Chapter2. Software Process and Other Models

System Modelling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Use of System Models

- As a means of facilitating discussion about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an existing system
  - Models should be an accurate representation of the system but need not be complete.
- As a detailed system description that can be used to generate a system implementation
  - Models have to be both correct and complete.

System Perspectives

- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

System Perspectives

- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

2.1 The Software Process Model

- Software process model (SPM) describes the processes that are done to achieve software development

- Software process models can be descriptive; they can describe what has happened in a development project. This can be useful in terms of identifying problems in the software development process.
- Software process models can be prescriptive; the software process model can describe what is supposed to happen. This can be used as training tools for new hire, for reference for uncommon occurrences, and for documenting what is supposed to be happening.

- A software process model usually includes the following:
  - Tasks
  - Artifacts (files, data, etc.)
  - Actors
  - Decisions (optional)
- The notations used can vary.

2.1 The Software Process Model

- The standard software process model
  - ovals for tasks and processes
  - rectangles for artifacts
  - stick figures for actors
- Many software process models do not include decisions. We will use diamonds whenever we show decisions.
- The flow is shown by arcs and is usually left-to-right and top-down.

- The following are rules and interpretations for correct process models:
  - Two tasks cannot be connected by an arc. Tasks must be separated by artifacts.
  - A task is not executable until its input artifacts exist.
  - There are one or more start tasks and one or more terminal tasks.
  - All tasks must be reachable from the start task.
  - There is a path from every task to the terminal task.

2.1 The Software Process Model

EXAMPLE 2.1

Figure 2-1 is a process model for unit testing software. There are two actors: the tester and the team leader. The unit tester, of course, is responsible for the unit testing. The unit tester uses the source code and the test plan to accomplish the unit testing. The result of this activity is an artifact, the test results. The team leader reviews the test results, and the result of this activity should be the approval of the unit testing. This model does not explicitly show what happens when the process is not successful. It could be inferred that the unit tester keeps testing until he or she is happy. Similarly, if the team leader is not ready to give the approval, then the process may be backed up to redo the unit testing.

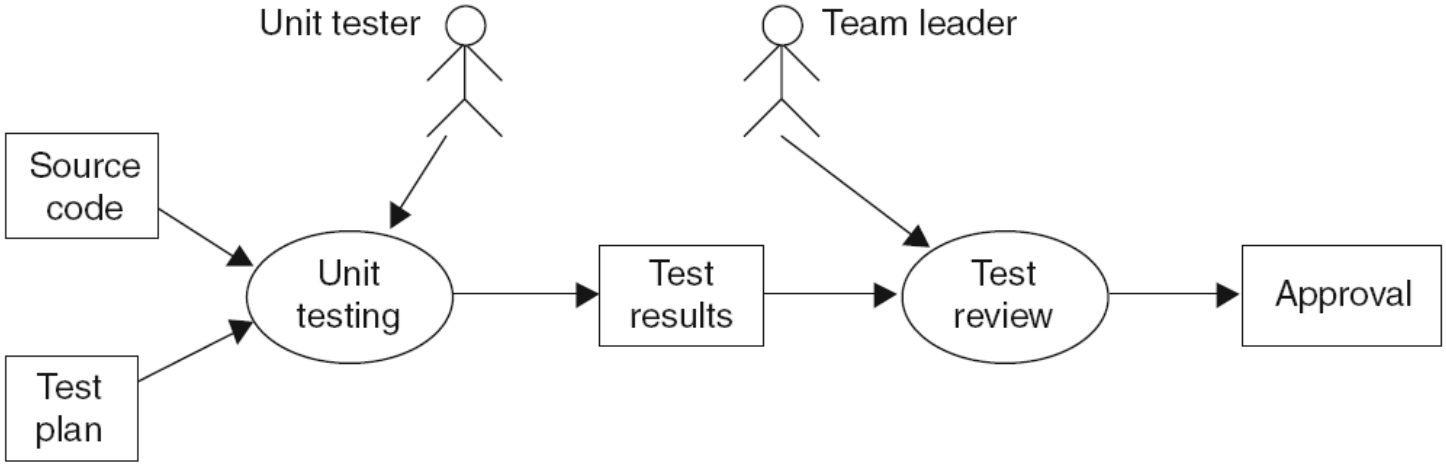


Fig. 2-1. Process diagram for unit testing.

2.1 The Software Process Model

EXAMPLE 2.2

Draw the process model showing decisions.

Adding decisions allows the process model to be more explicit about what happens in all circumstances, as shown in Fig. 2-2.

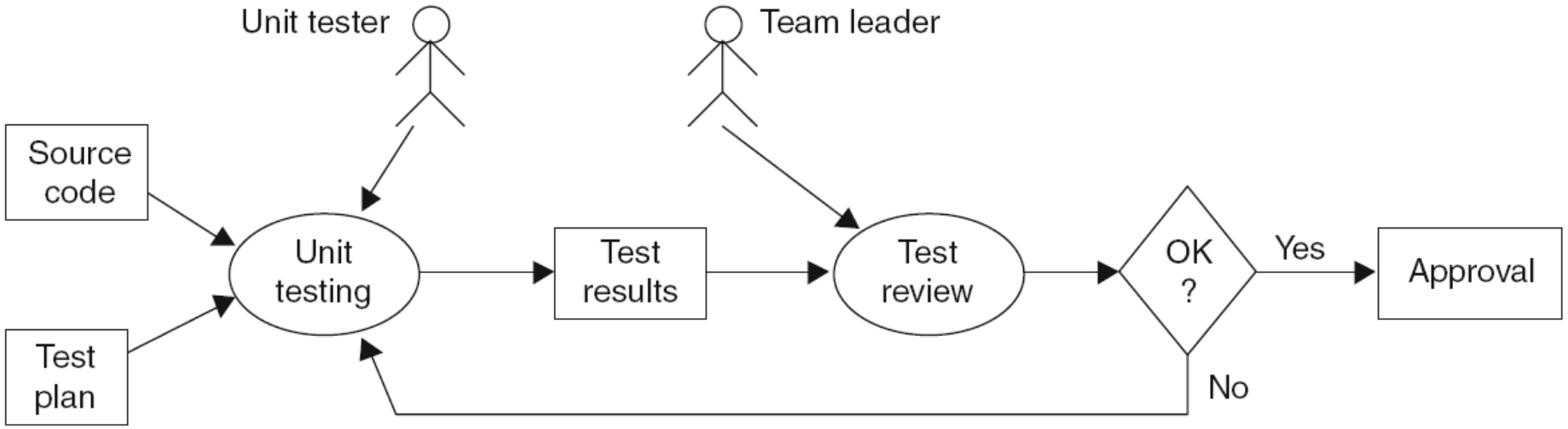


Fig. 2-2. Process model with decisions.

2.2 Data Flow Diagrams

- One of the most basic diagrams in software development is the data flow diagram.
- A data flow diagram shows the flow of the data among a set of components.
- The components may be tasks, software components, or even abstractions of the functionality that will be included in the software system.
- The actors are not included in the data flow diagram.
- The sequence of actions can often be inferred from the sequence of activity boxes.

- The following are rules and interpretations for correct data flow diagrams:
  1. Boxes are processes and must be verb phrases.
  2. Arcs represent data and must be labeled with noun phrases.
  3. Control is not shown. Some sequencing may be inferred from the ordering.
  4. A process may be a one-time activity, or it may imply a continuous processing.
  5. Two arcs coming out a box may indicate that both outputs are produced or that one or the other is produced.

2.2 Data Flow Diagrams

EXAMPLE 2.3

The unit testing example from the previous section can be depicted as a data flow diagram, as shown in Fig. 2-3.

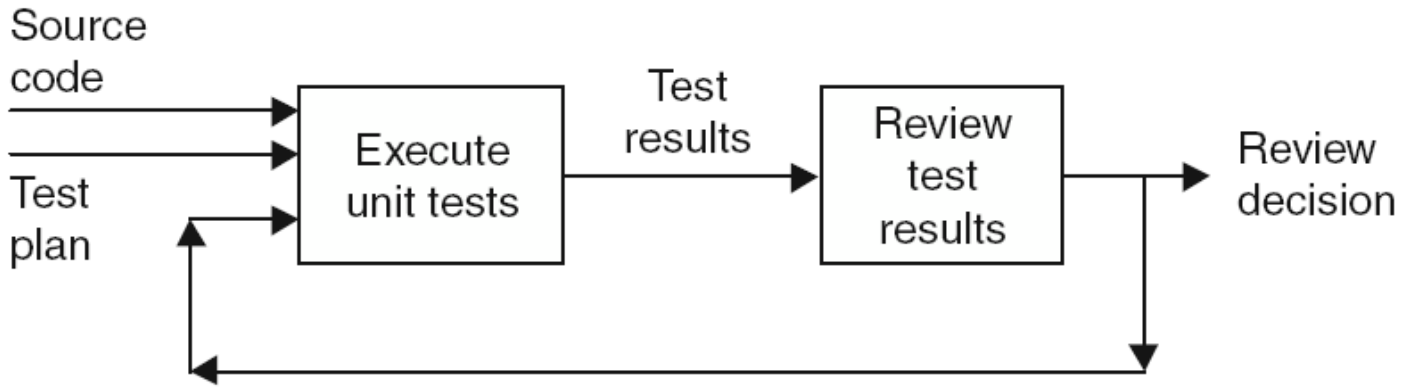


Fig. 2-3. Data flow for unit testing.

Figure 2-3 illustrates some of the rules. The phrases within the boxes are verb phrases. They represent actions. Each arrow/line is labeled with a noun phrase that represents some artifact.

The data flow diagram does not show decisions explicitly. The example shows that the results of testing can influence further testing and that the results of the test review action can also affect the testing (or retesting).

2.2 Data Flow Diagrams

EXAMPLE 2.4

The calculation of the mathematical formula (x+y) * (w+z) can be shown as a sequence of operations, as shown in Fig. 2-4:

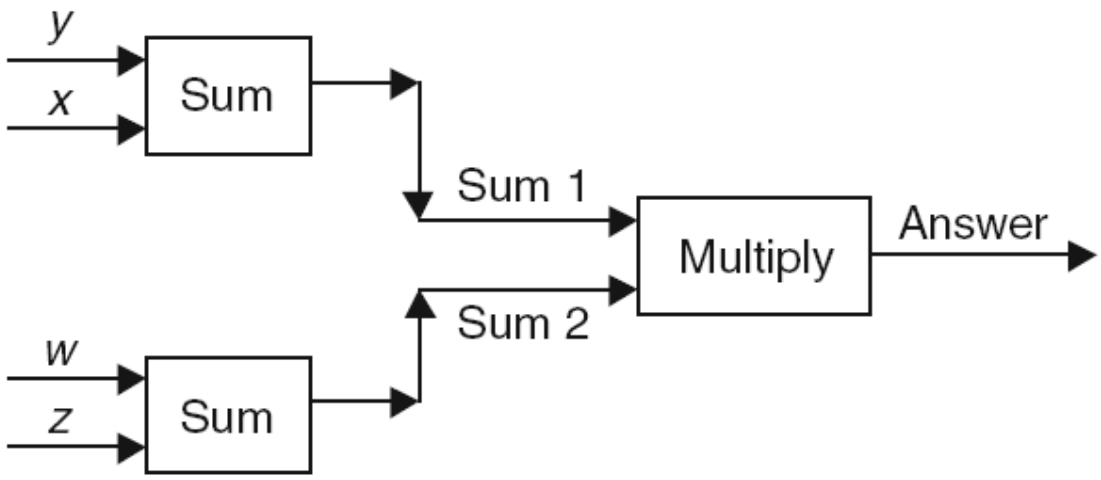
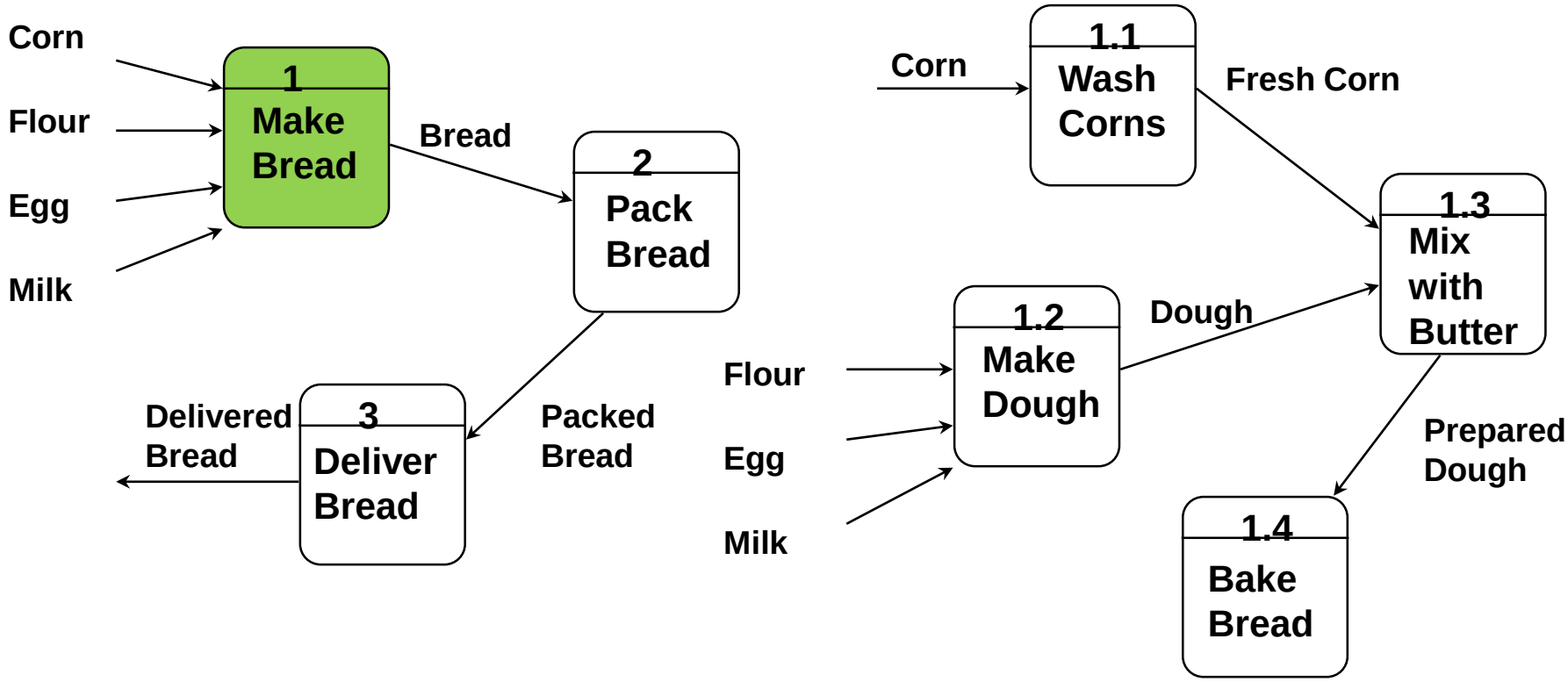


Fig. 2-4

2.2 Data Flow Diagrams

EXAMPLE - Data Flow Diagram for Bread Company



2.3 Petri Net Models

- The basic petri net model consists of condition nodes, arcs, event nodes, and tokens.
- If the input condition nodes for an event node all have tokens, then the event can fire, the tokens are removed from the input nodes, and tokens are placed on all of the output nodes of the firing position.
- The condition nodes are usually represented by circles and the event nodes by horizontal lines or rectangles.

- The condition nodes usually represent some required condition.
- A token at the condition means that the condition is met.
- An event node (the horizontal line) represents an event that can happen (fire) when all the requirements are met.

2.3 Petri Net Models

EXAMPLE 2.5

A petri net model of testing is shown in Fig. 2-5.

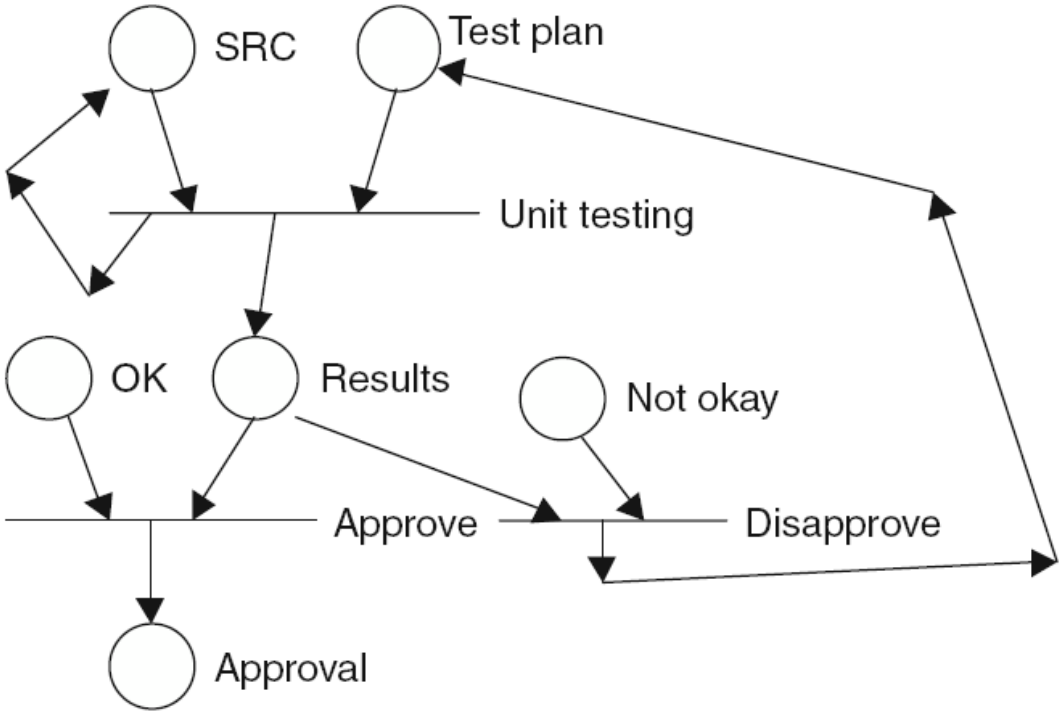


Fig. 2-5. Petri net model.

2.4 Scenarios

- A scenario is a description of one sequence of actions that could occur in this problem domain.

EXAMPLE 2.11

Write a scenario for the library problem.

Fred, a patron, goes to the library and checks out a book. Two months later, he brings the overdue library book back to the library.

UML Diagram Types

- Activity diagrams, which show the activities involved in a process or in data processing .
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, which show the object classes in the system and the associations between these classes.
- State diagrams, which show how the system reacts to internal and external events.

2.5 Use Case Diagrams

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML. A use case diagram is part of the UML set of diagrams.

- Each use case represents a discrete task that involves external interaction with a system.

- It shows the important actors and functionality of a system. Actors are represented by stick figures and functions by ovals. Actors are associated with functions they can perform.

- Actors in a use case may be people or other systems.
- Represented diagramatically to provide an overview of the use case and in a more detailed textual form.

2.5 Use Case Diagrams

EXAMPLE 2.10

Draw a use case diagram for the simple library. (See Fig. 2-10.)

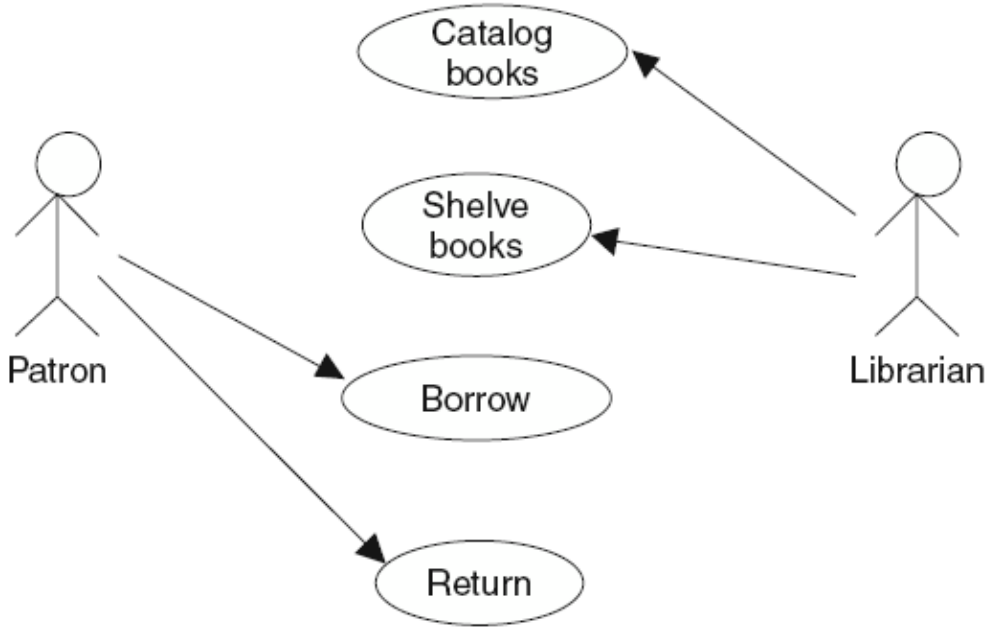


Fig. 2-10. Use case for simple library.

The functions in the ovals are methods of the classes in the object model. The patron object can borrow and return copies. The librarian actor is not an object on the object model. The librarian in the use case shows that some functions—for instance, catalog and shelve books—are not functions available to the patron.

2.6 Object Models

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object. When you are developing models during the early stages of the software engineering process, objects represent something in the real world.

- Object models represent entities and relationships between entities.
- Each box represents a type of object, and the name, attributes, and the methods of the object are listed inside the box.
- The top section of the box is for the name of the object, the second section is for the attributes, and the bottom section is for the methods.
- An arc between two objects represents a relationship between the objects. Arcs may be labeled in the center with a name of the association.
- At each end a multiplicity may be given indicating how many different associations of the same kind are allowed.

2.4 Object Models

- An association is a link between classes that indicates that there is some relationship between these classes.
- The three major types of relationships are (1) inheritance, (2) aggregation, and (3) association.
- An inheritance relationship implies that the object at the bottom of the arc is a special case of the object at the top of the arc. For example, the top object might be a vehicle and the bottom object a car, This is often called an ‘‘is-a’’ relationship.
- An aggregation relationship implies that the object at the bottom of the arc is a component of the object at the top of the arc. For example, the top object might be a car and the bottom object might be the engine. This is often called a ‘‘part-of’’ relationship.
- Association, this arc implies that somehow one of the objects is associated with the other object. For example, a ‘‘father-son’’ relationship is an association.

2.6 Object Models

EXAMPLE 2.6

Construct an object model for a library. The objects in the simple library shown in Fig. 2.6 consist of the library, books, copies of books, and patrons. None of the methods of the objects are shown. The library has an aggregation relationship with book and with patron. That is, the library is really made up of books and patrons. The relationship between book and copy is neither aggregation nor inheritance. The object book represents the abstraction of a book, while the copy is the physical item that is loaned out. The relationship between patron and copy is called ‘‘loan.’’ From the view of copy, the role is ‘‘checked out by’’ and from patron the role is ‘‘checkout.’’ The multiplicities indicate that a copy can either not be checked out or can have this relationship with only one patron at a time (‘‘0.1’’). The other multiplicity, ‘‘0.*’’, indicates that a patron can have zero or one or many relationships of ‘‘checkout’’ at a time.

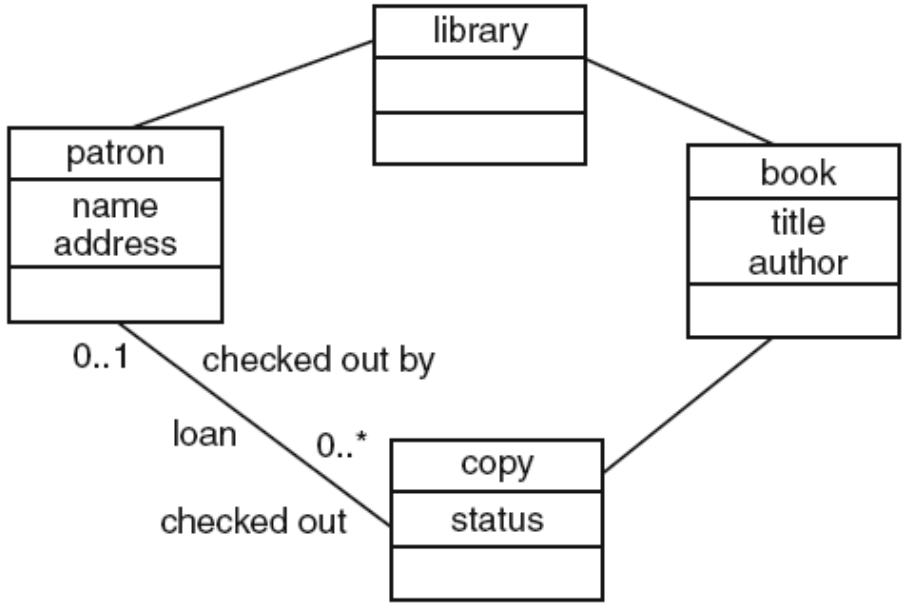


Fig. 2-6. Object model of simple library.

2.6 Object Models

EXAMPLE 2.7

Construct an object model for a family-tree system that stores genealogical information about family members.

Figure 2-7 indicates that everyone has a birthfamily. Every marriage has a father person and a mother person. Many attributes have been left off the diagram, and no functions are shown.

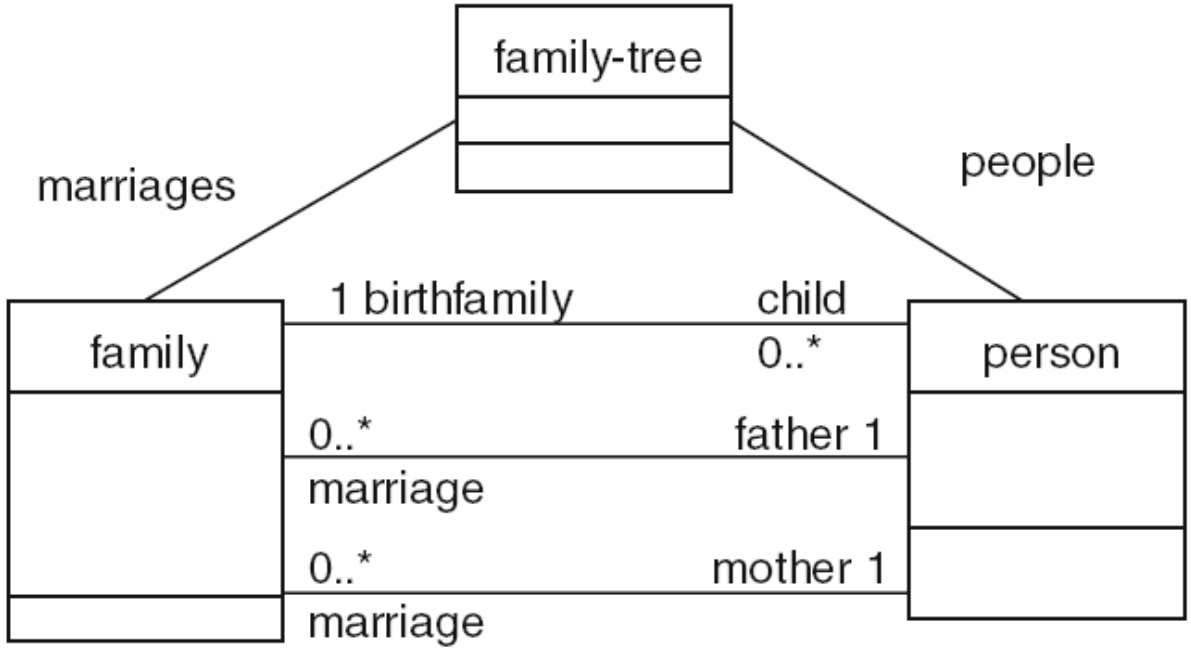


Fig. 2-7. Family-tree object model

2.6.2 Instance Diagrams

- Object diagrams represent types of objects. Thus, a box labeled ‘‘car’’ represents the attributes and functions of all cars.
- Sometimes the relationships between instances of objects are not very clear in an object diagram. An instance diagram shows example instances of objects and may clarify the relationships.

2.6.2 Instance Diagrams

EXAMPLE 2.9

Draw an instance model showing Fred, his wife Sue, their children Bill, Tom, and Mary, and his parents Mike and Jean. (See Fig. 2-9.)

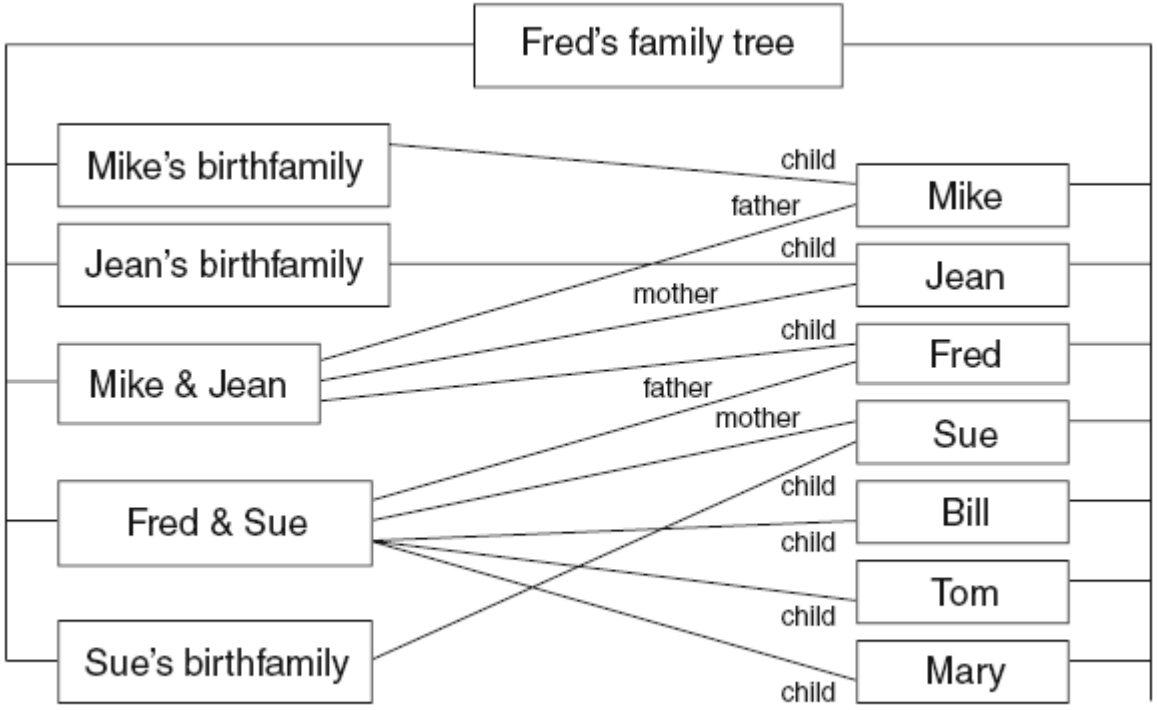


Fig. 2-9. Instance diagram of Fred's family tree.

2.7 Sequence Diagrams

- A sequence diagram is part of the UML set of diagrams and is used to model the interactions between the actors and the objects within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

- The diagram has vertical lines, which represent instances of classes. Each vertical line is labeled at the top with the class name followed by a colon followed by the instance name.

- Horizontal arrows depict function calls. The tail of the arrow is on the line of the calling class, and the head of the arrow is on the line of the called class. The name of the function is on the arrow. The wide block on the vertical line shows the execution time of the called function.

2.7 Sequence Diagrams

EXAMPLE 2.12

Draw a sequence diagram for the scenario of Example 2.11. (See Fig. 2-11.)

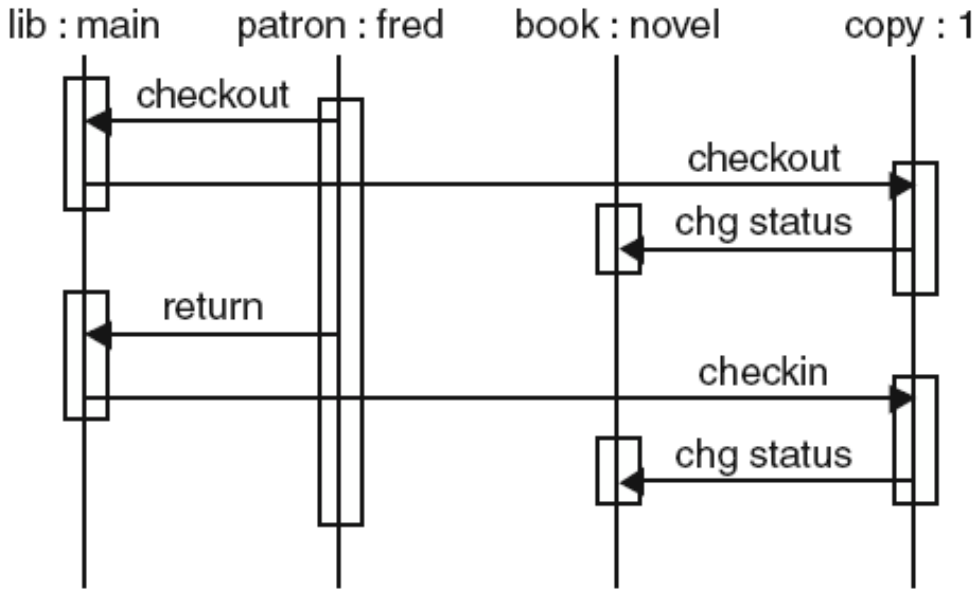


Fig. 2-11. Sequence diagram for checkout scenario.

This diagram is much closer to the design phase than the object model presented in Example 2.8. There are functions used in this diagram that are not represented in the earlier object model. Also, the sequence of calls represented in this diagram is dependent on the actual design.

2.8 State Diagrams

- These model the behaviour of the system in response to external and internal events.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- Statecharts are an integral part of the UML and are used to represent state machine models.

- The state of a machine or program is the collection of all the values of all the variables, registers, and so on. A state diagram shows the states of the system and the possible transitions between these states.
- These states can be diagrammed to show the transitions between the states.
- The following are rules for correct state diagrams:
  1. There is one initial state.
  2. Every state can be reached from the initial state.
  3. From each state, there must be a path to a stop state.
  4. Every transition between states must be labeled with an event that will cause that transition.

2.8 State Diagrams

EXAMPLE 2.15

Draw a state diagram for a fixed-size stack. (See Fig. 2-14.)

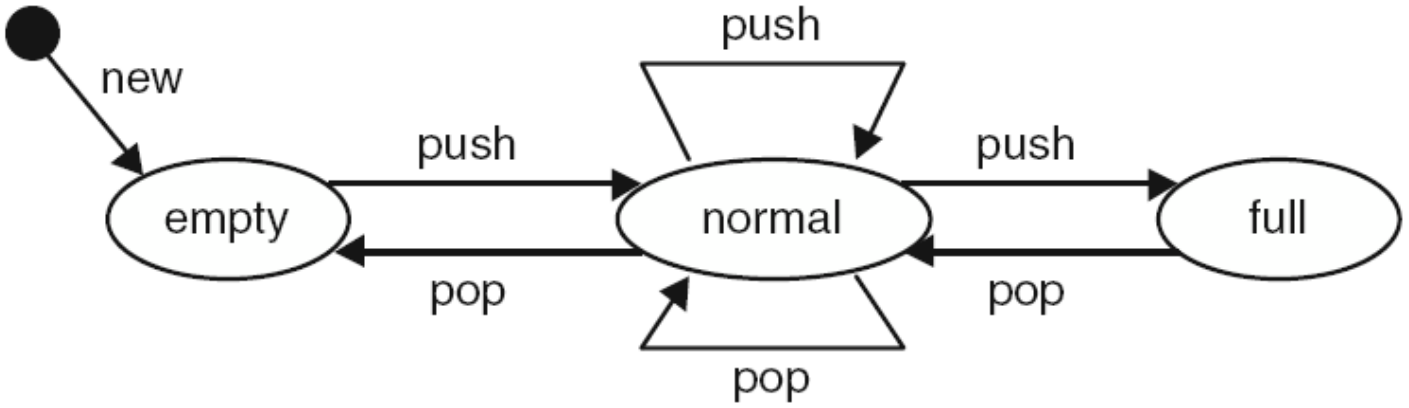


Fig. 2-14. State diagram for a fixed-size stack.

There are two approaches to state diagrams. Only legal or nonerror transitions are specified. Another approach is to show all transitions, including transitions that cause errors.

2.8 State Diagrams

EXAMPLE 2.16

Draw a state diagram for a stack with all the error transitions shown. (See Fig. 2-15.)

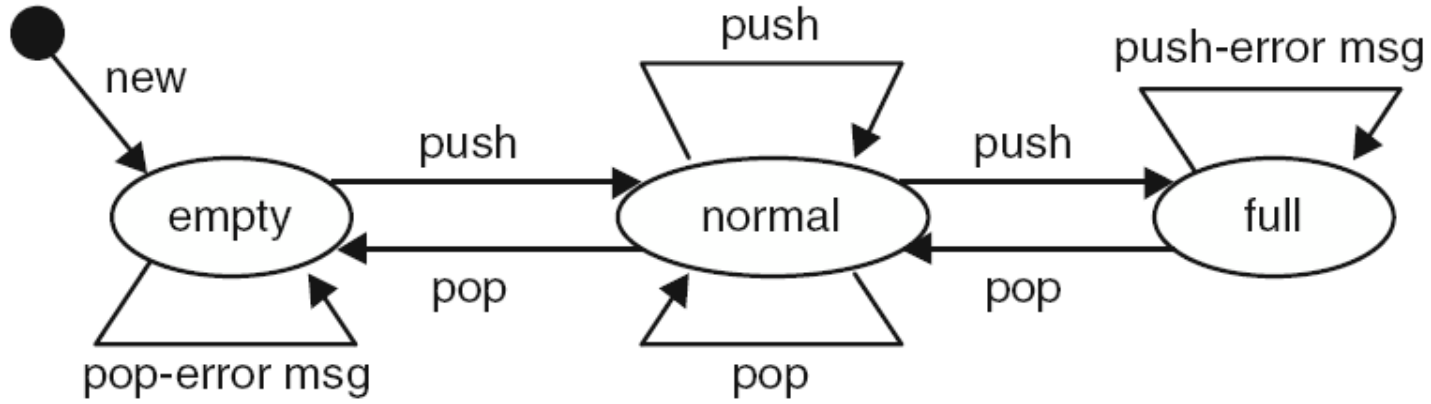


Fig. 2-15. State diagrams showing error transitions.

2.9 Activity Diagrams

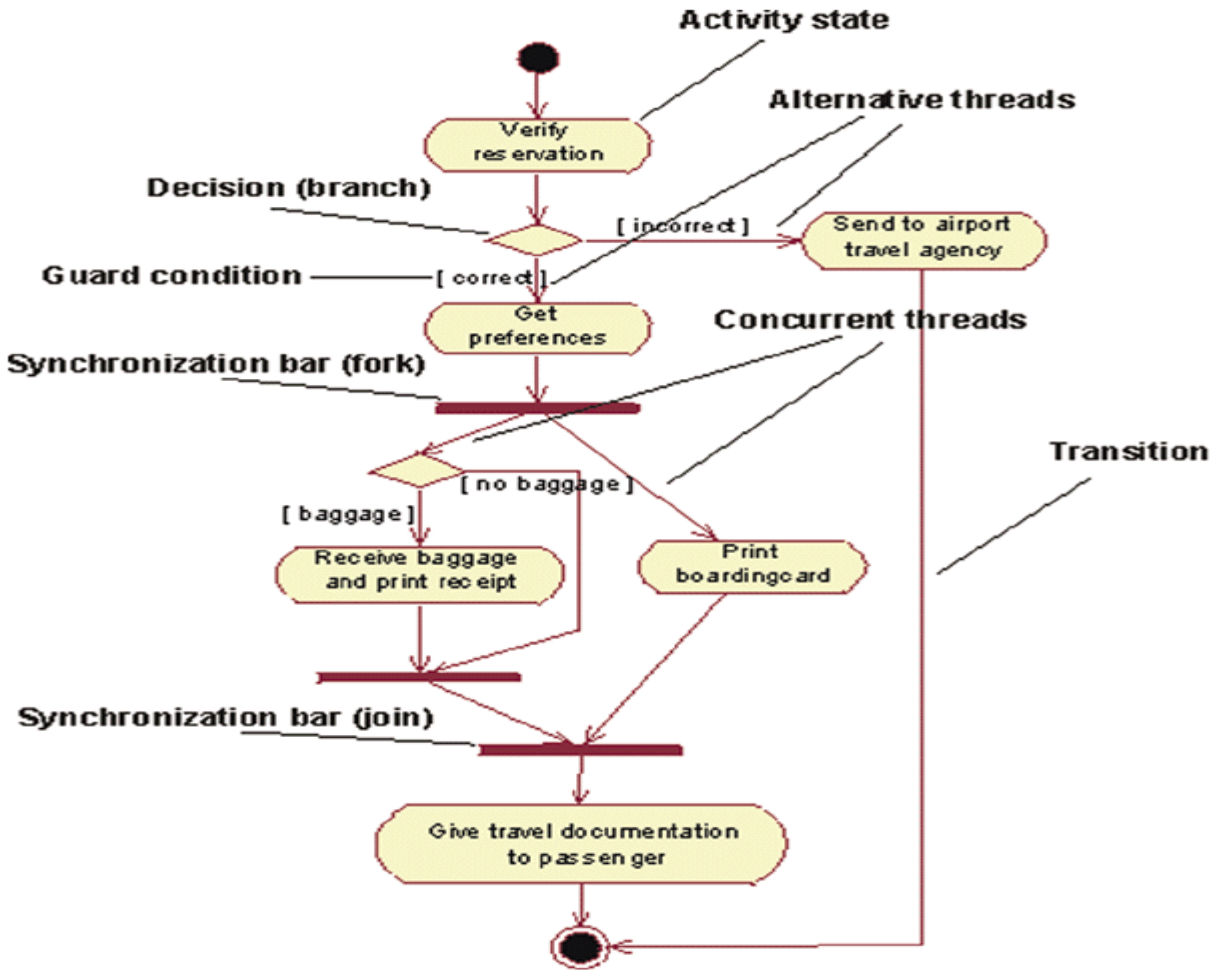
- These are graphical representation of workflows od stepwise activities and actions with support for choice, iteration and concurrency.
- Activity diagrams may be regarded as a form of flowchart.

- Notations
  - Actions : rounded rectangles
  - Decisions : diamonds
  - Start or end of concurrent activities : bars
  - Start of the workflow : a black circle
  - Final state : an encircled black circle
  - Order in which activities happen : arrows

2.9 Activity Diagrams

EXAMPLE 2.16

Draw a activity diagram for an individual check-in in the business usecase model of airport check-in.



2.10 Control Flow Graphs

- A control flow graph (CFG) shows the control structure of code. Each node (circle) represents a block of code that has only one way through the code.
- Arcs between nodes represent possible flows of control.
- If it is possible that block B is executed, right after block A, then there must be an arc from block A to block B.
- The following are rules for correct control flow diagrams:
  1. There must be one start node.
  2. From the start node, there must be a path to each node.
  3. From each node, there must be a path to a halt node.

2.10 Control Flow Graphs

EXAMPLE 2.14

Draw a control flow graph for the following triangle problem.

```
read x,y,z;
type = ''scalene'';
if (x == y or x == z or y == z) type =''isosceles'';
if (x == y and x == z) type =''equilateral'';
if (x >= y+z or y >= x+z or z >= x+y) type =''not a triangle'';
if (x <= 0 or y <= 0 or| z <= 0) type =''bad inputs'';
print type;
```

In Fig. 2-13, the ''a'' node represents the first two statements and the `if` statement. The ''type = isosceles'' is in the node labeled ''isosceles''. Similarly, the ''c'' node represents the next `if` statement. and the ''equilateral'' node represents the body of the `if`.

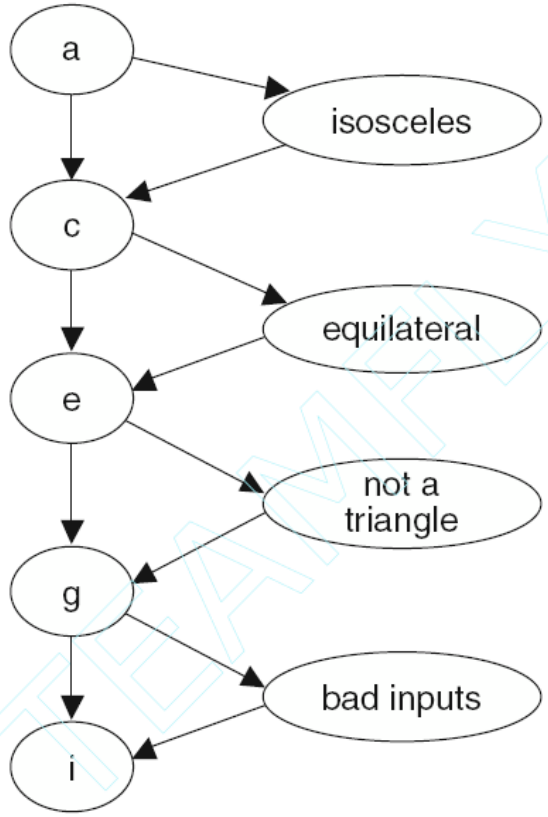


Fig. 2-13. Control flow graph for triangle program.