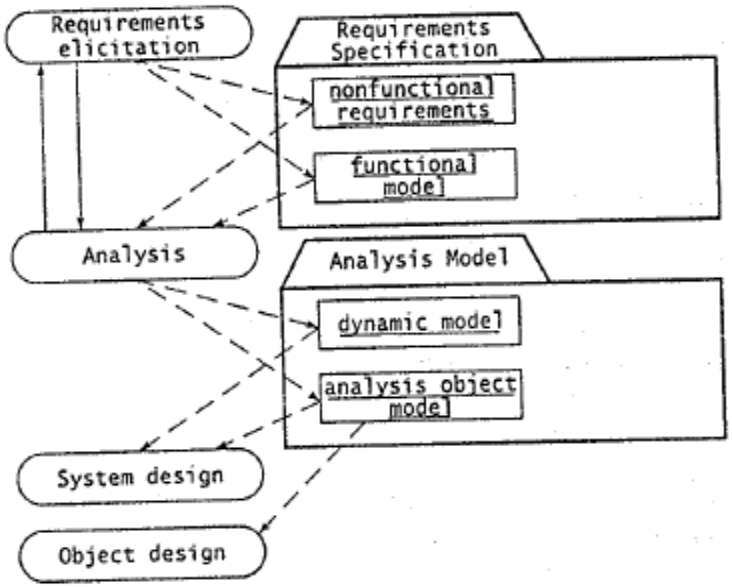
SOFTWARE ENGINEERING
OBJECT-ORIENTED SE USING UML, PATTERNS AND JAVA

Dan CHIOREAN

Lecture 5 – The Analysis model – based on Bernd Bruegge's book

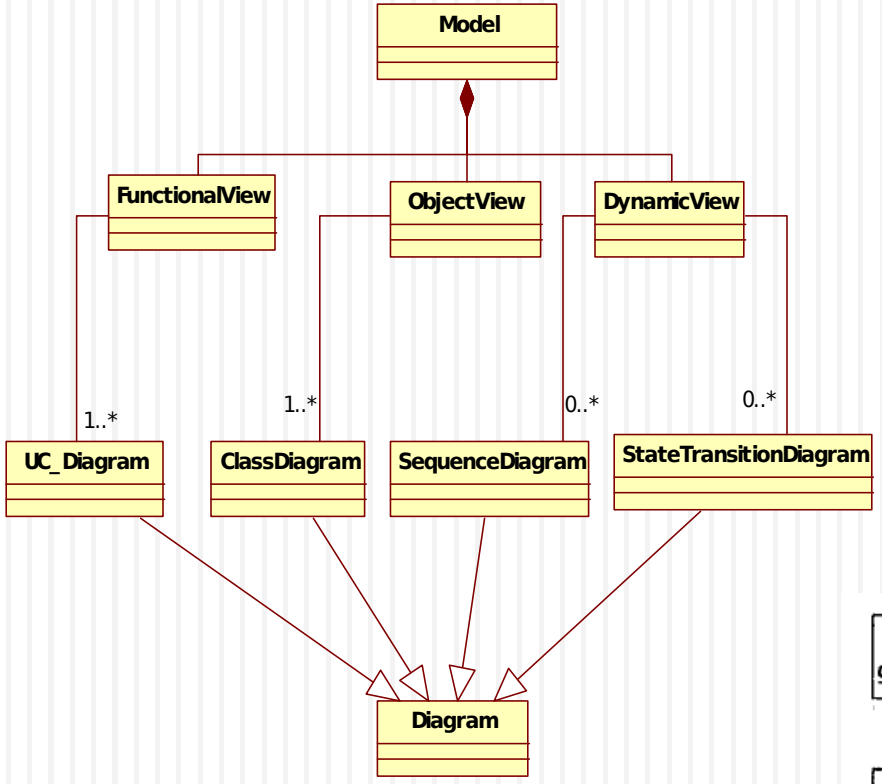
An Overview of Analysis

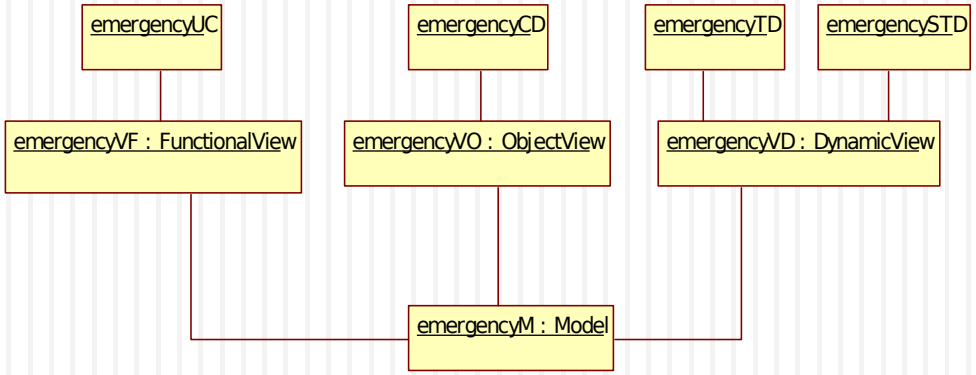
- Analysis focuses on producing a model of the system, called **the analysis model, which is correct, complete, consistent and verifiable**.
- Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users.
- The formalization leads to new insights and to discovery of errors in requirements.

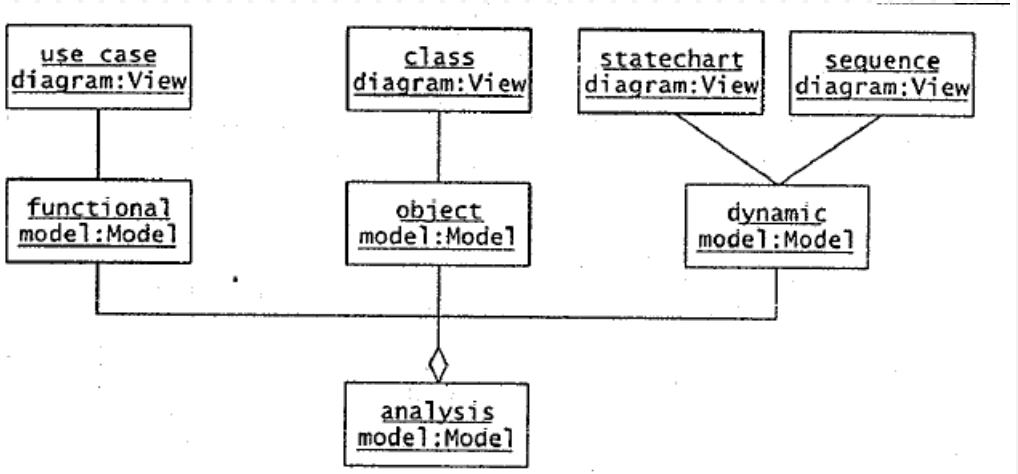


Products of requirements elicitation and analysis

The Analysis Model





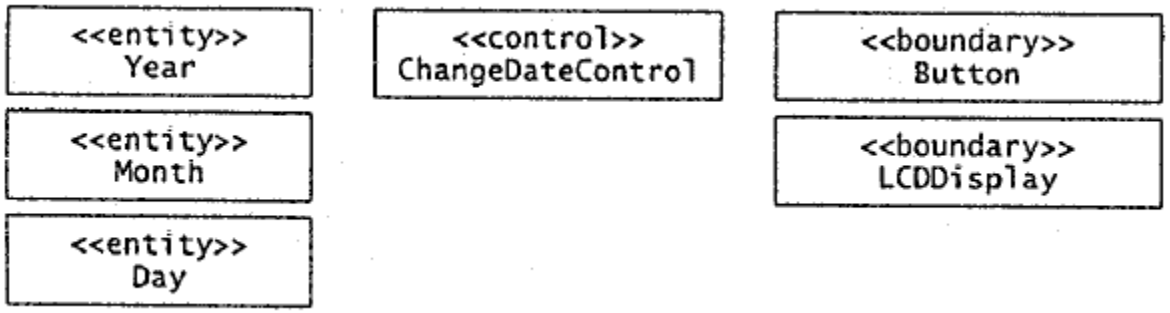


Analysis Object and Dynamic Views

- The Analysis Object View consists of **entity**, **boundary** and **control objects**
  - **Entity objects** represent the persistent information tracked by the system
  - **Boundary objects** represent the interaction between the actors and the system
  - **Control objects** are in charge of realizing use cases
- *Example*: In the 2Bwatch example, Year, Month and Day are entity objects, Button and LCDDisplay are boundary objects, ChangeDataControl is a control object that represents the activity of changing the date by pressing combination of buttons

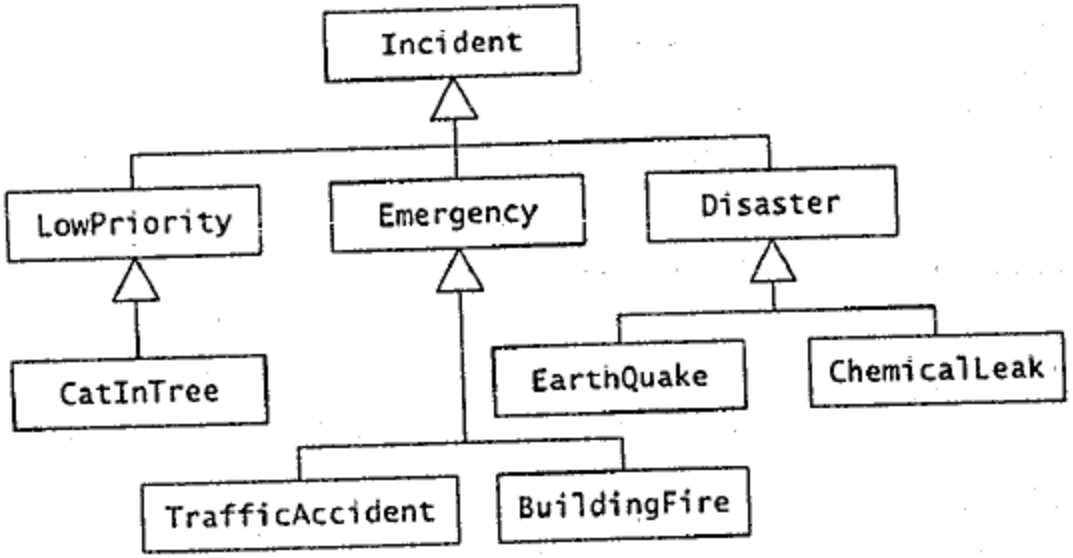
Analysis Object and Dynamic Views/2

- To distinguish between different types of objects, UML provides the stereotype mechanism to enable the developer to attach such meta-information to modeling elements.



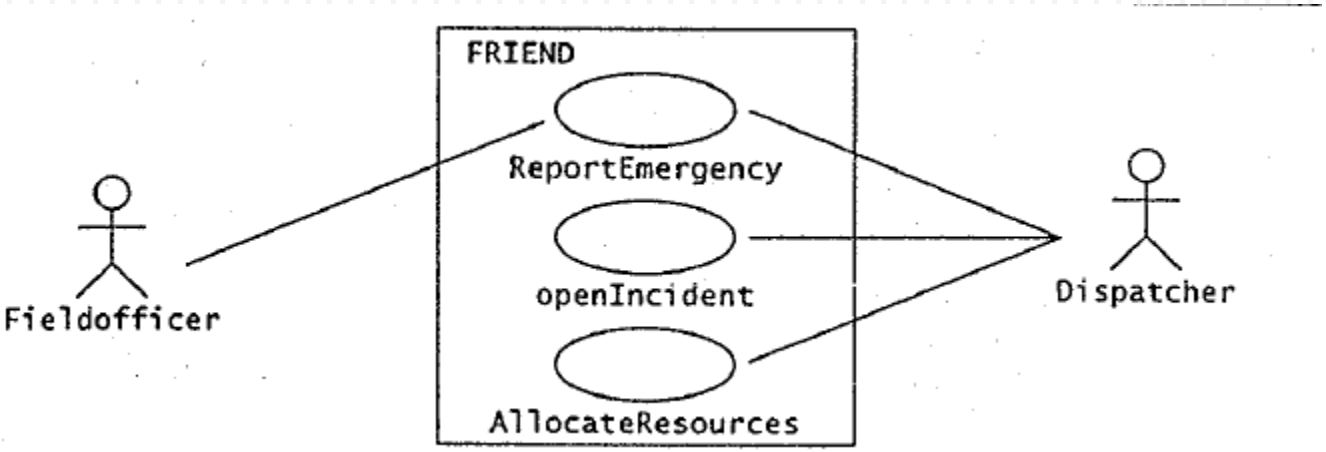
Analysis Object and Dynamic Views – Generalization and Specialization

- In both cases, generalization and specialization result in the specification of **inheritance relationships** between concepts. In some instances, modelers call inheritance relationships **generalization-specialization relationships**. In Bruegge's book, **inheritance** is used to **denote the relationship** and terms "**generalization**" and "**specialization**" to **denote activities** that find inheritance relationships.



The FRIEND system (book pp. 44-59)

- FRIEND – an accident management system in which field officers such as police officer or fire fighter have access to a wireless computer that enable the to interact with a dispatcher. The dispatcher in turn can visualize the current state of all its resources such as police cars or trucks, on a computer screen and dispatch a resource by issuing commands from a workstation.



The FRIED system - cont

Use case name	ReportEmergency
Participating actors	Initiated by FieldOfficer Communicates with Dispatcher
Flow of events	1. The FieldOfficer activates the "Report Emergency" function of her terminal. 2. FRIEND responds by presenting a form to the FieldOfficer. 3. The FieldOfficer fills out the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form. 4. FRIEND receives the form and notifies the Dispatcher. 5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the report. 6. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.
Entry condition	• The FieldOfficer is logged into FRIEND.
Exit condition	• The FieldOfficer has received an acknowledgment and the selected response from the Dispatcher, OR • The FieldOfficer has received an explanation indicating why the transaction could not be processed.
Quality requirements	• The FieldOfficer's report is acknowledged within 30 seconds. • The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

Figure 2-14 An example of a use case, ReportEmergency.

Analysis Activities: **From Use Cases to Objects**

- Identifying Entity Objects,
- Identifying Boundary Objects,
- Identifying Control Objects,
- Mapping Use Cases to Objects with Sequence Diagrams
- Mapping Interactions among Objects with CRC Cards
- Identifying Associations
- Identifying Aggregates
- Identifying Attributes
- Modeling State-Dependent Behavior of Individual Objects
- Modeling Inheritance Relationships
- Reviewing the Analysis Model

Analysis Activities: **Identifying Entity Objects**

Heuristics for identifying entity objects

- Terms that developers or users need to clarify in order to understand the use case
- Recurring nouns in the use case (e.g. Incident)
- Real-world entities that the system needs to track (e.g. FieldOfficer, Dispatcher, Resource)
- Real-world activities that the system needs to track (e.g. EmergencyOperationsPlan)
- Data sources or sinks (e.g. Printer)

Analysis Activities: **Identifying Entity Objects/2**

Table 5-2 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 fire officer on duty. A FieldOfficer 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.

Analysis Activities: **Identifying Boundary Objects**

Heuristics for identifying boundaryobjects

- Identify user interface controls that the user needs to initiate the use case (e.g., ReportEmergencyButton)
- Identify forms the users need to enter data into the system (e.g., EmergencyReportForm)
- Identify notices and massages the system uses to respond to the user (e.g. AcknoledgmentNotice)
- When multiple actors are involved in a use case, identify actor terminals (e.g., DispatcherStation) to refer to the user interface under consideration
- Do not model the visual aspects of the interface with boundary objects (user mock-ups are better suited for that)
- Always use the end user's terms for describing interfaces; do not use terms from the solution or implementation domains.

Analysis Activities: **Identifying Boundary Objects/2**

Table 5-3 Boundary 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.

Analysis Activities: Identifying ControlObjects
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. If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case probably does not have well-defined entry and exit conditions

Analysis Activities: Identifying Control Objects/2

Table 5-4 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 "Report Emergency" 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 acknowledgment 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

- A **sequence diagram** ties use cases with objects. It shows how the behavior of a use case (or scenario) is distributed among participating objects.

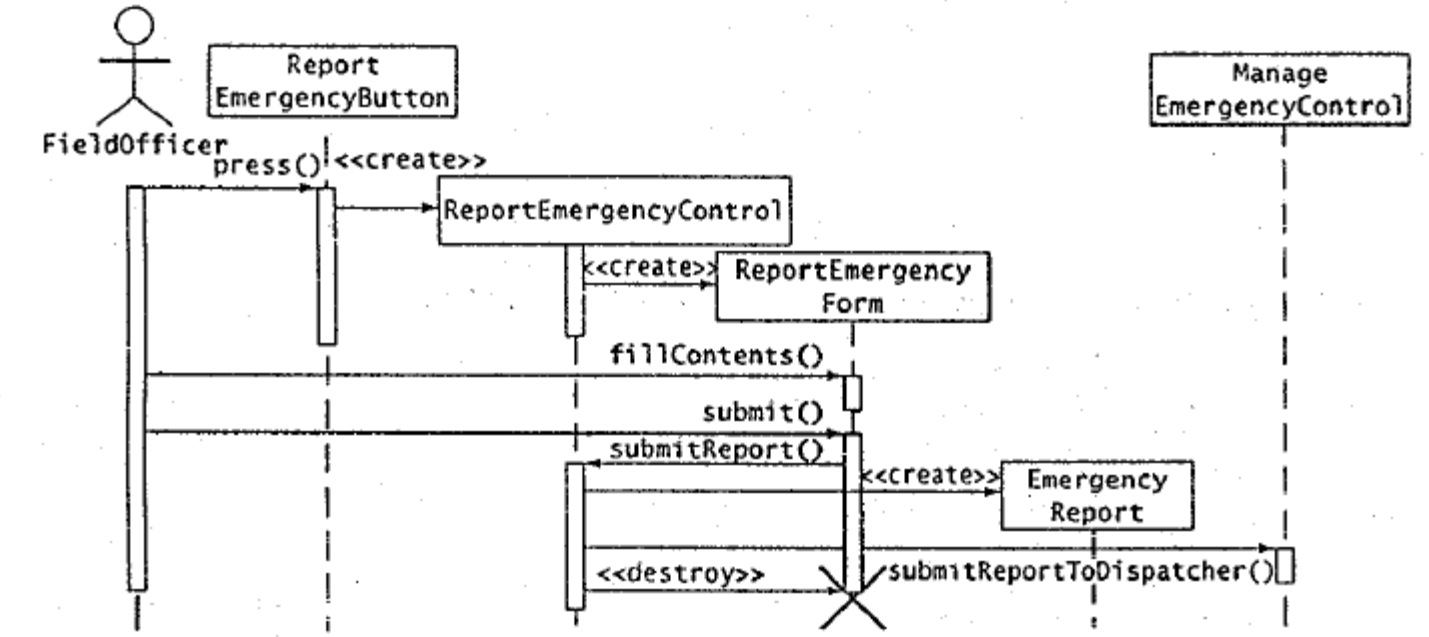


Figure 5-8 Sequence diagram for the ReportEmergency use case.

Mapping Use Cases to Objects with Sequence Diagrams

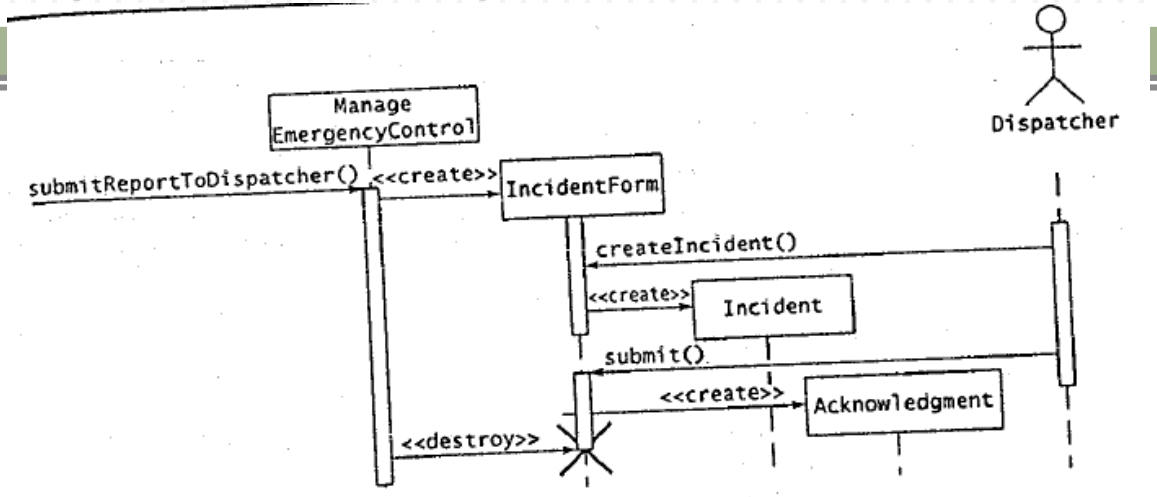


Figure 5-9 Sequence diagram for the ReportEmergency use case (continued from Figure 5-8).

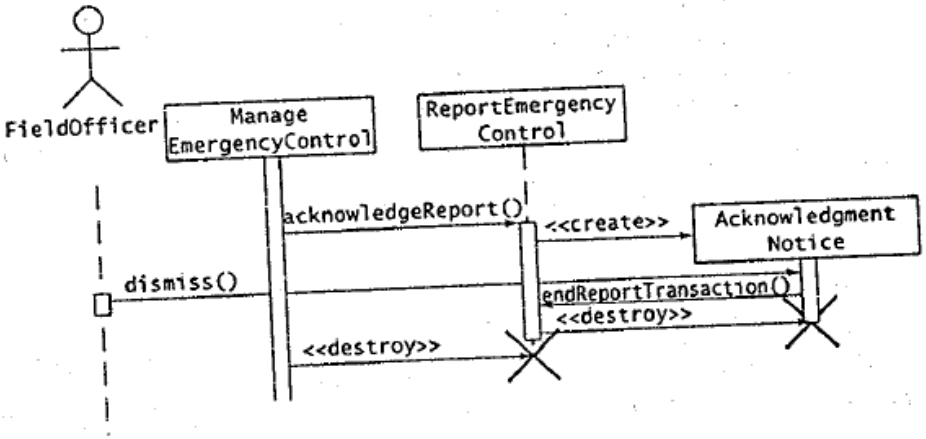


Figure 5-10 Sequence diagram for the ReportEmergency use case (continued from Figure 5-9).

Mapping Use Cases to Objects with Sequence Diagrams

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 columns should be the control object that manages the rest of use case.
- Control objects are created by boundary objects initiating use cases.
- Boundary objects are created by control objects.
- **Entity objects are accessed by** control and boundary objects.
- **Entity objects never access** boundary or control objects; this makes it easier to share entity objects across use cases.

Modeling interactions among Objects with CRC Cards

- An alternative for identifying interactions among objects are CRC cards [Beck & Cunningham, 1989]. CRC cards (class, responsibilities, collaborators) were initially introduced as a tool for teaching object-orientation concepts to novices and to experienced developers unfamiliar with object-orientation.
- (Objects of!)Each class is represented with an index card. The name of the class is depicted on the top, its responsibilities on the left column, and the name of the classes it needs to accomplish the responsibilities are depicted on the right column.
- CRC cards and sequence diagrams are two different representations for supporting the same type of activity.

Modeling interactions among Objects with CRC Cards/2

- Sequence diagrams are a better tool for a single modeler or for documenting a sequence of interactions because are more precise and compact. CRC cards are a better tool for a group of developers refining and iterating over an object structure during a brainstorming session because are easier to create and to modify.

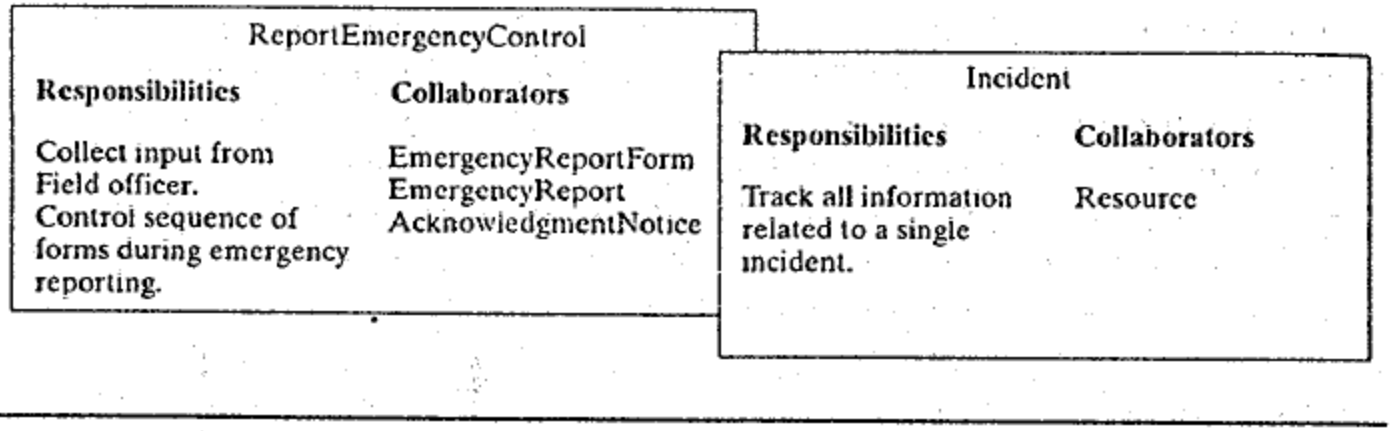


Figure 5-12 Examples of CRC cards for the `ReportEmergencyControl` and the `Incident` classes.

Identifying associations

- An association shows a relationship between two or more different objects instantiated by the same class (recursive association) or by different classes.
- Identifying associations has two advantages:
  - It clarifies the analysis model by making relationships between objects explicit (e.g. , an EmergencyReport can be created by a FieldOfficer or a Dispatcher).
  - It enables the developer to discover boundary cases associated with links. Boundary cases are exceptions that must be clarified in the model.
- Understanding the role of different artifacts used in the process of software development and acquiring the ability of realizing and using these artifacts

Identifying associations/2

- Association have several properties:
  - **Associations names are optional** and need not to be unique globally.
  - A **role** at each end, identifying the function of each class' instances with respect to the instances of the opposite end (e.g., author is the role played by FieldOfficer in relation with EmergencyReport.
  - A **multiplicity** at each end, identifying the possible number of instances.

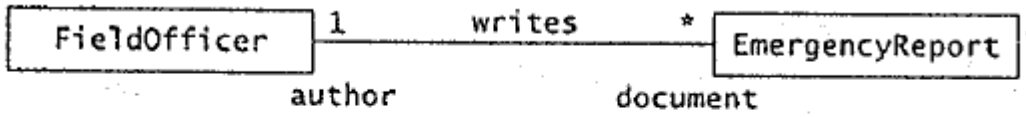


Figure 5-13 An example of association between the EmergencyReport and the FieldOfficer classes.

Identifying associations/3
Heuristics for identifying associations

- Examine verb phrases
- Name associations and roles as suggestive as possible
- 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 associations/4
Identifying Aggregates

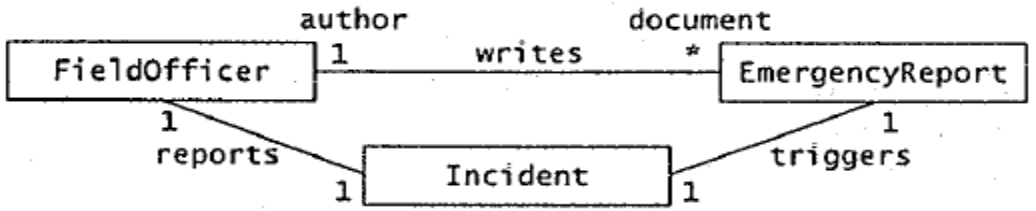


Figure 5-14 Eliminating redundant association. The receipt of an EmergencyReport triggers the creation of an Incident by a Dispatcher. Given that the EmergencyReport has an association with the FieldOfficer that wrote it, it is not necessary to keep an association between FieldOfficer and Incident.

- Aggregations are special types of associations denoting a whole-part relationship.
- There are two types of aggregations: composition and shared.
- A solid diamond denotes composition. A composition indicates that the existence of the parts depends on the whole
- A shared aggregation relationship indicate that the whole and the parts can exist independently

Identifying associations/5
Identifying Aggregates

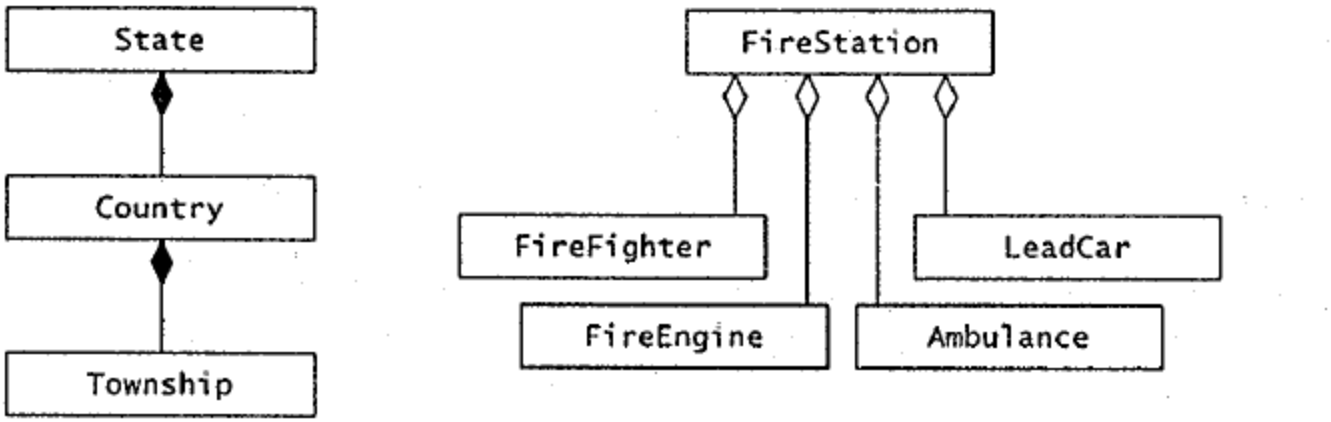


Figure 5-15 Examples of aggregations and compositions (UML class diagram). A State is composed of many Counties, which in turn is composed of many Townships. A FireStation includes FireFighters, FireEngines, Ambulances, and a LeadCar.

Identifying Attributes

- Attributes are properties of individual objects
- When identifying properties of objects, only the attributes relevant to the system should be considered.
- Properties that are represented by objects are not attributes
- Developers should identify as many associations as possible before identifying attributes to avoid confusing attributes and objects.
- Attributes have:
  - A **name** identifying them within an object
  - A brief description
  - A **type** describing the legal values it can take

Identifying Attributes/2
Heuristics for identifying attributes

- Examine possessive phrases.
- Represent stored state as an attribute of the entity object.
- Describe each attribute.
- Do not represent an attribute as an object; use an association instead
- Do not waste time describing fine details before the object structure is stable.

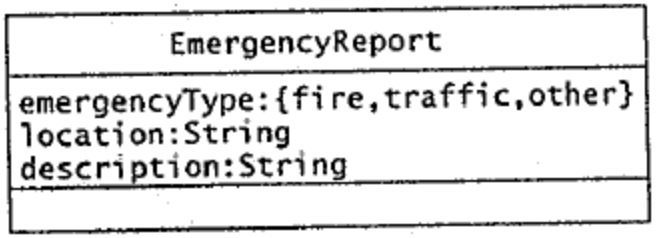


Figure 5-16 Attributes of the EmergencyReport class.

Modeling State-Dependent Behavior of Individual Objects

- Statechart diagrams represent behavior from the perspective of a single object. Viewing behavior from the perspective of each object enable the developer to build a more formal description of the behavior of the object, and consequently, to identify missing use cases. By focusing on individual states, developers may identify new behavior. By examining each transition in the statechart diagram that is triggered by a user action that triggers the transition.
- Only objects with an extended lifespan and state-dependent behavior are worth considering. This is almost always the case for control objects, less often for entity objects, and almost never for boundary objects.

Modeling State-Dependent Behavior of Individual Objects/2

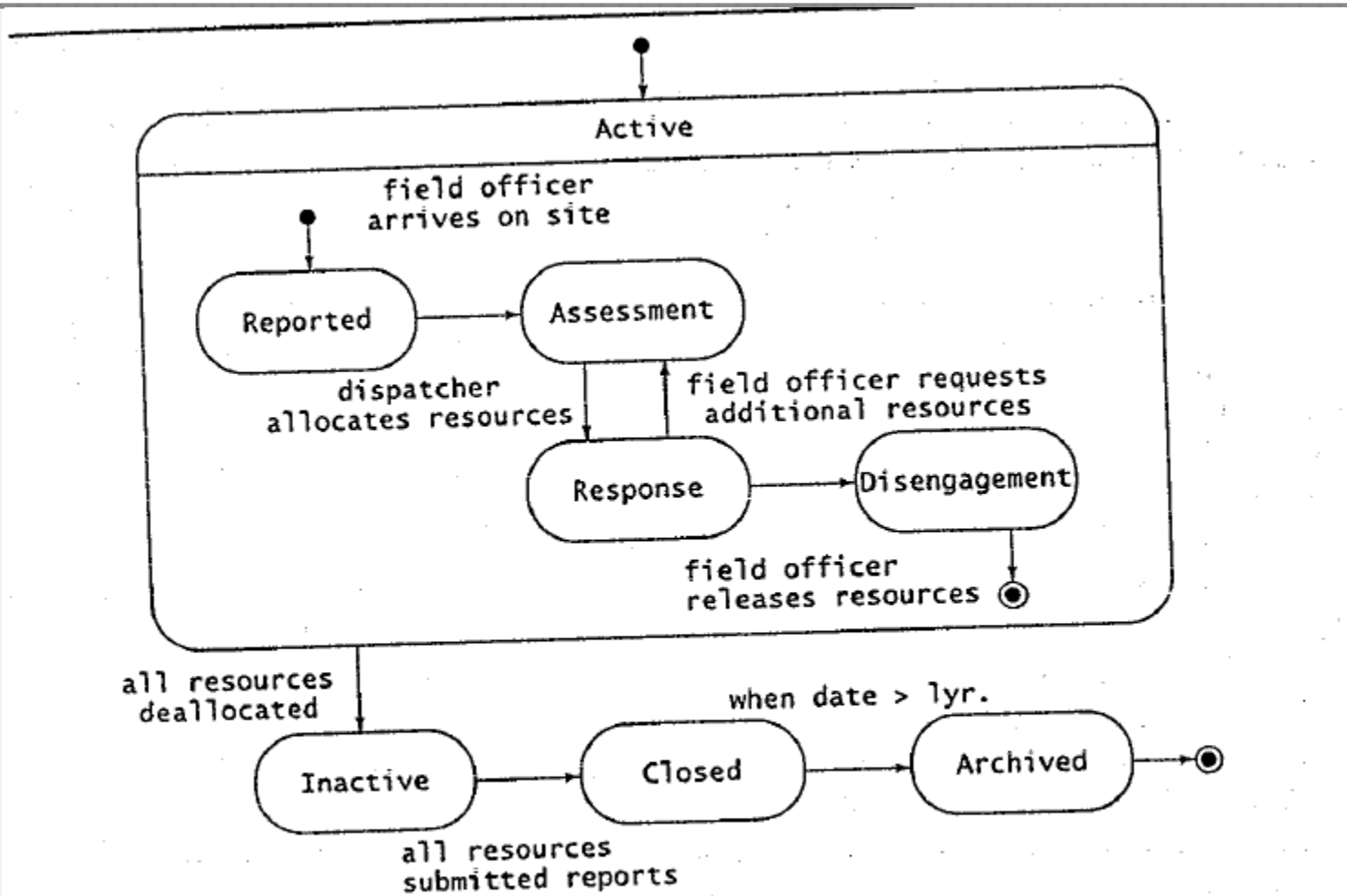


Figure 5-17 UML statechart for Incident.

Modeling Inheritance Relationships between Objects

- Generalization is used to eliminate redundancy from the analysis model. If two or more classes share attributes or behavior, the similarities are consolidated into a superclass. Both Dispatchers and FieldOfficers have a badgeNumber attribute that serves to identify them within a city. To model this similarity, we introduce an abstract *PoliceOfficer* class.

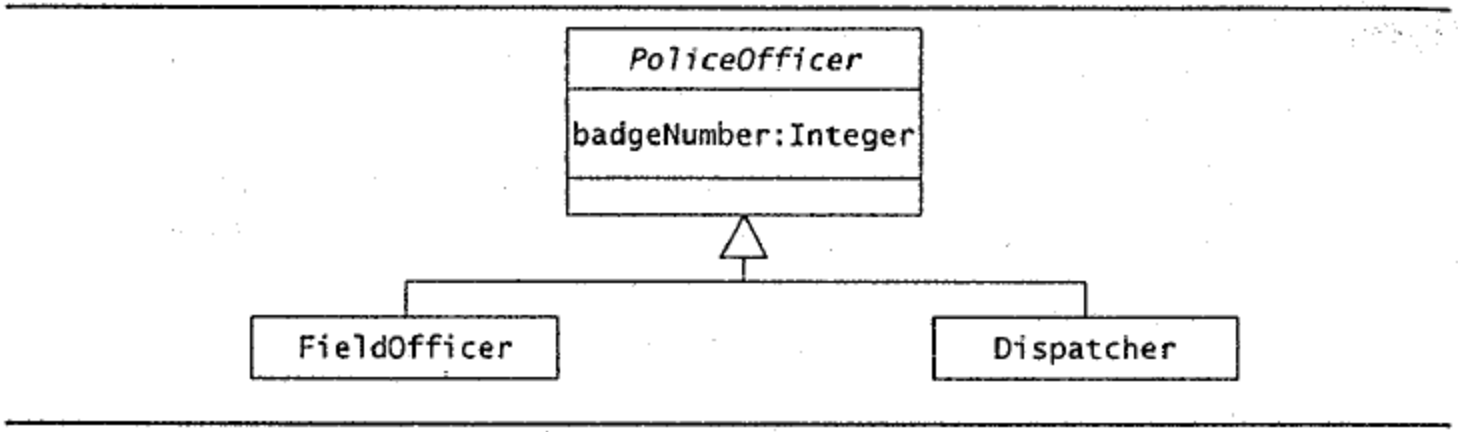


Figure 5-18 An example of inheritance relationship (UML class diagram).

Reviewing the Analysis Model

- The analysis model is built incrementally and iteratively. This model is seldom correct or even complete on the first pass. Before the analysis model converges toward a correct specification usable by the developers for design and implementation, several iterations with the client and the user are necessary.
- Once the analysis model becomes stable (i.e., when the number of changes to the models are minimal and the scope of the changes localized), the analysis model is reviewed, first by the developers (i.e., internal reviews), then jointly by the developers and the client.
- The goal of the review is to make sure that the requirements specification is correct, complete, consistent and unambiguous.
- Developers should be prepared to discover errors downstream and make changes to the specification.

Reviewing the Analysis Model/2

- To ensure that **the model is correct**, the following questions should be asked:
  - Is the glossary of entity objects understandable by the user?
  - Do abstract classes correspond to user-level concepts?
  - Are all descriptions in accordance with the users' definitions?
  - Do all entity and boundary objects have meaningful noun phrases as names?
  - Do all use cases and control objects have meaningful verb phrases as names?
  - Are all error cases described and handled?

Reviewing the Analysis Model/3

- To ensure that **the model is complete**, the following questions should be asked:
  - For each object: Is it needed by any use case? In which use case is it created? modified? Destroyed? Can it be accessed from a boundary object?
  - For each attribute: When is it set? What is its type? Should it be a qualifier?
  - For each association: When is it traversed? Why was the specific multiplicity chosen? Can associations with one-to-many multiplicities qualified?
  - For each control object: Does it have the necessary associations to access the objects participating in its corresponding use case?

Reviewing the Analysis Model/4

- To ensure that **the model is consistent**, the following questions should be asked:
  - Are there multiple classes or use cases with the same name?
  - Do entities (e.g., use cases, classes, attributes) with similar names denote similar concepts?
  - Are there objects with similar attributes and associations that are not in the same generalization hierarchy?
- To ensure that the system described by **the analysis model is realistic**, the following questions should be asked:
  - Are there any novel features in the system? Were any studies or prototypes built to ensure their feasibility?
  - Can the performance and reliability requirements be met? Were these requirements verified by any prototypes running on the selected hardware?

Analysis Summary

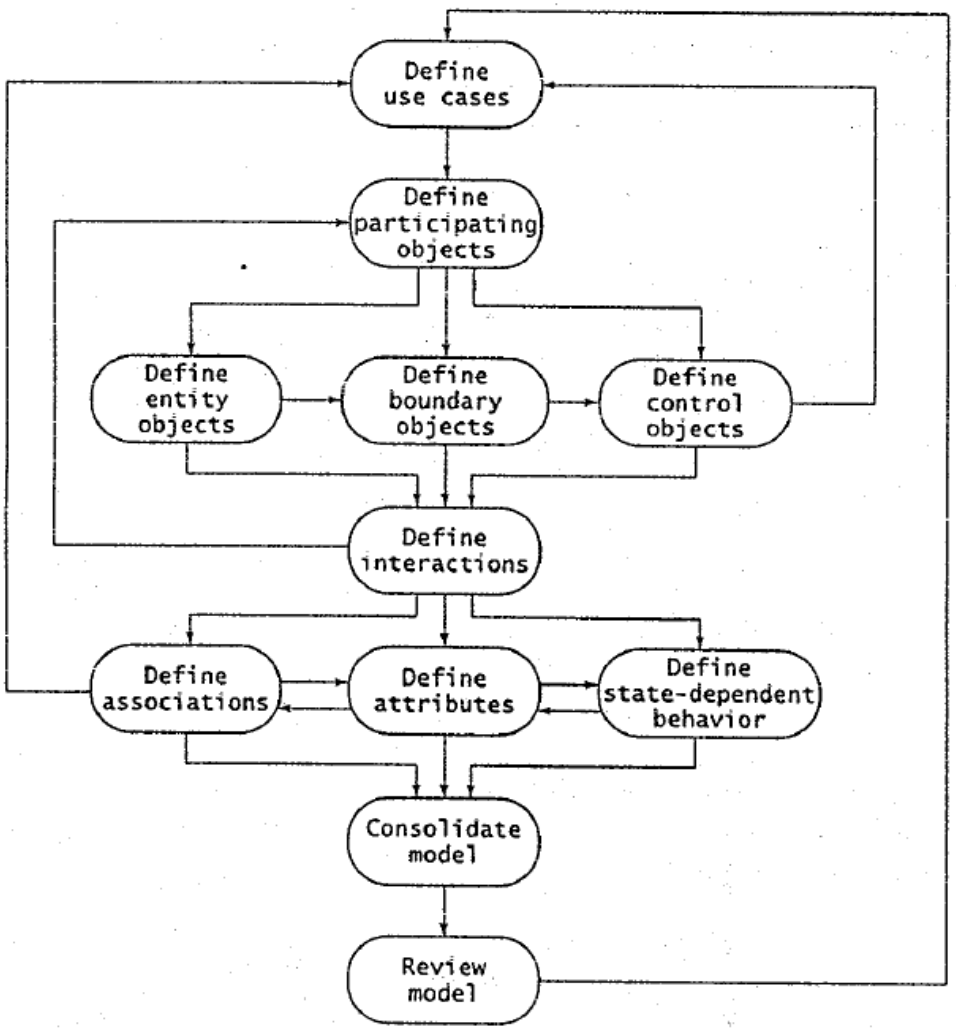


Figure 5-19 Analysis activities (UML activities diagram).

Managing Analysis – **Documenting Analysis**

- The primary challenge in managing requirements is to maintain consistency while using so many resources.
- The requirements elicitation and analysis activities are documented in the Requirements Analysis Document (RAD). During analysis the sections of the above document are revised as ambiguities and new functionality are discovered.
- The **object model** documents in detail all the objects identified with their attributes and associations.
- The **dynamic model** documents the behaviour of the object model by means of statechart diagrams and sequence diagrams.
- The revision history of the RAD will provide a history of changes, including the author responsible for each change.

Managing Analysis – **Documenting Analysis_2**

Requirements Analysis Document

1. Introduction
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - 3.4.3 Object model
       - 3.4.3.1 Data dictionary
       - 3.4.3.2 Class diagrams
     - 3.4.4 Dynamic models
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

Figure 5-20 Overview outline of the Requirements Analysis Document (RAD). See Figure 4-16 for a detailed outline.

Assigning Responsibilities

- Analysis requires the participation of a wide range of individuals. The target user provides application domain knowledge. The client funds the project and coordinates the user side of the effort. The analyst elicits application domain knowledge and formalizes it. Developers provide feedback on feasibility and cost. The project manager coordinates the effort on the development side. For large systems, many users, analysts and developers may be involved, introducing additional challenges during integration and communication requirements. **These challenges can be met by assigning well-defined roles and scopes to individuals**. There are **three main types of roles**: generation of information, integration and review.

SE is … more than programming

Software engineering

… includes programming, but is more than programming

As von Clausewitz did not write: "the continuation of programming through other means".