

OBJECT-ORIENTED ANALYSIS AND DESIGN

Construction and Testing

UNIT III



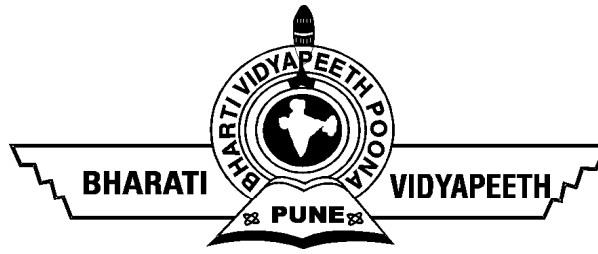
Learning Objectives

Construction

- Introduction
- the design model
- block design
- working with construction

Testing

- Introduction
- on testing
- unit testing
- integration testing
- system testing
- the testing
- process



CONSTRUCTION



Learning Objectives

- What is Construction Phase
- Why Construction
- Add a Dimension
- Artifacts for Construction
- Design (What, Purpose, Goals, Levels)
- Implementation Environment
- Traceability
- Interaction Diagram
- Block design
- Block Behavior
- Implementation



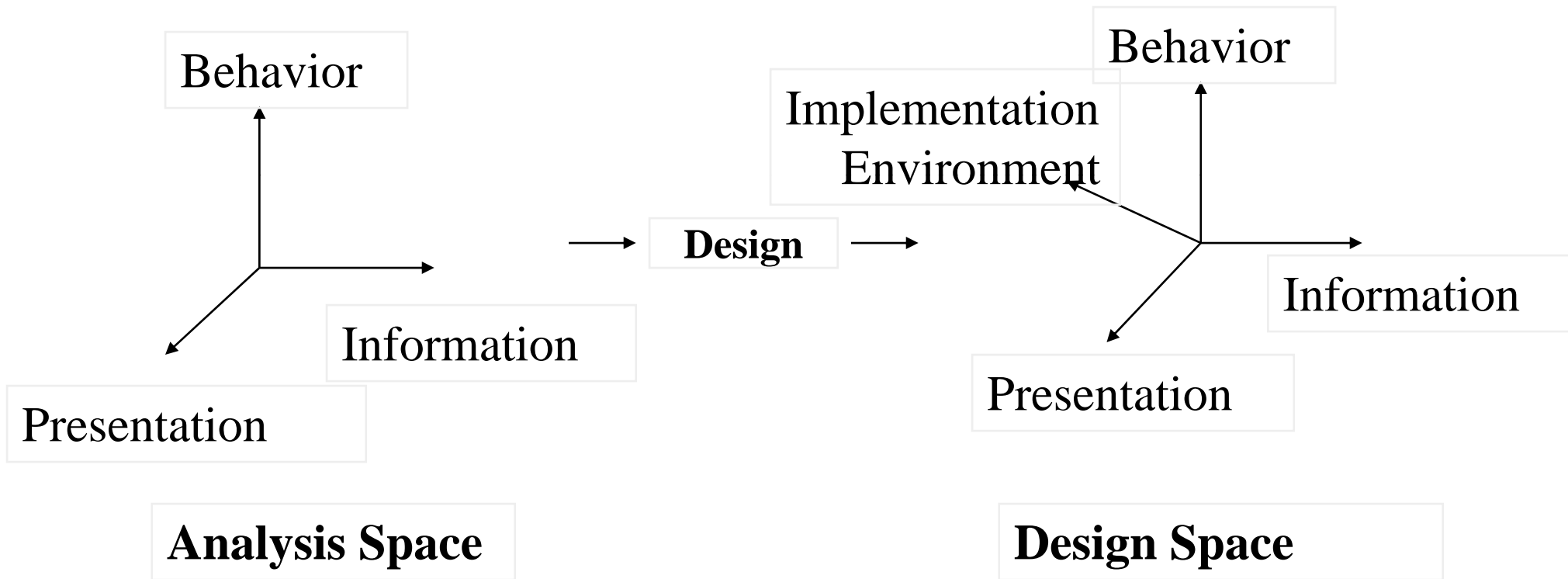
What is Construction Phase?

- All about “BUILDING” the system from model of analysis & requirement phase.
- Consists of **Design and Implementation.**
- Start from elaboration & continues to construction

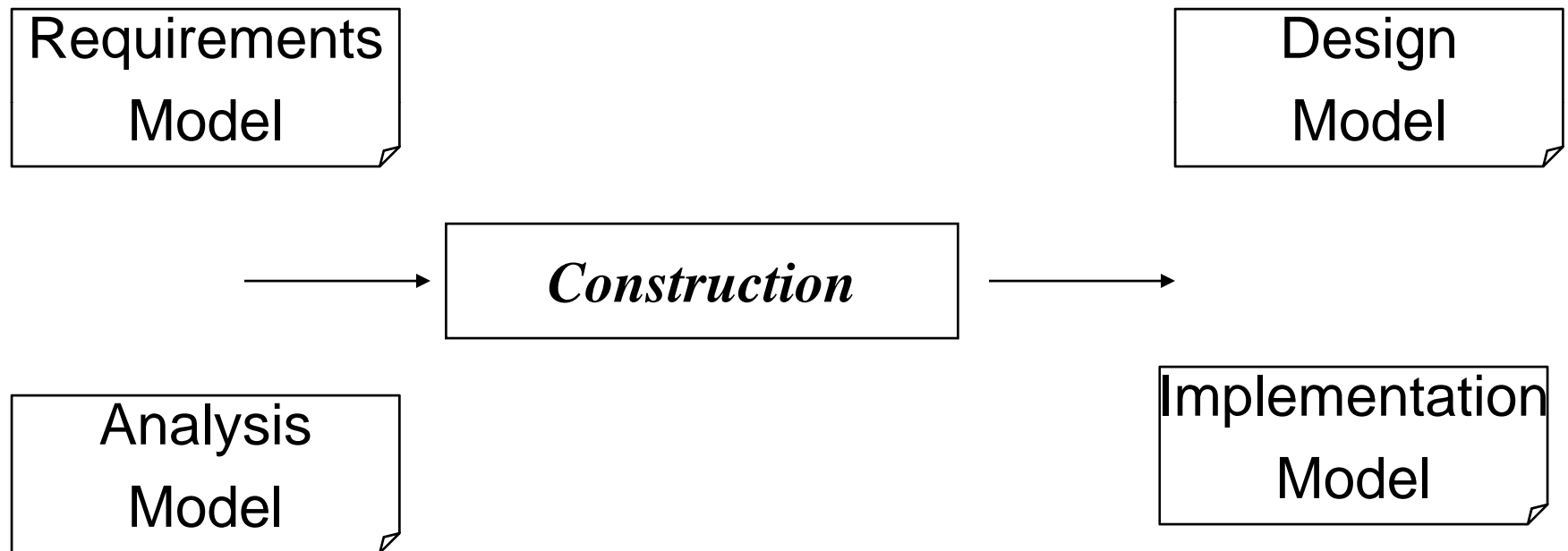
Why construction?

- For seamless transition to source code; analysis model not sufficient.
- The actual system must be adapted to the implementation environment.
- Must explore into more dimensions.
- To validate the analysis result.

Add A Dimension: Analysis To Design Space



Artifacts for Construction



Input & Output Models of Construction

Design

“There are two ways of constructing a software design:

- make it so simple that there are obviously no deficiencies
- make it so complicated that there are no obvious deficiencies.”

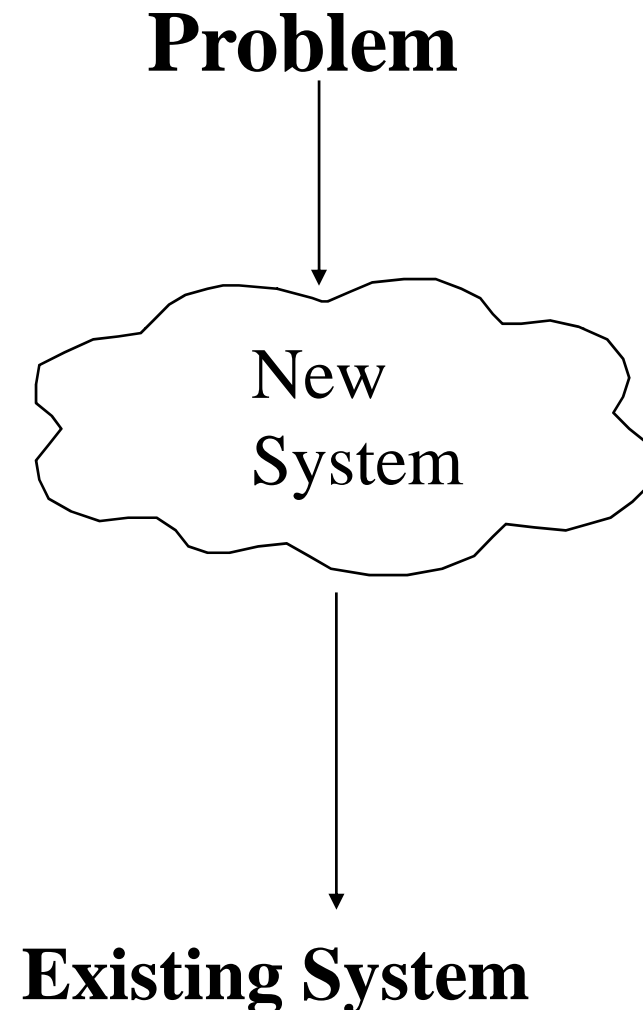
- C.A.R. Hoare

What is Design?

- Specification Is about What, and Design is the start of the How
- Inputs to the design process
 - Specification document, including models etc.
- Outputs of the design process
 - A design document that describes how the code will be written. Includes design models!
 - ✓ **What subsystems, modules or components are used**
 - ✓ **How these integrate (i.e. work together)**
 - Information allowing testing of the system

The Purpose of System Design

- Bridging the gap between desired and existing system in a manageable way
- Use Divide and Conquer
 - We model the new system to be developed as a set of subsystems



Why is Design so Difficult?

Analysis: Focuses on the application domain

Design: Focuses on the solution domain

- Design knowledge is a moving target
- The reasons for design decisions are changing very rapidly

✓ **Halftime knowledge in software engineering**

✓ **What I teach today will be out of date in 3 years**

✓ **Cost of hardware rapidly sinking**

Design Goals..

Qualities Of A Good Design:

- **Correct, Complete, Changeable, Efficient, Simple**

Correctness:

- It Should Lead To A Correct Implementation

Completeness:

- It Should Do Everything. Everything? It should follow the specifications.

Changeable:

- It Should Facilitate Change—Change Is Inevitable

Design Goals..

Efficiency

- It Should Not Waste Resources. But:
- Better A Working Slow Design Than A Fast Design That Does Not Work

Simplicity

- It Should Be As Understandable As Possible
- **Important: A design should fully describe how coders will implement the system in the next phase**
 - **Designs are blue-prints for code construction**

Levels of Design

Three possible levels:

- ✓ **System Design, if appropriate**
 - **Part of Systems Engineering**
- ✓ **High-level Software Design**
 - **Architecture, architectural design**
- ✓ **Low-level Software Design**
 - **Detailed Design, Module Design**

Systems Engineering: Large combinations of hardware and software.

- Decompose system into subsystems
- Determine which subsystems are HW, which are SW
- Software Engineering activities thus become a part of a larger activity.

Develop Design Model

- Create detailed “plans” (like blueprints) for implementation
- Identify the “Implementation Environment” & draw conclusions.
- Incorporate the conclusions & develop a “**First approach to a design model**” from requirement models.
 - Use analysis model as base & translate analysis objects to design objects in design model fit for current implementation.
 - Why can't this be incorporated in analysis model?

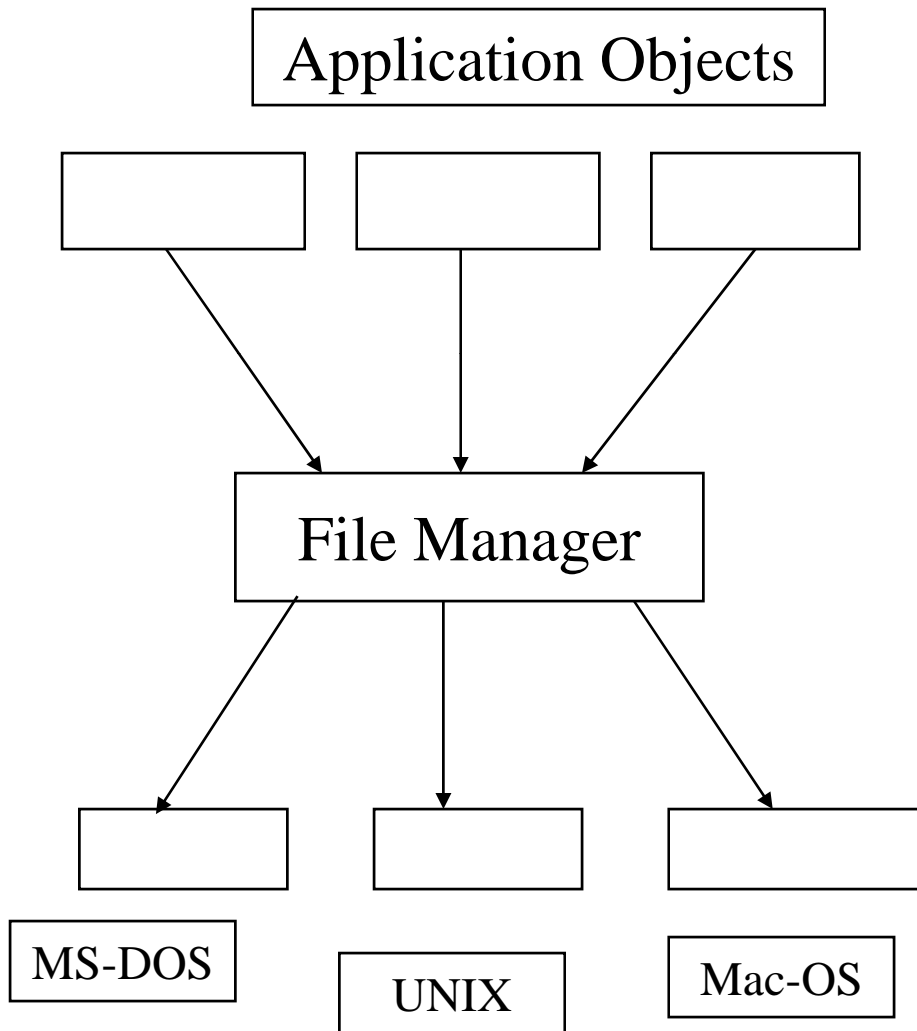
Develop Design Model

- Describe how the “**Object Interact**” in each specific use case & how stimuli between objects is exchanged
- **Create design models before coding so that we can:**
 - Compare different possible design solutions
 - Evaluate efficiency, ease of modification, maintainability, etc

Implementation Environment

- Identify the actual technical constraints under which the system should be built
- Including:
 - The target environment
 - Programming language
 - Existing products that should be used (DBMSs, etc)
- Strategies:
 - n As few objects as possible should be aware of the constraints of the actual implementation environment

Implementation Environment: Target Environment



**several
implement
ations of
the file
manager
block**

Implementation Environment

Target environment

- Create a new blocks that represent occurrences of the changed parts in the target environment

Strategies:

- Specified an abstract class
 - ✓ polymorphism
 - The object can check the platform at run-time
 - ✓ CASE statement in the source code
 - Decide this when the system is delivered
 - ✓ Provide several different modules which will be chosen later
-
- Investigate whether the target environment will execute in a distributed way
 - on different processors or different processes

Implementation Environment

Programming language

- Affect the design in translating the concepts used
- The basic properties of the language and its environment are fundamental for the design
 - ✓ Inheritance and Multiple inheritance
 - ✓ Typing
 - ✓ Standard
 - ✓ Portability
 - ✓ Strategies for handling errors during run-time
 - Exception (Ada)
 - Assertions (Eiffel)
 - ✓ Memory management
 - Automatic garbage collection
- The use of component
 - ✓ Component library, such as interface objects

Implementation Environment..

Using existing products

- DBMS
- UIMS (User Interface Management System)
- Network facilities
- Internally or externally developed applications that should be incorporated
- Products used during development
 - ✓ **Compilers**
 - ✓ **Debuggers**
 - ✓ **Preprocessor**

Other considerations

- Requirement for performance
- Limitations of memory

Implementation Environment

- Other considerations
 - Strategies:
 - ✓ To postpone optimizations until they are needed or you are absolutely sure that they will be needed
 - the real bottlenecks are often missed and then new optimizations are necessary
 - Use simulation or prototyping to investigate potential optimization problem early
 - ✓ Extensive experiences may help to judge at an early stage
- If you're not sure of the correctness of a performance optimizations, you should not make it until you're sure of how it should be done

Implementation Environment

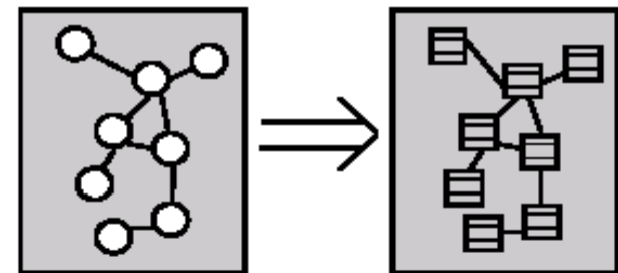
- The people and organization involved in the development could also affect the design
 - The principal strategy:
 - ✓ such factors should not affect the system structure.
 - ✓ The reason: the circumstances (organizations, staffing, competence areas) that are in effect today will probably change during the system's life cycle

Traceability

- refines the analysis model in light of actual implementation environment.
- Explicit definition of interfaces of objects, semantics of operation. Additionally, different issues like DBMS, programming language etc. can be considered.
- The model is composed of “**BLOCKS** ” which are the design objects.
- One block is implemented as one class

Traceability..

- The blocks abstract the actual implementation
- Traceability is extremely important aspect of the system.
 - Changes made will be only local to a module.
 - Provides high functional localization (high cohesion).



Analysis Model:

*logical,
conceptual,
frozen*

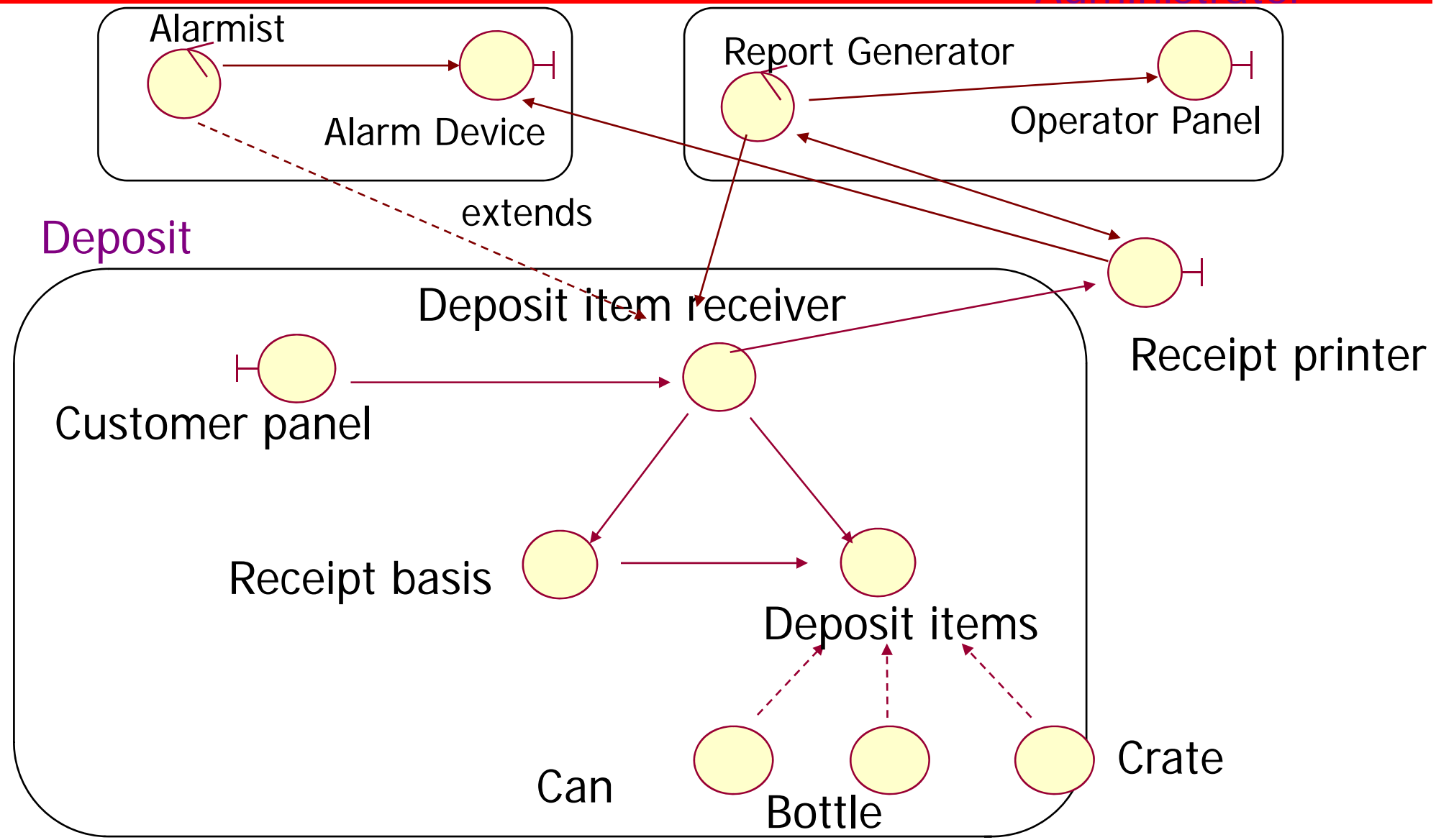
Design Model:

*a practical
abstraction*

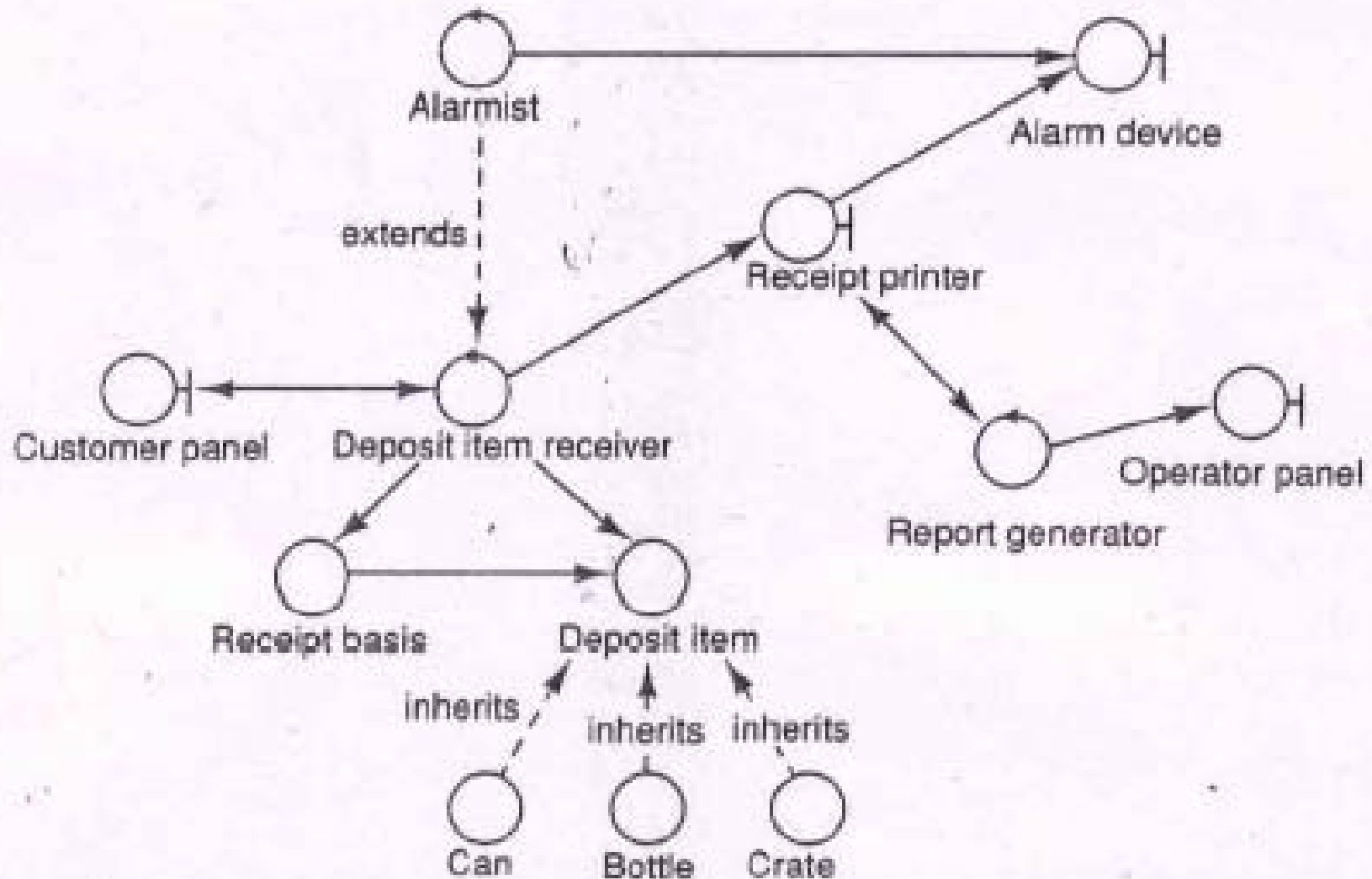
Example Recycling Machine Analysis Model

Alarm

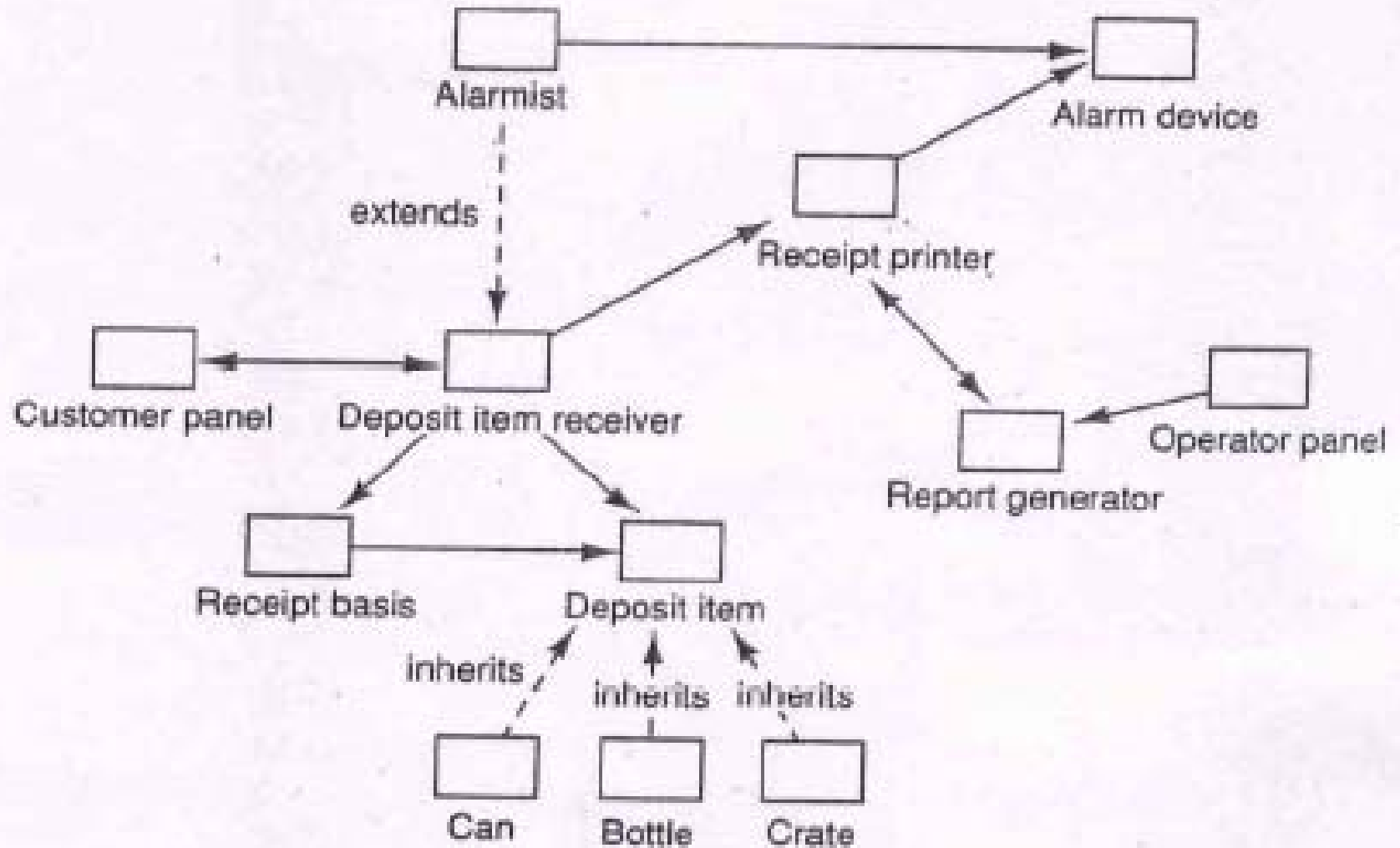
Administrator



Example Recycling Machine Analysis Model



Example Recycling Machine Design Model



Working with the Design Model

- Changes can and should occur, but all changes should be justified and documented (for robustness reason)
- We may have to change the design model in various way:
 - To introduce new blocks which don't have any representation in the analysis model
 - To delete blocks from the design model
 - To change blocks in the design model (splitting and joining existing blocks)
- To change the associations between the blocks in the design model
- Use the component

Change in Environment

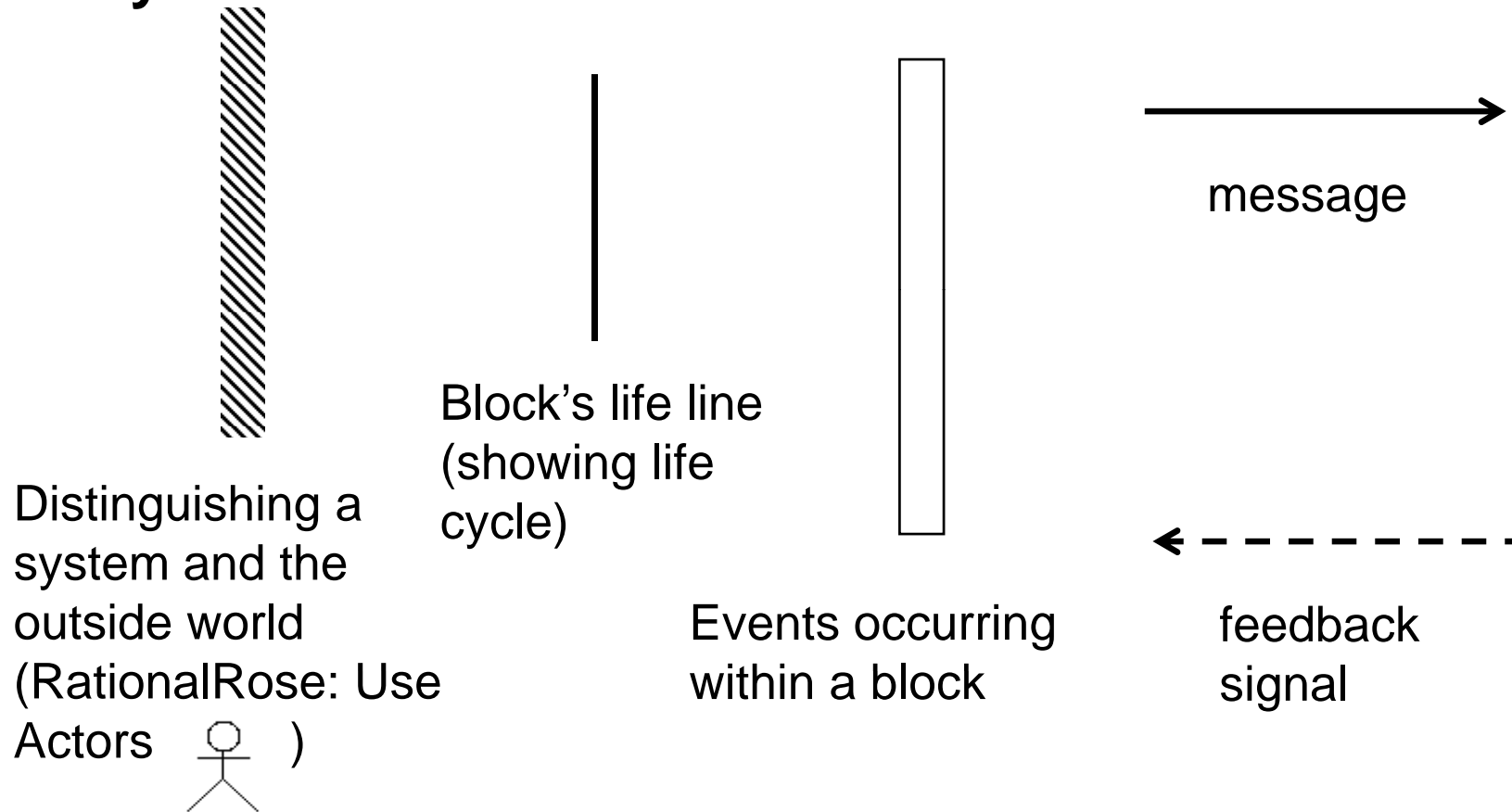
- Changing the associations between the blocks in the design model.
 - **extensions to stimuli**
 - **inheritance to delegation**

Interaction Diagrams

- Describe how the blocks are to communicate by designing the use case
- The main purpose of the use case design is to define the protocols of the blocks
- The interaction diagram describes how each usecase is offered by communicating objects
 - The diagram shows how the participating objects realize the use case through their interaction
 - The blocks send stimuli between one another
 - All stimuli are defined including their parameters
- For each concrete use case, draw an interaction diagram

Sequence Diagram (Cont'd)

Syntax and Semantics



Building an Interaction Diagram

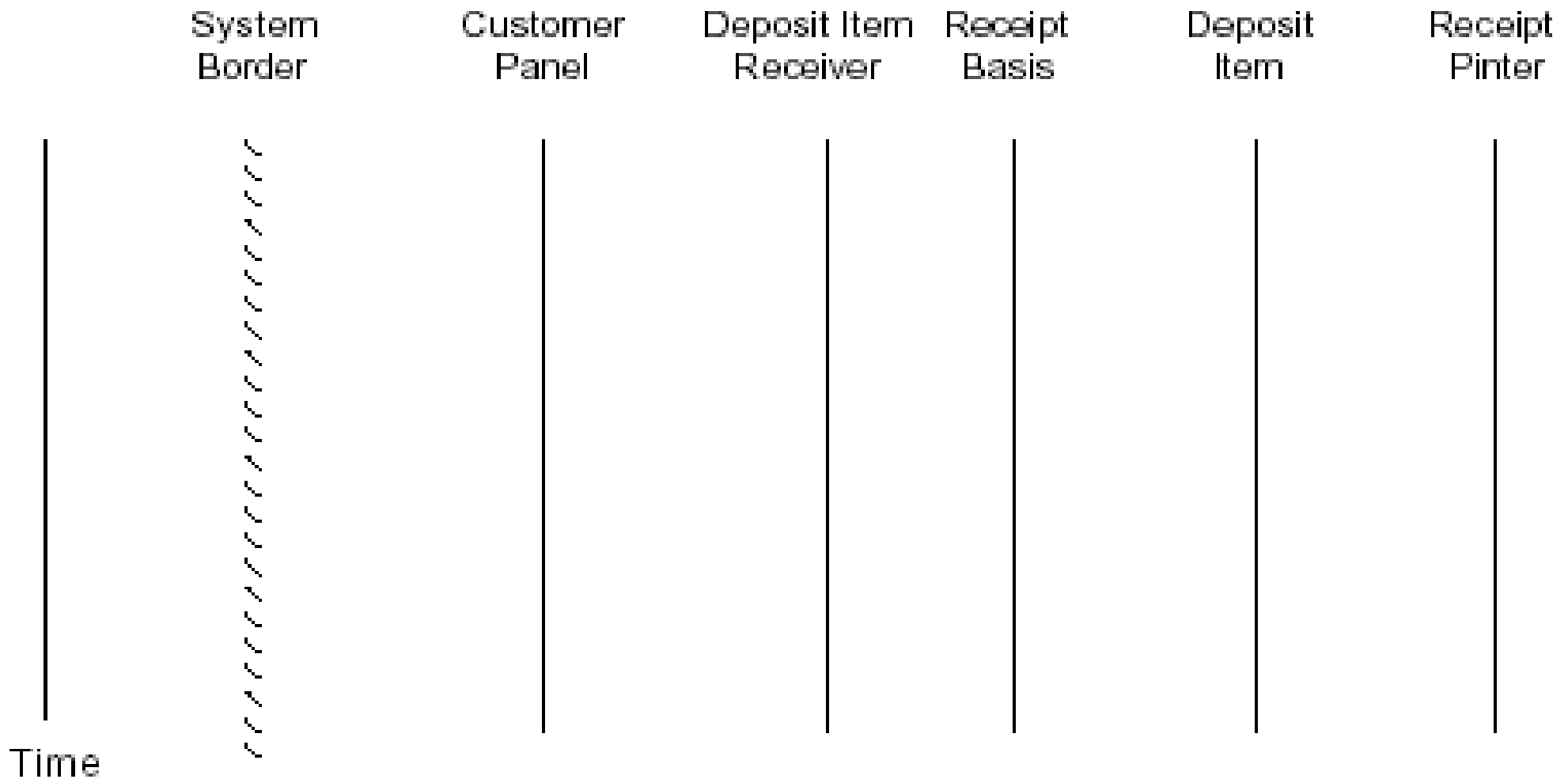
- **Identify blocks**
- **Draw skeleton, consist of:**
 - **System border**
 - **Bars for each block that participates**
- **Describes the sequences**
 - **Structured text or pseudo-code**
- **Mark the bar to which operations belongs with a rectangle representing operation**
- **Define a stimulus**

Building an Interaction Diagram..

- **Draw a stimulus as a horizontal arrow**
 - **Start: bar of the sending block**
 - **End: bar of the receiving block**
- **Structure the interaction diagram**
 - **Fork diagram**
 - **Stair diagram**

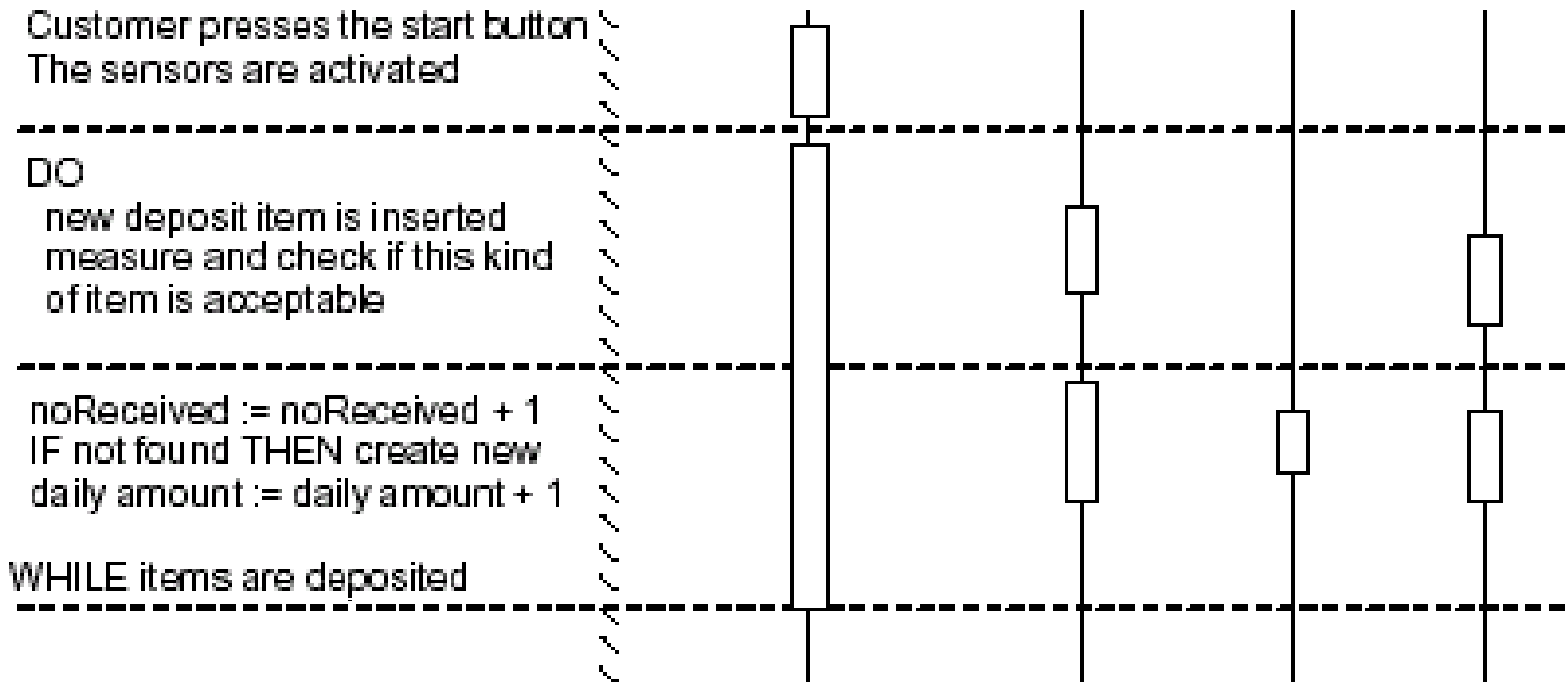


The Skeleton for the Interaction Diagram



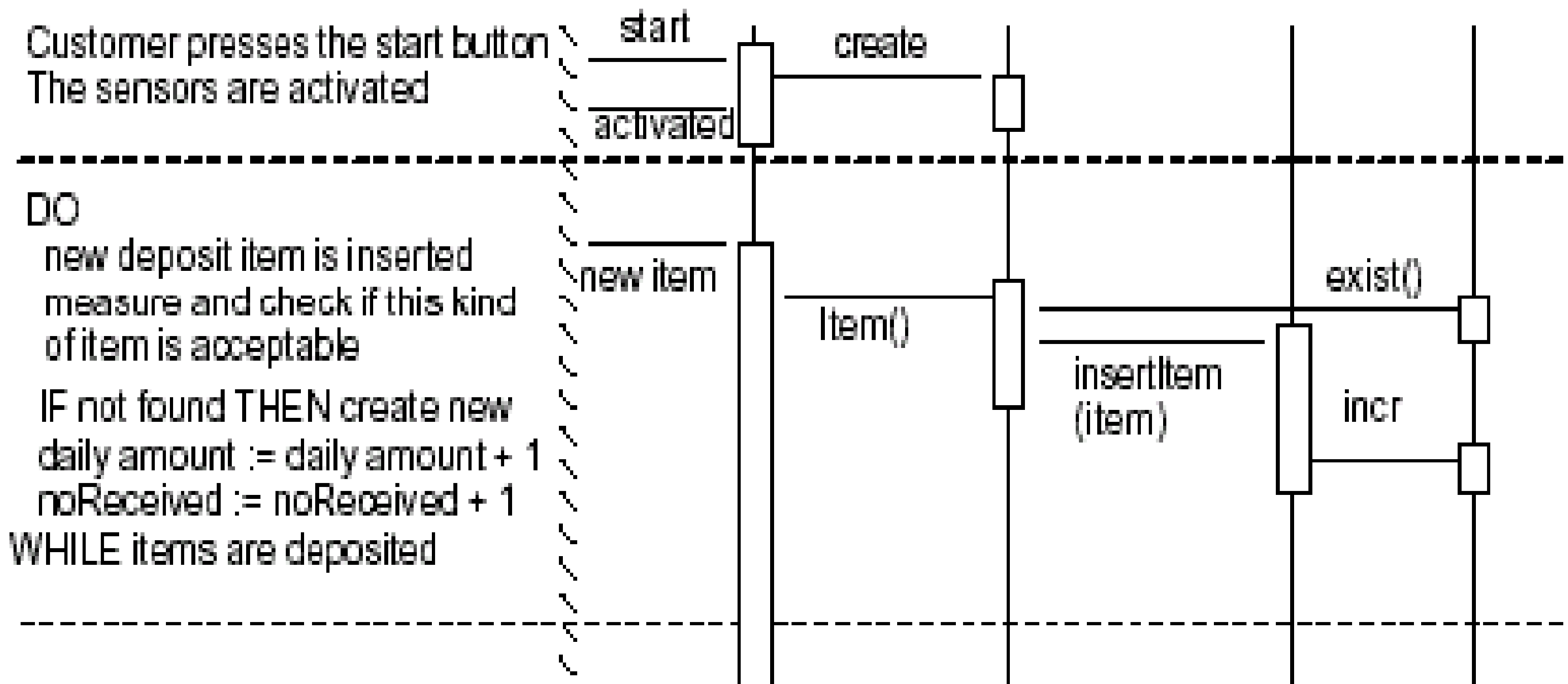
Example – Interaction Diagram for Use Case Returning Item

System Border Customer Panel Deposit Item Receiver Receipt Basis Deposit Item

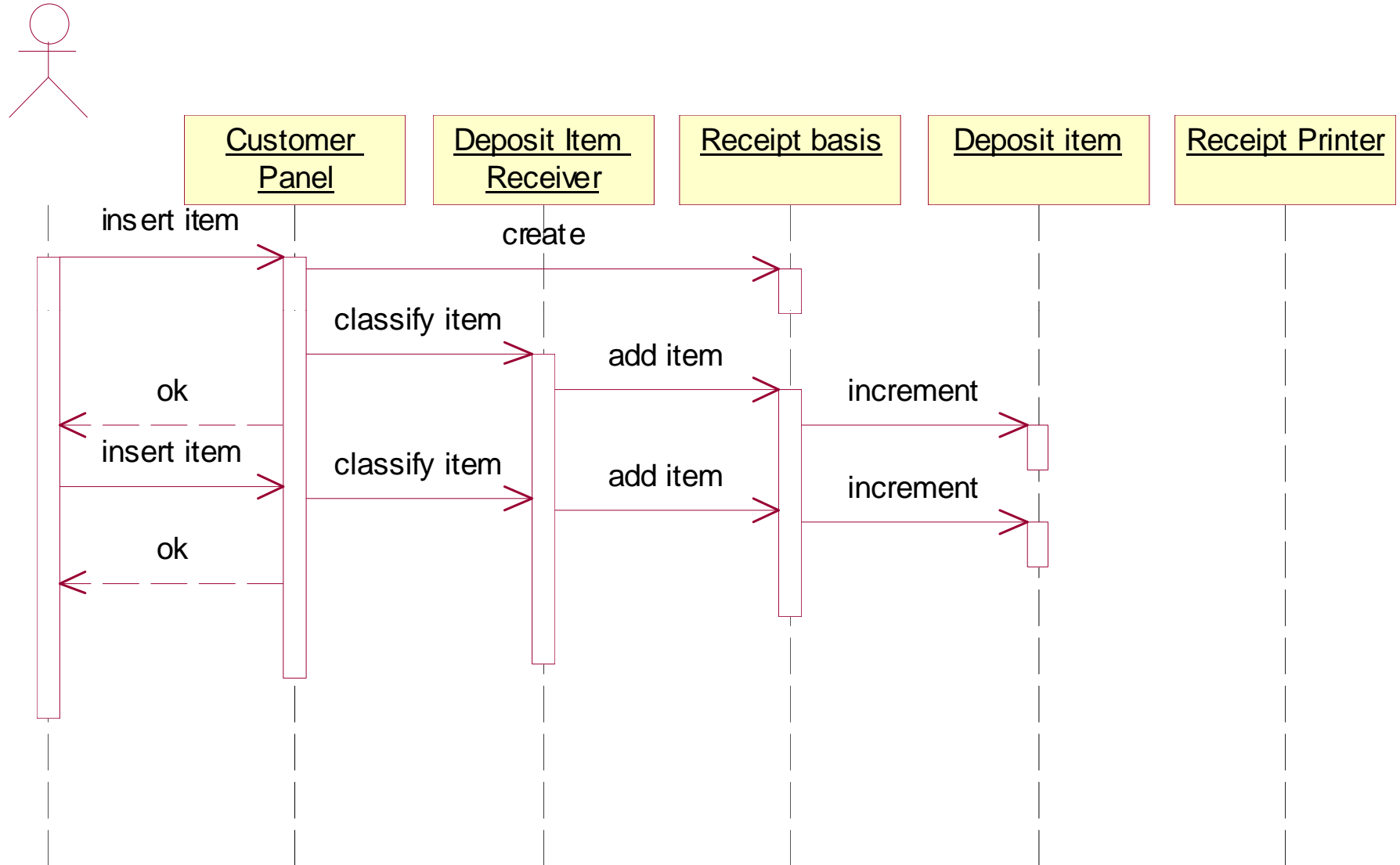


Example – Interaction Diagram for Use Case Returning Item

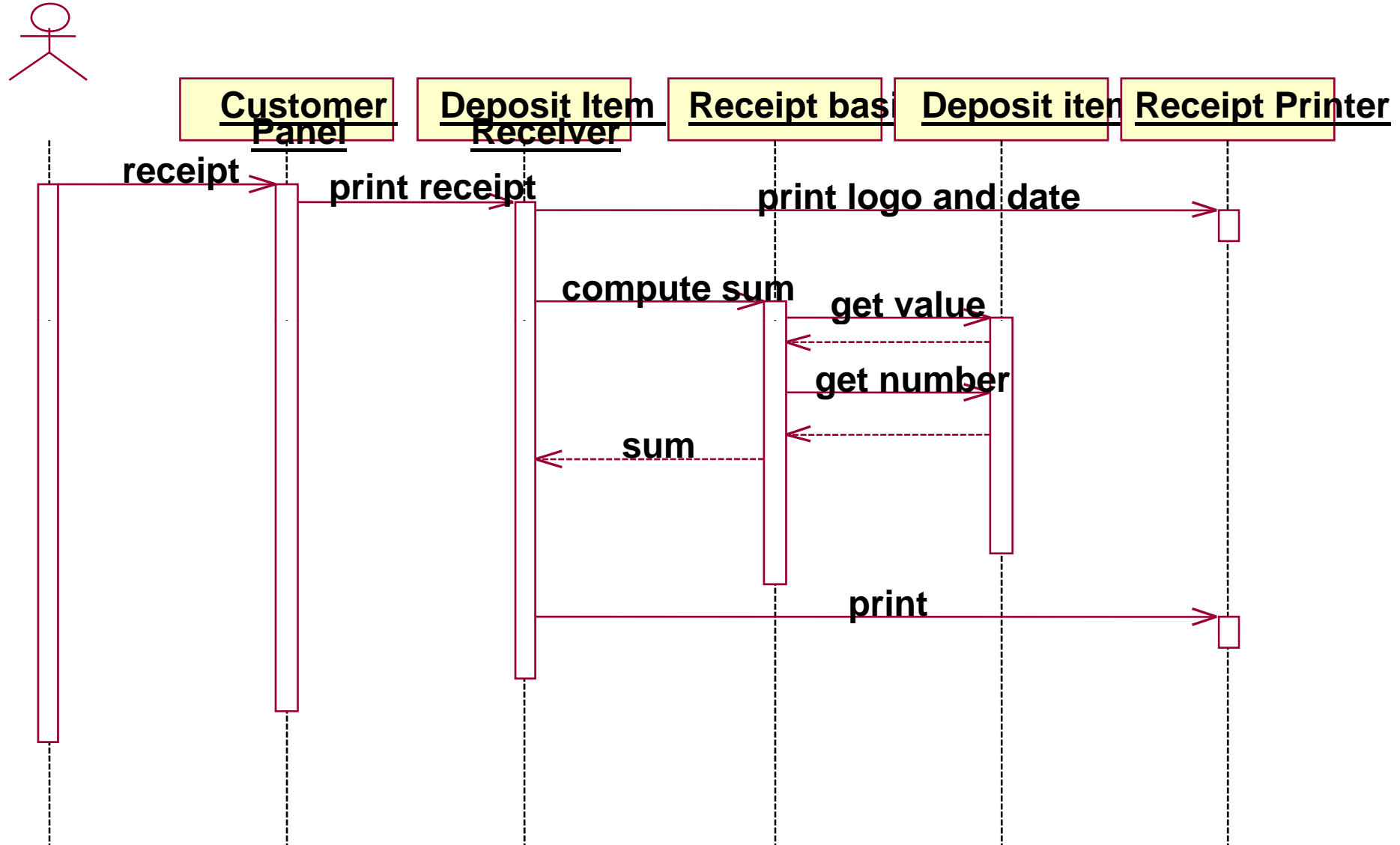
System Border Customer Panel Deposit Item Receiver Receipt Basis Deposit Item



Sequence Diagram



Sequence Diagram



Sequence Diagram

- **Events (when the customer presses receipt button):**

**Print Logo and date;
for (all types of items)**

{

Find name and number for a type of item;

Find deposit value for a type of item;

Sum;

Print sum;

}

Ready for the next customer;

Sequence Diagram

- **Events (when a customer insert an item):**

```
If (customer is new)
{
    Create a new account;
}
Do
{
    If (returned item is acceptable == true)
    {
        Classify;
        Increment items;
        Increment values;
    }
    else
        reject;
}
```

Sequence Diagram

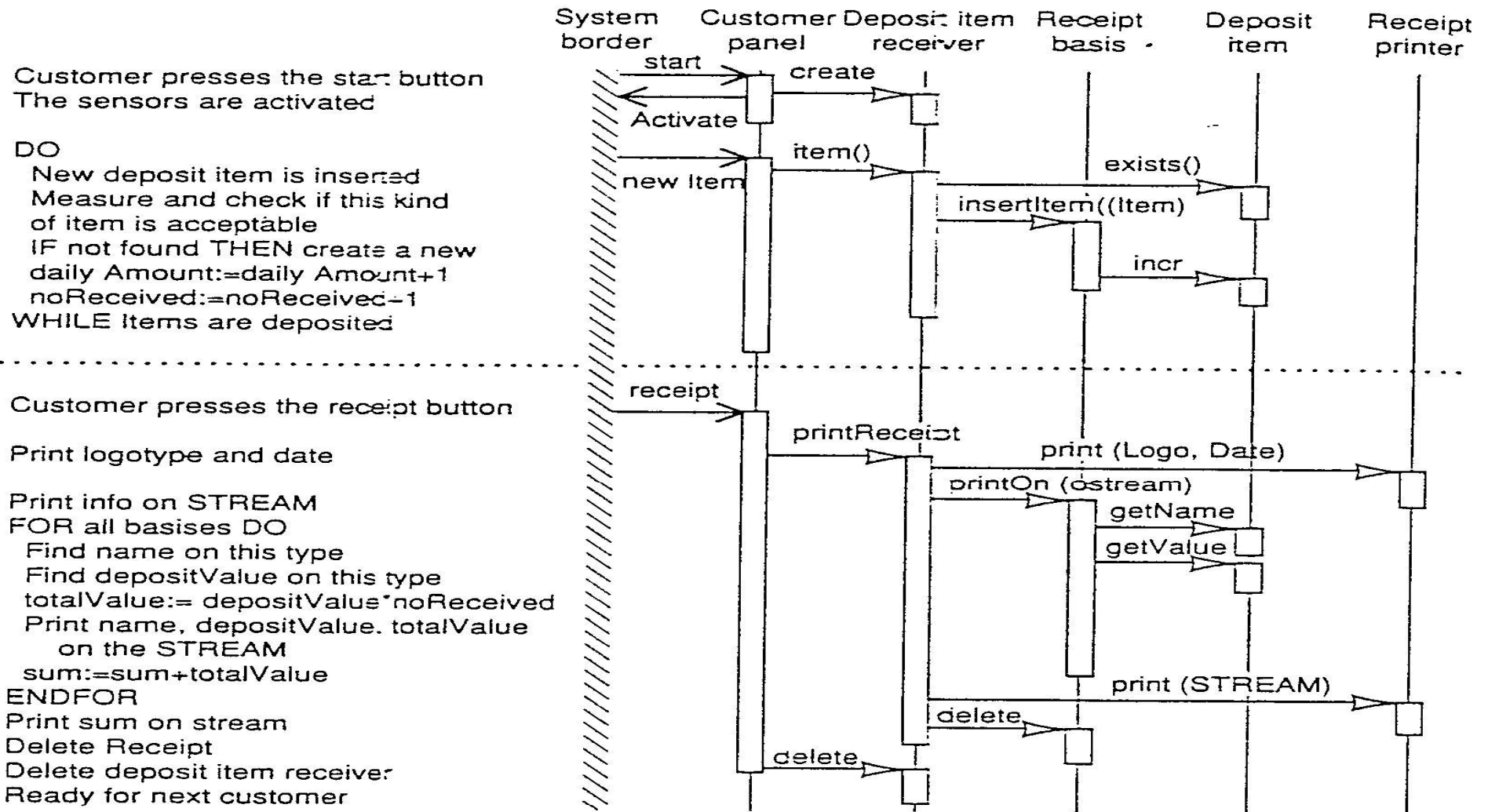
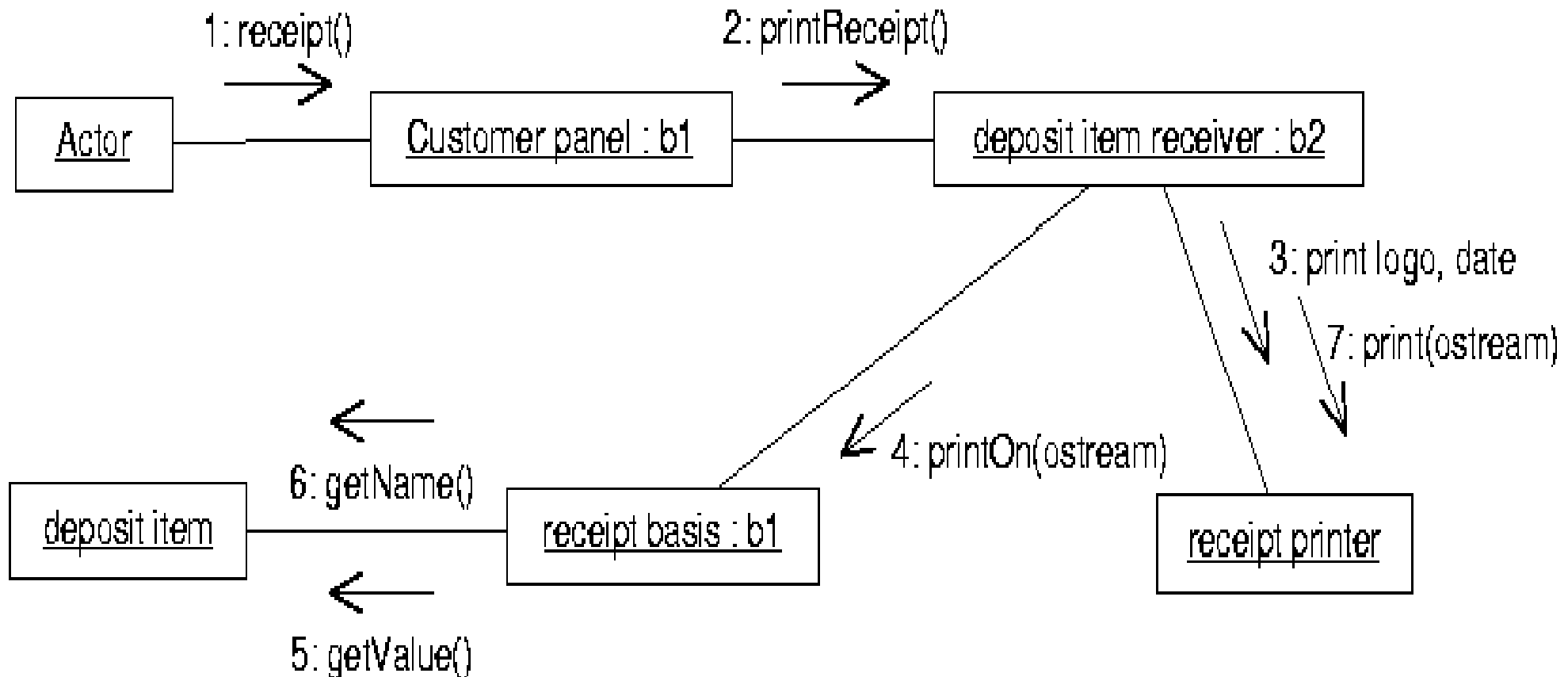


Figure 8.11 Interaction diagram for the use case *Returning Item*.

Rational Rose – Collaboration Diagram

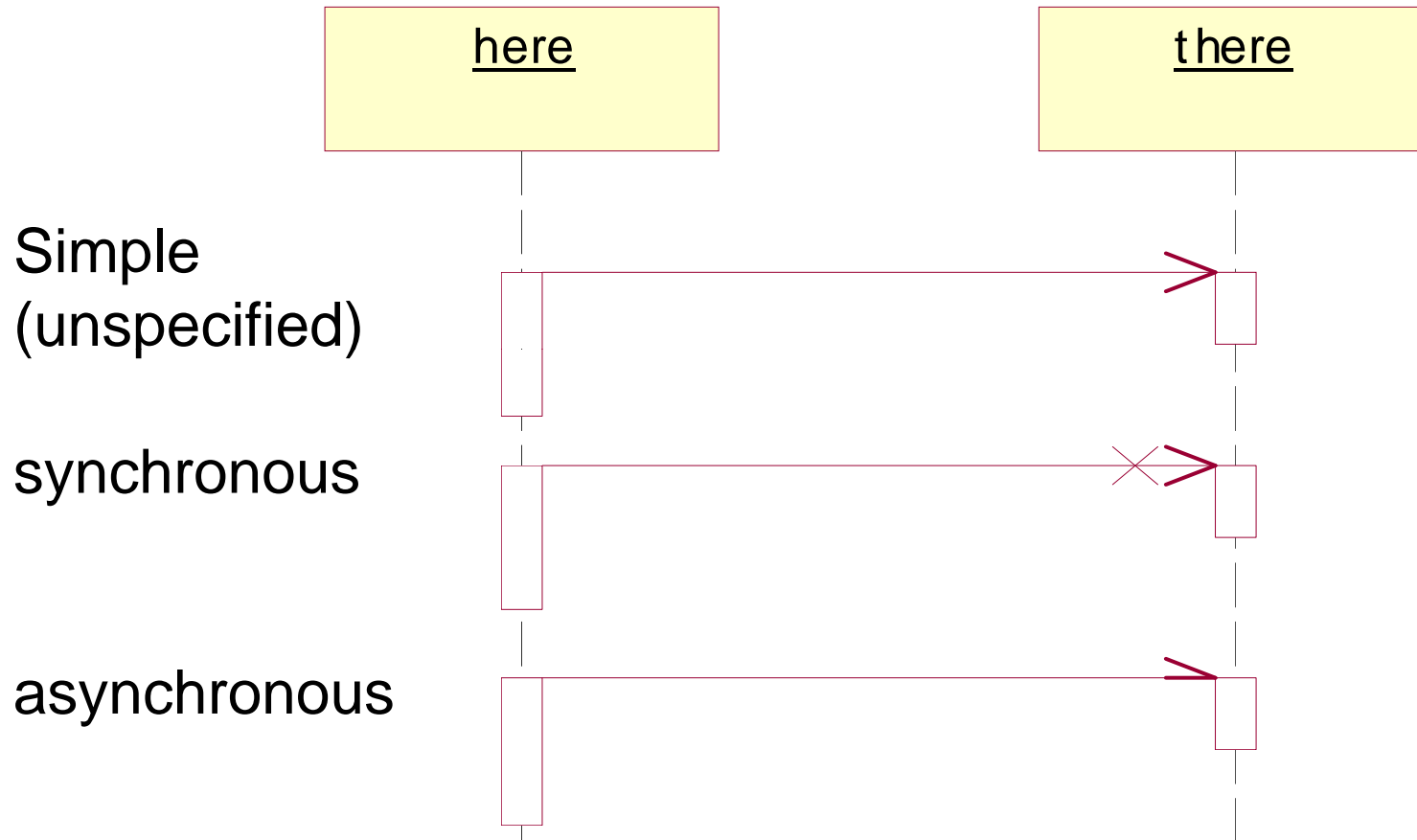


Advance Interaction Diagrams

Synchronous and asynchronous

- A **synchronous message/signal** is a control which has to wait for an answer before continuing.
 - the sender passes the control to the receiver and cannot do anything until the receiver sends the control back.
- An **asynchronous message** is a control which does not need to wait before continuing.
 - the sender actually does not pass the control to the receiver.
 - The sender and the receiver carry on their work concurrently.

Synchronous & Asynchronous





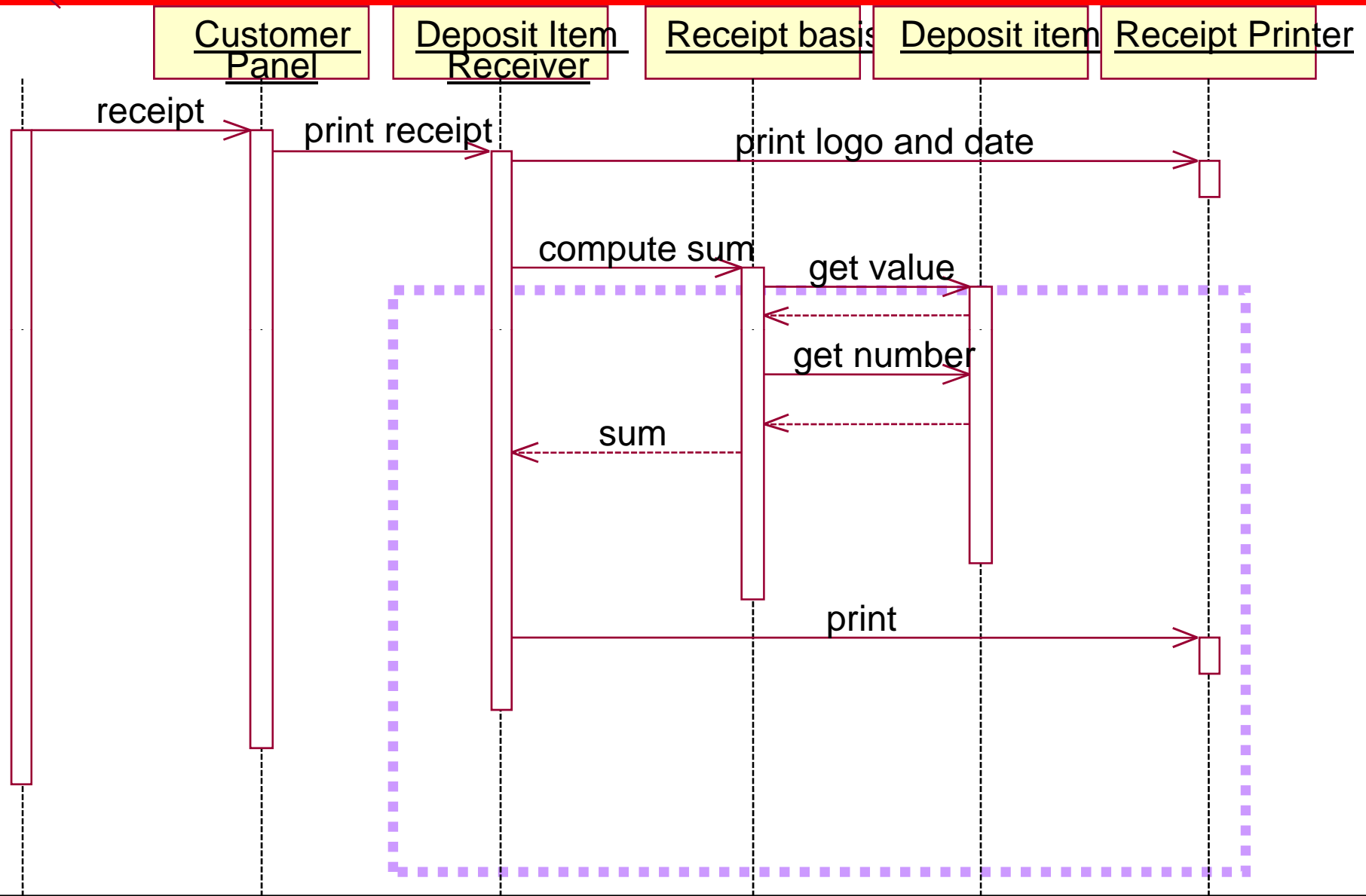
Synchronous & Asynchronous ..

- Example: the “return item” use case in the recycling machine.
- Concentrate on the rounded rectangle and answer the question:
- **“Should getName and getValue be synchronous or asynchronous?”**

Synchronous & Asynchronous..

- On one hand, getValue signal and getName signal can be asynchronous as the system does not need to wait for values before requesting name.
- On the other hand, the “print” message could not be sent out to the printer if getValue and getName were asynchronous.
- For this reason, getValue signal and getName signal must be **synchronous**.

Synchronous & Asynchronous..



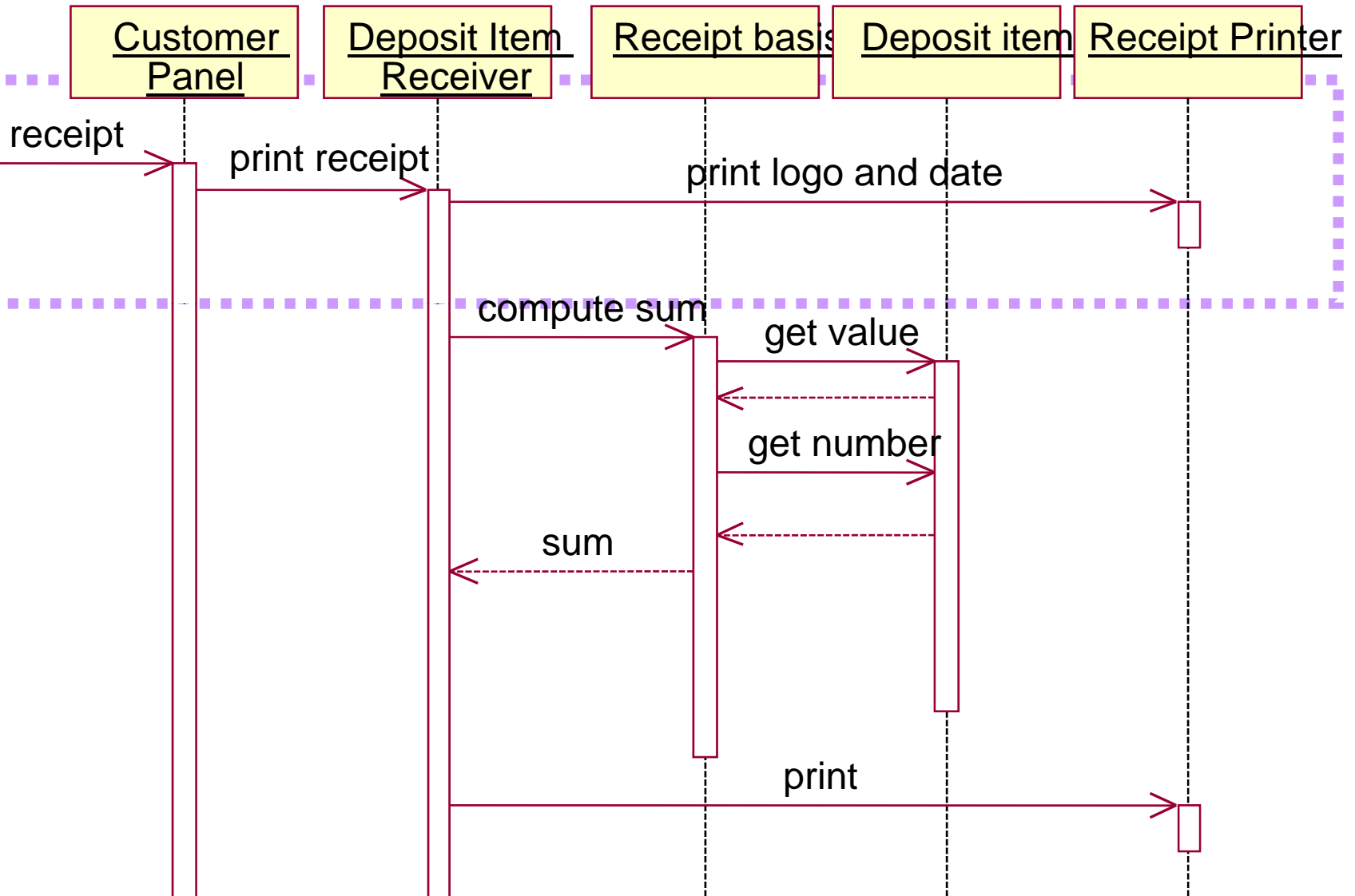
Synchronous & Asynchronous..

- **Concentrate on other signals and messages.**
 - **receipt signal:** it does not even need a feedback.

Therefore, it can **asynchronous**.

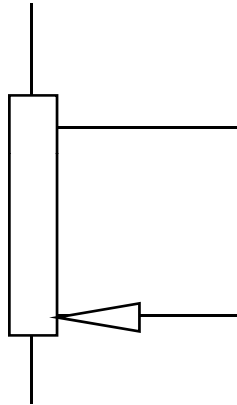
- **printReceipt signal:** it can be **asynchronous**, too.
- **printLogo, date message:** it can also be **asynchronous**, as no feedback is required.

Synchronous & Asynchronous..



Recursion in SD

- Syntax:



Use Case with Extension

- Described by a probe position in the interaction diagram
- The probe position indicates a position in the use case to be extended
 - Often accompanied by a condition which indicates under what circumstances the extension should take place

Example – Probe position

System Border Customer Panel Alarmist Alarm device

PROBE: In use case Returning Item when measuring the Deposit Item, the Item stuck

Start the alarm

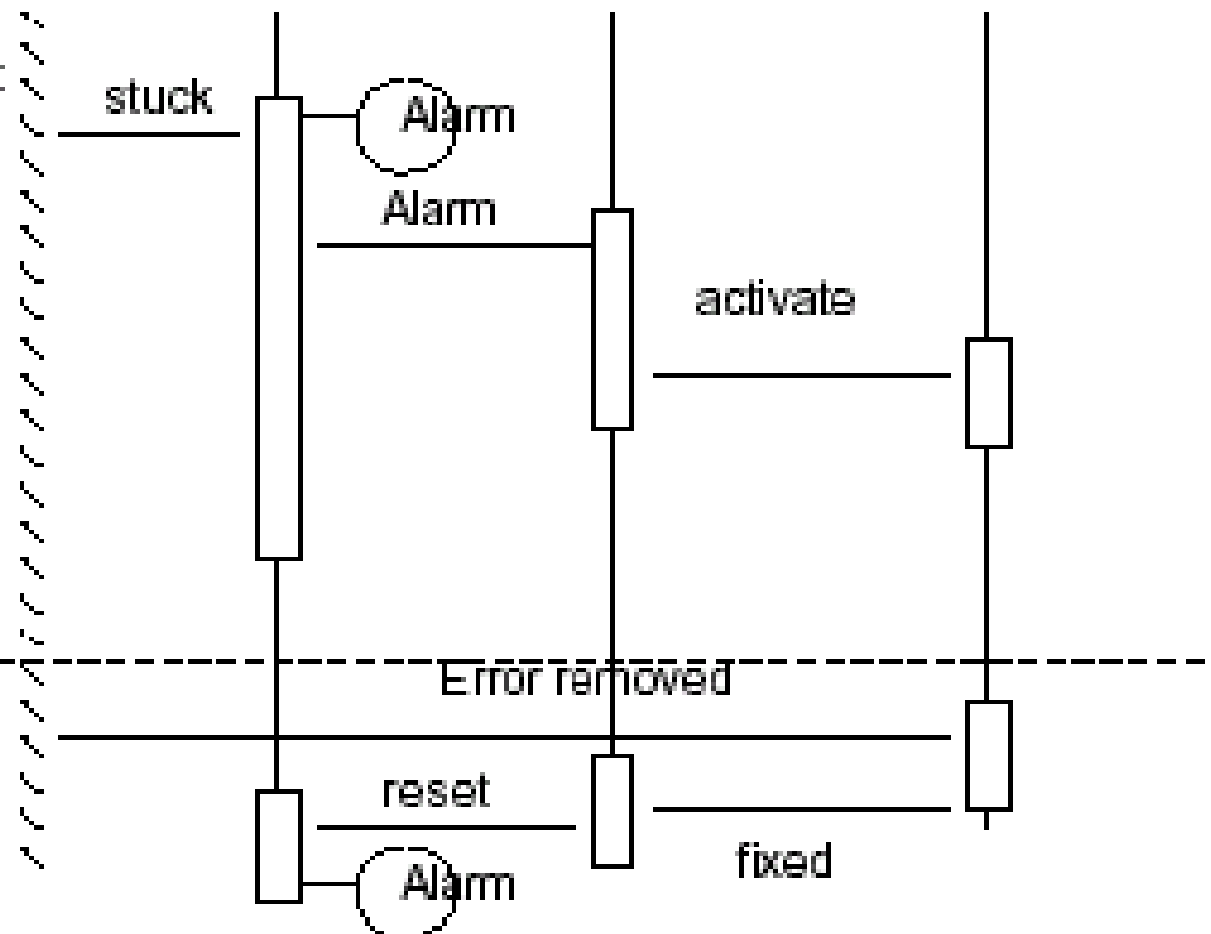
Activate Alarm device

Wait for error to be fixed

Error fixed

Turn off the alarm

Back to normal insertion



Homogenization

- In parallel design process, several stimuli with the same purpose or meaning are defined by several designers.
- These stimuli should be consolidated to obtain as few stimuli as possible.
 - Called *homogenization*.

Example - Homogenization

What_is_your_phone_number?

Where_do_you_live?

Get_address

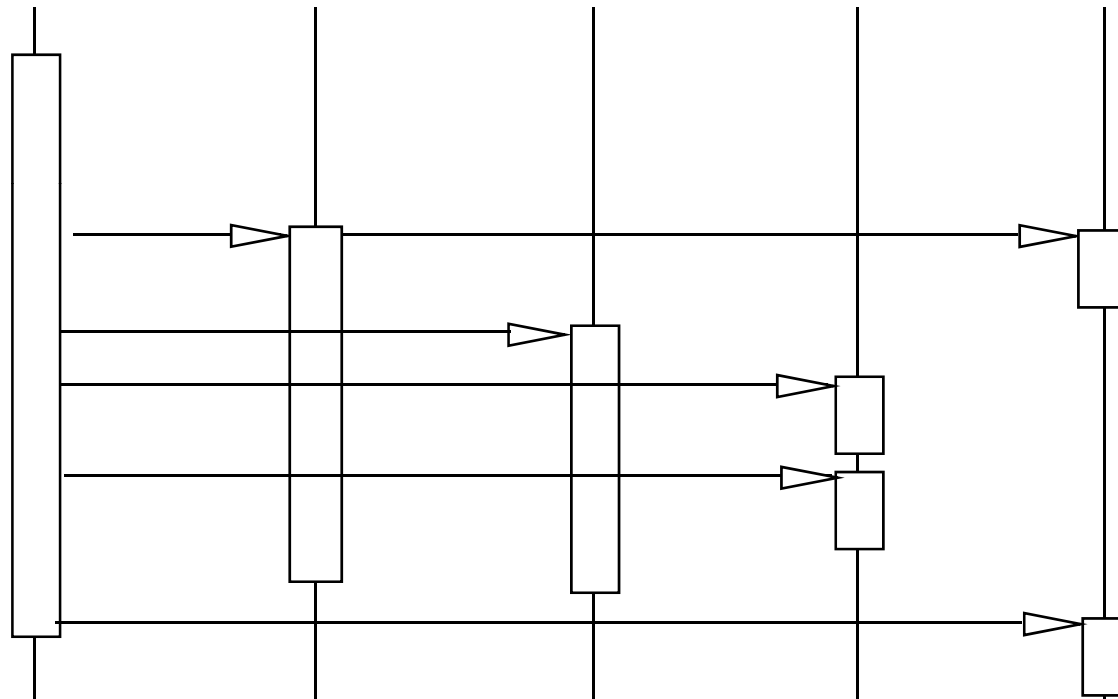
Get_address_and_phone_number

Homogenized into:

- Get_address
- Get_phone_number

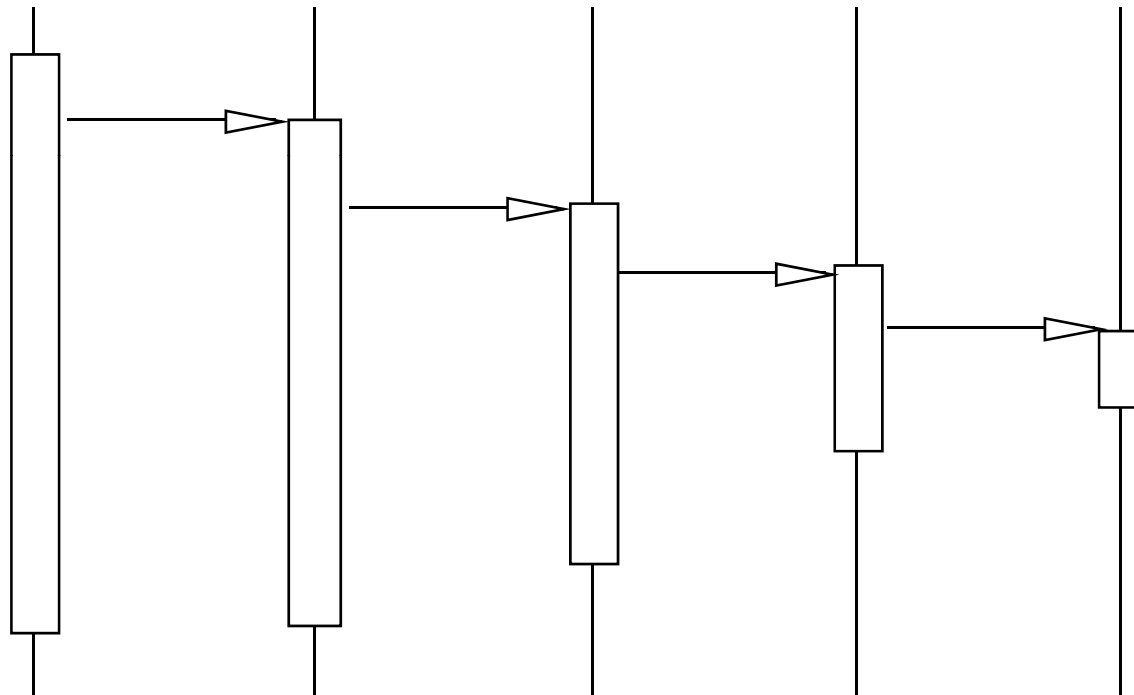
Structure of SD

- Centralised structure -- Fork: Everything is handled and controlled by the left-most block.



Structure of SD...

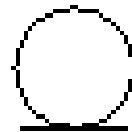
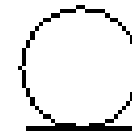
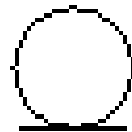
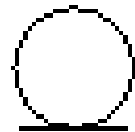
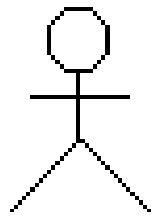
- Decentralised structure -- Stair: There is no central control block.



Structure of SD...

decentralized structure is appropriate:

- If the sub-event phases are tightly coupled. This will be the case if the participating objects:
 - Form a part-of or consists-of hierarchy, such as Country - State - City;
 - Form an information hierarchy, such as CEO - Division Manager - Section Manager;
 - Represent a fixed chronological progression (the sequence of sub-event phases will always be performed in the same order), such as Advertisement - Order - Invoice - Delivery - Payment; or
 - Form a conceptual inheritance hierarchy, such as Animal - Mammal - Cat.



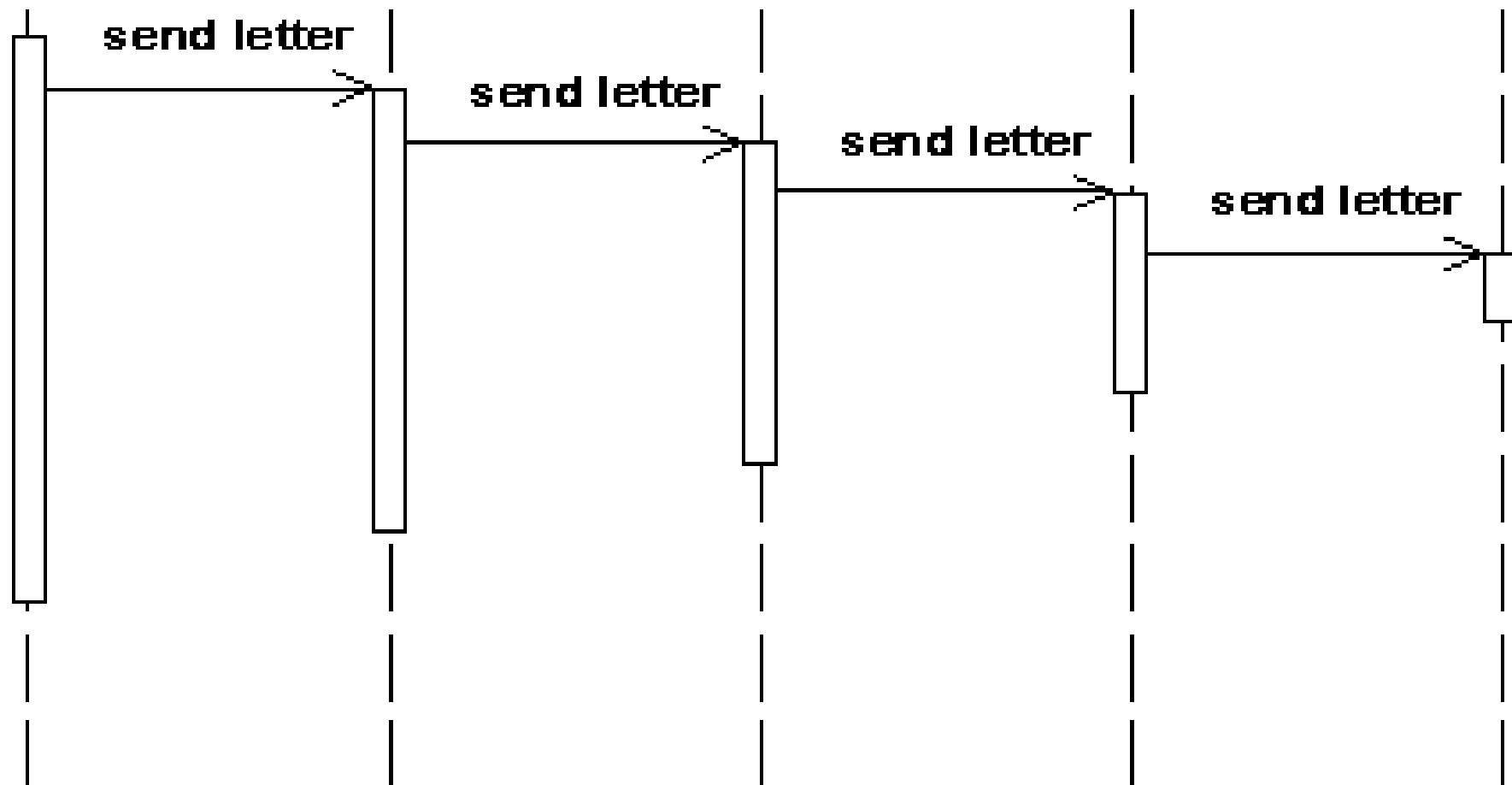
: Post Office
Customer

: Post Office

: Country

: City

: Addressee

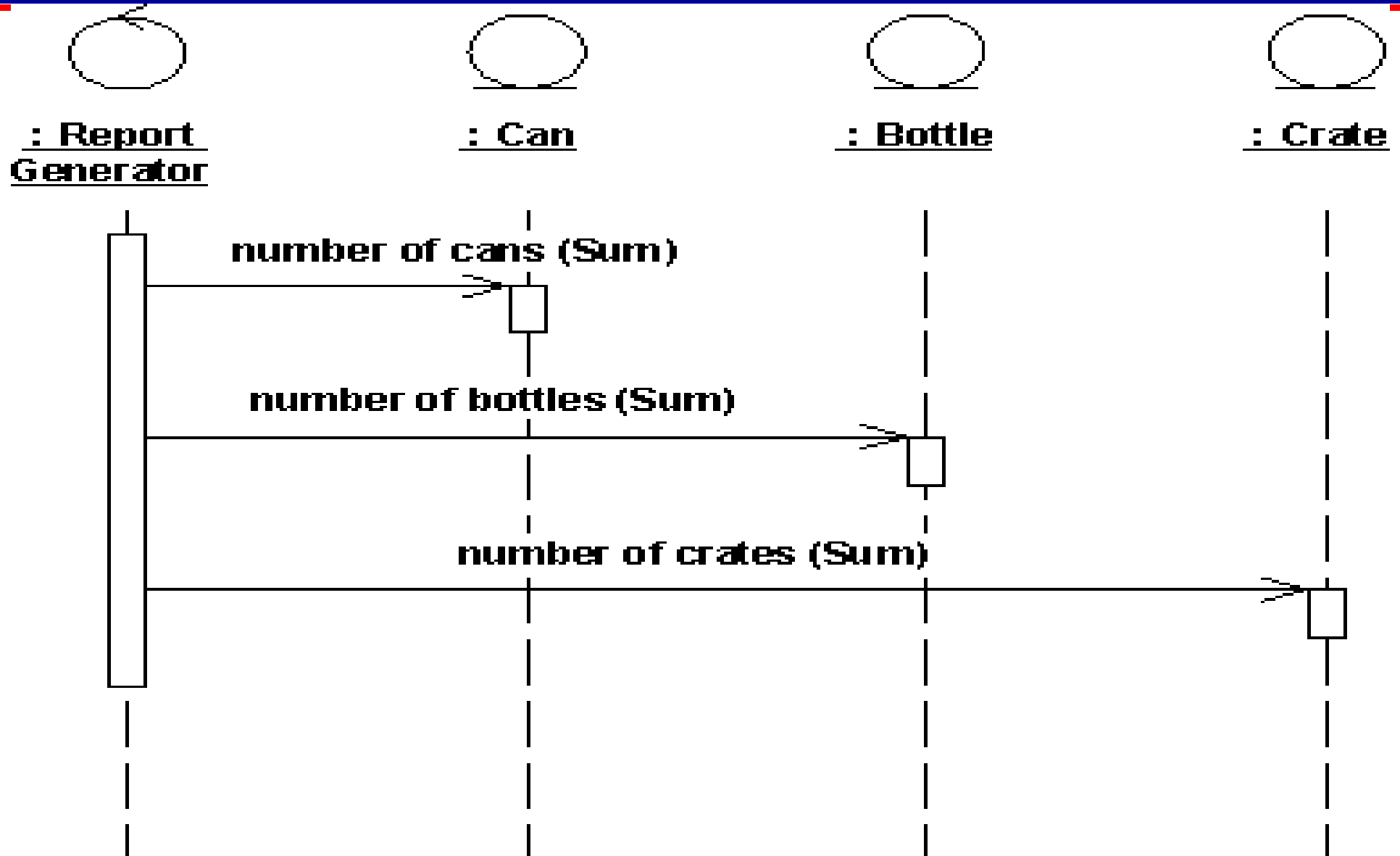


Structure of SD...

A centralized structure is appropriate:

- **If the order in which the sub-event phases will be performed is likely to change.**
- **If you expect to insert new sub-event phases.**
- **If you want to keep parts of the functionality reusable as separate piec**

Structure of SD...



Structure of SD..

Fork

- indicates a centralized structure and is characterized by the fact that it is an object controls the other objects interacted with it.
- This structure is appropriate when:
 - ✓ The operations can change order
 - ✓ New operations could be inserted

Structure of SD....

Stair

- indicates decentralized structure and is characterized by delegated responsibility.
- Each object only knows a few of the other objects and knows which objects can help with a specific behavior.
- This structure is appropriate when:
- The operation have a strong connection. Strong connection exists if the objects:
 - ✓ form a 'consist-of' hierarchy
 - ✓ form an information hierarchy
 - ✓ form a fixed temporal relationship
 - ✓ form a (conceptual) inheritance relationship
- The operation will always be performed in the same order



Structural Control in Sequence Diagram

- Optional Execution
- Conditional Execution
- Parallel Execution
- Loop Execution
- Nested

Block Design

- Block design can start when all use cases for a specific block have been designed.
- The implementation (code) for the block can start when the interfaces are stable and are frozen.
- Ancestor objects (classes) should be designed before descendant.
- Look at all interaction diagrams where a block participates, and extract all the operations defined for it, yielding a complete picture of the interface.

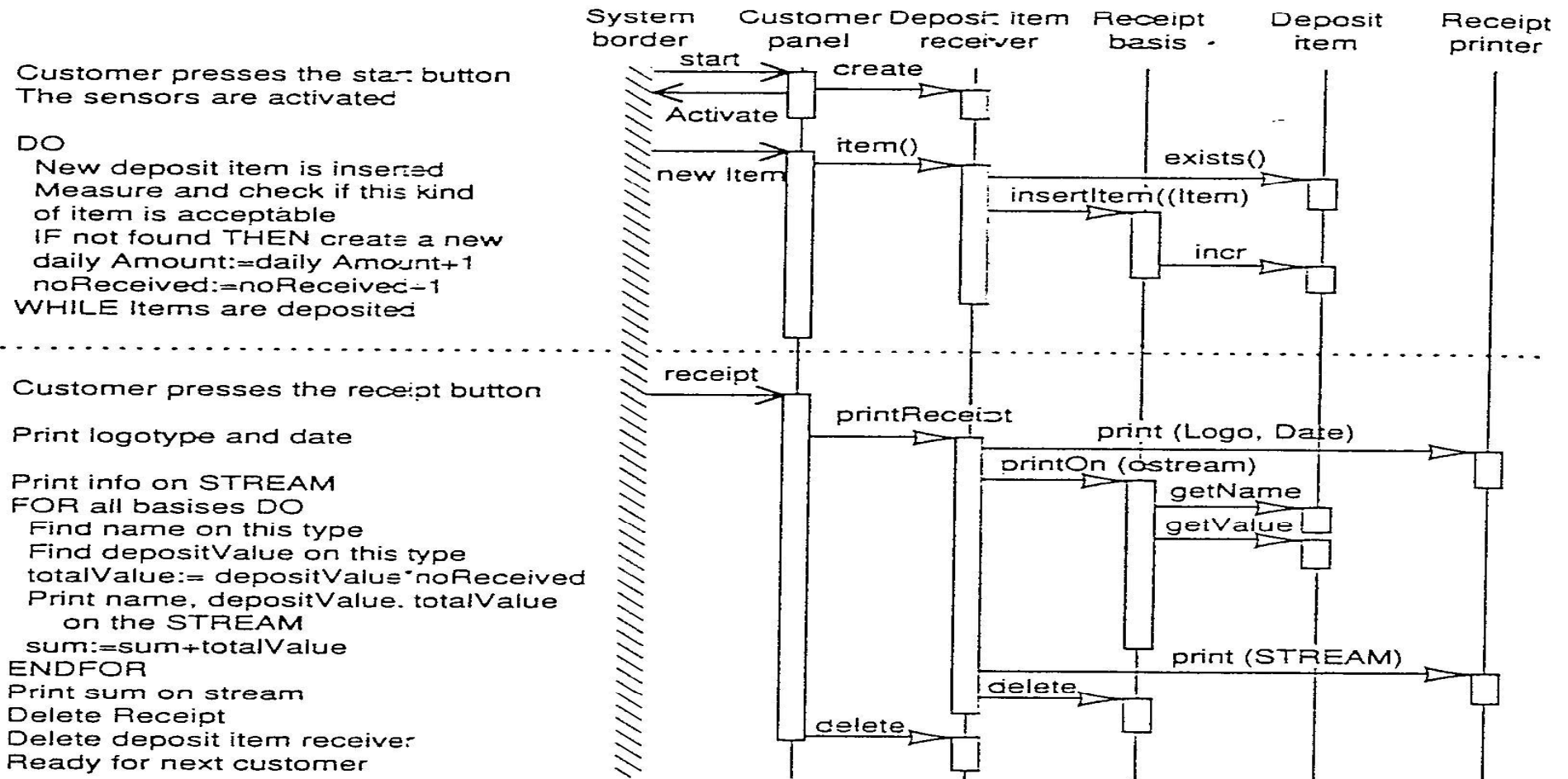


Figure 8.11 Interaction diagram for the use case *Returning Item*.

Block Design...

- From the interaction diagram for **Returning Item**:
- The interface for **Deposit Item**:
exists, incr, getName, getValue
- The interface for **Receipt Basis**:
insertItem(item), printOn(ostream), delete
- When writing the class interface for a block in OOP language, may need additional classes, that are not seen by other blocks

Block Design Comments

- The description of the operation is extracted from the text to the left of the diagram.
- Can work in parallel once interfaces are frozen (recall the open closed principle).



Object Behavior

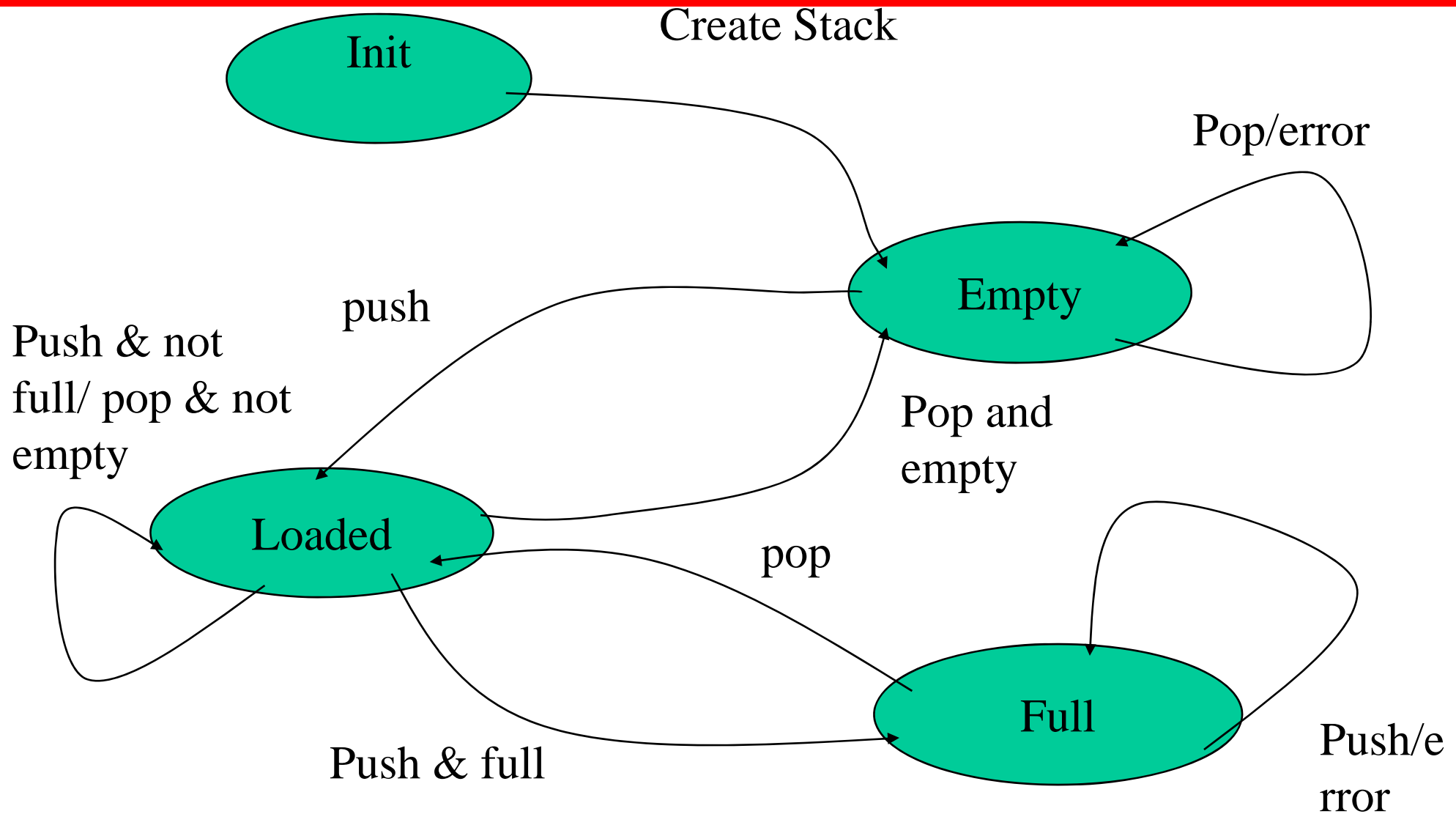
- An intermediate level of object internal behavior may be described using a state machine.
- The input alphabet is the set of stimuli.
- State represent modes of operation of the object.
- The state machine specifies possible orders of stimuli, that will later be expressed in preconditions.
- This is particularly important in reactive systems



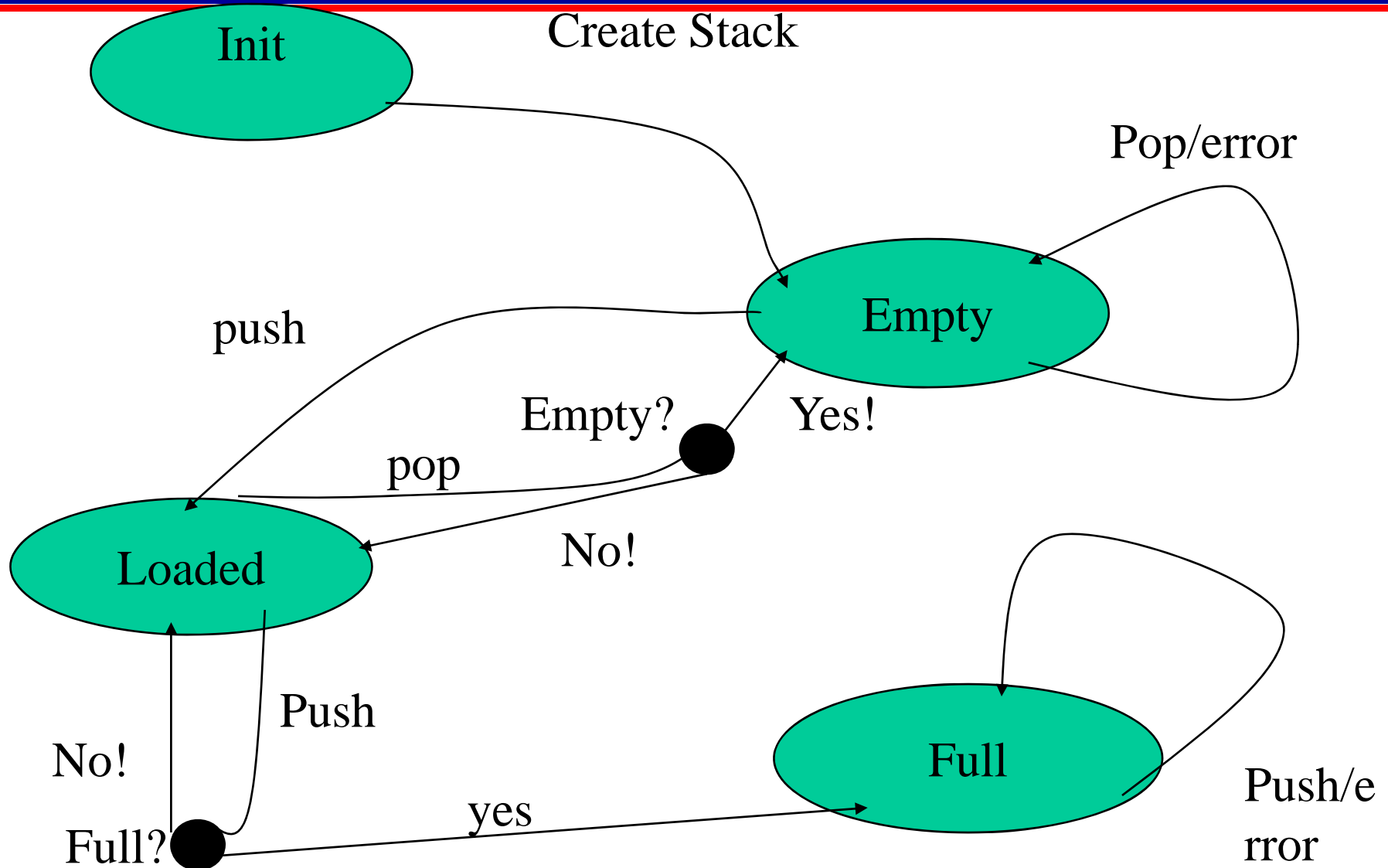
Object Behavior...

- To provide a simplified description that increases understanding of the block without having to go down to source code
- Less dependant on programming language

Object Behavior...



Object Behavior...



Textual Notation

Machine stack

State init

input createinstance

next state empty

otherwise error;

State empty

input push

do store on top

next state loaded

print "stack is empty"

otherwise error;

..

endmachine



Object Behavior...

Stimulus control object

- An object that perform the same operation independent of state when a certain stimulus is received.
- Entity objects














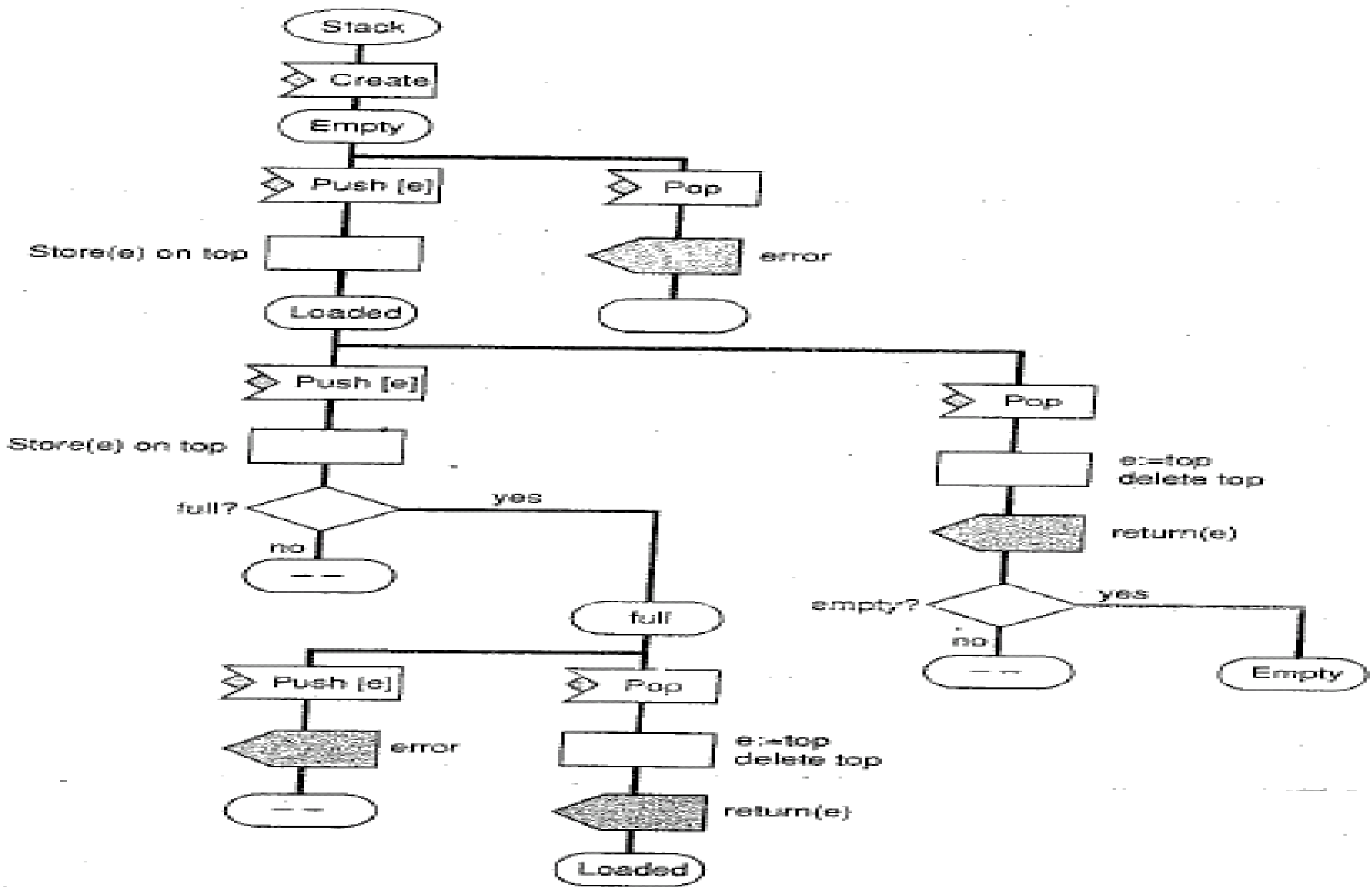
Object Behavior...

state-controlled objects

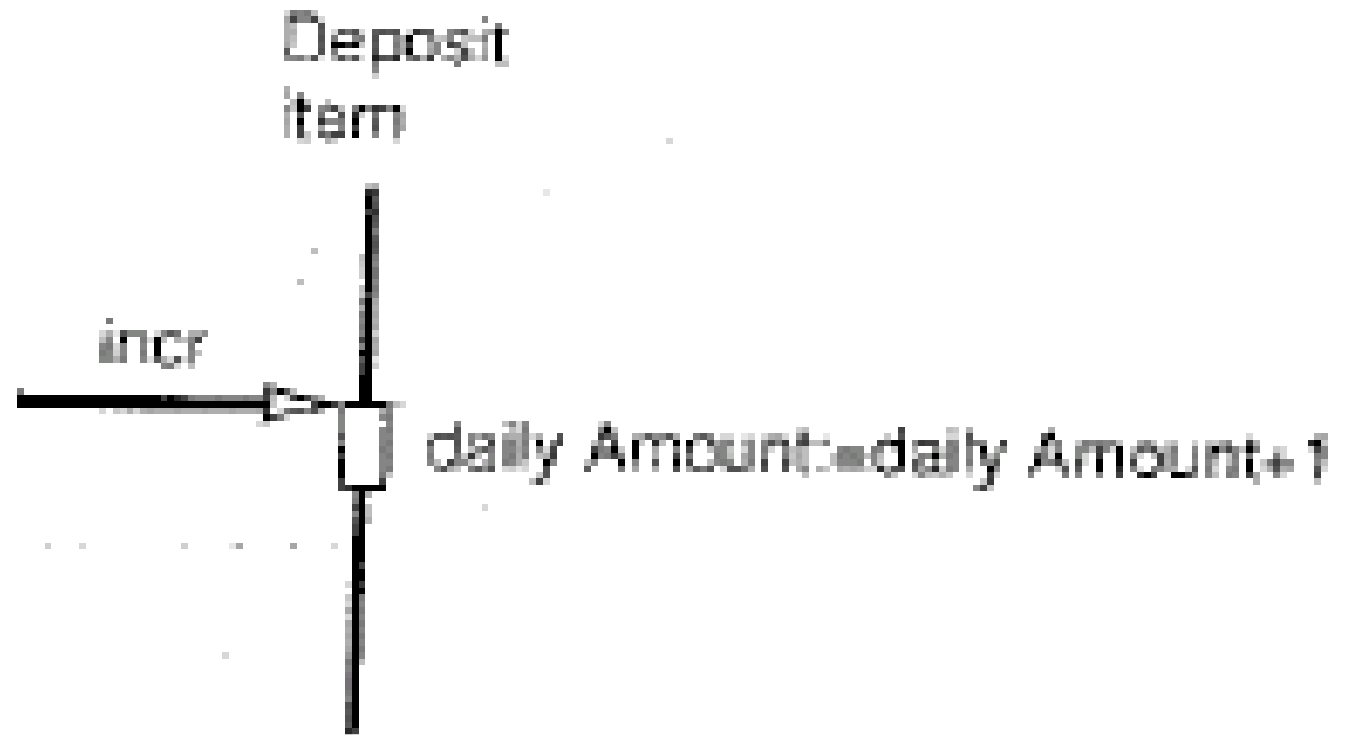
- Objects that select operations not only from the stimulus received, but also from the current state
- Control object

Notation used for State Transition Diagram

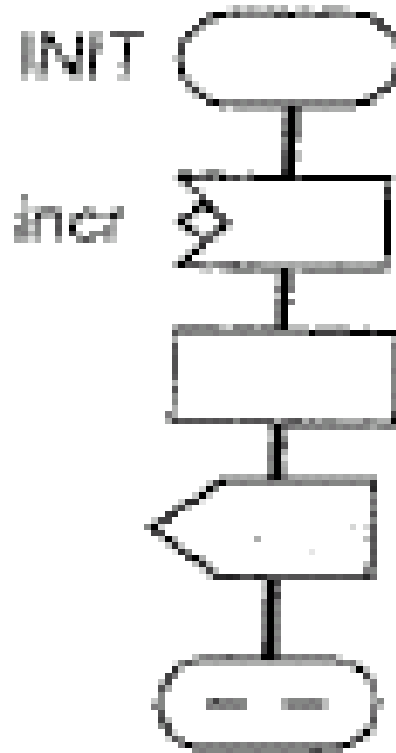
	Start symbol
	State
	Send message
	Receive message
	Return from message
	Send signal
	Receive signal
	Perform a task
	Decision
	Destroy object
	Label



Operation

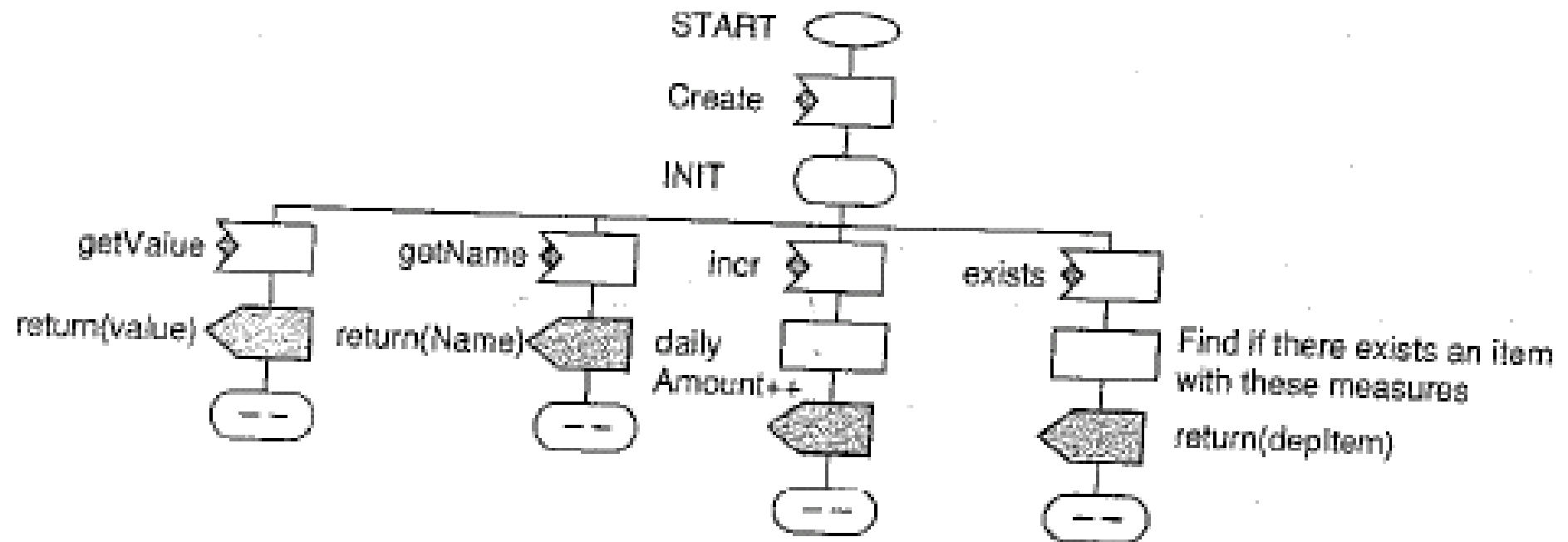


STD

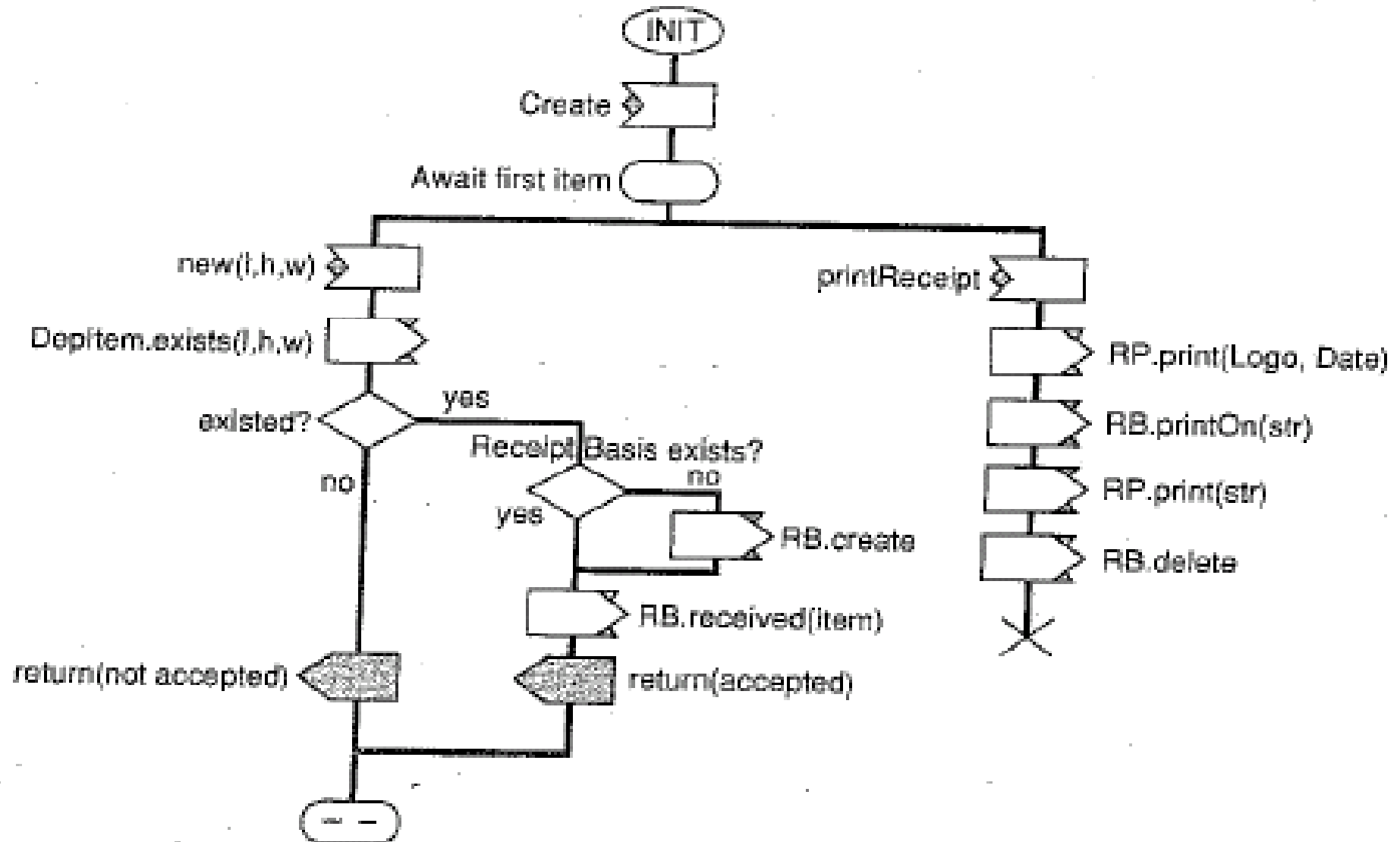


daily Amount= daily Amount+1

STD for Deposit Item



STD for Deposit Item Receiver



Internal Block Structure

- In case of OOPL object-module becomes classes otherwise module unit
- Generally more classes than object
- split class when required
- 5-10 times longer to design a component class than an ordinary class

Implementation

- Now, need to write code for each block.
- Implementation strategy depends on the programming language.
- In an **OOP** language, the implementation of a block starts with one class.
- Sometimes there is a need for additional classes, that are not seen by other blocks.

Mapping

Analysis	Design	Source code C++
Analysis objects	Block	1..N classes
Behavior in objects	Operations	Member functions
Attributes(class)	Attributes(class)	Static variables
Attributes(instance)	Attributes(instance)	Instance variables

Mapping..

Analysis	Design	Source code C++
Acquuaintance ass.	Acquuaintance ass.	Instance variables
Communication ass.	Communication ass.	Reference to a function
Interaction between objects	Stimulus	Call to a functionom
Usecase	Designed usecase	Sequence of calls
subsystem	Subsystem	File

Implementation Environment

Everything that does not come from analysis phase, including performance requirements.

- Design must be adapted to implementation environment.
- Use of existing products must be decided. Includes previous version of the system.
- To use an existing product we must adapt our design.
- Tradeoff - less development vs. more complex architecture.
- Also consider testing costs.



Other Considerations in the Construction Phase

- Subsystems defined in analysis phase are used to guide the construction phase.
- Developed separately as much as possible.
- Incremental development - start construction phase in parallel with analysis phase - to identify implementation environment.
- How much refinement to do in analysis phase? (How early/late to move from analysis to design) - decided in each project

What we learnt

- ✓ What is Construction Phase
- ✓ Why Construction
- ✓ Add a Dimension
- ✓ Artifacts for Construction
- ✓ Design (What, Purpose, Goals, Levels)
- ✓ Implementation Environment
- ✓ Traceability
- ✓ Interaction Diagram
- ✓ Block design
- ✓ Block Behavior
- ✓ Implementation

Class Diagram

Categorize the following relationships

1. A country has a capital city.
2. A dining philosopher uses a fork.
3. A file is an ordinary file or a directory file.
4. A polygon is composed of an ordered set of points.

Class Diagram

Prepare a class diagram for a graphical document editor that supports grouping. Assume that a document consists of several sheets. Each sheet contains drawing objects, including text, geometrical objects and groups. A group is simply a set of drawing objects, possibly including other groups. A group is simply a set of drawing objects, possibly including other groups. A group must contain at least two drawing objects. A drawing object can be a direct member of at most one group. Geometrical objects include circles, ellipses, rectangles, lines, and squares.

State Diagram Problem

A simple digital watch has a display and two buttons to set it, the A button and the B button. The watch has two modes of operation, display time and set time. In the display time mode, the watch displays hours and minutes, separated by a flashing colon.

The set time mode has two submodes, set hours and set minutes. The A button selects modes. Each time it is pressed, the mode advances in the sequence: display, set hours, set minutes, display, etc. Within the submodes, the B button advances the hours or minutes once each time it is pressed. Buttons must be released before they can generate another event. Prepare a state diagram of the watch.

Testing

**process of executing a program with the
intent of finding errors**



Software Testing

- Consumes at least half of the labor
- Process of testing software product
- Contribute to
 - Delivery of higher quality product
 - More satisfied users
 - Lower maintenance cost
 - More accurate and reliable results

TERMINOLOGY

- Error
- Fault/Defect
- Failure
- Incident
- Test
- Test Case
- Testing
- Verification
- Validation
- Debugging
- Certification
- Clean Room Software Engineering

Error

- People make errors.
- Typographical error
- Misreading of a specification
- Misunderstanding of functionality of a module
- A good Synonym is Mistake.
- When people make mistakes while coding we call these mistakes “**bugs**”

Fault/Defect

Representation of an error

- DFD
- Hierarchy chart
- Source Code
- An error may lead to one or more faults

Fault of Omissions

- If certain specifications have not been programmed

Fault of Commission

- If certain program behavior have not been specified



Fault/Defect..

- Defects generally fall into one of the following three categories
 - Wrong
 - Missing
 - Extra



Failure, Incident

Failure

- A particular fault may cause different failures, depending on how it has been exercised

Incident

- When a failure occurs, it may or may not readily appear to the user
- Incident is the symptom associated with a failure that alerts the user to the occurrence of a failure



Test, Test Case

Test:

- A test is the act of exercising S/W with test cases

Test Case:

- A test case has an identity and is associated with a program behaviour
- It has a set of inputs and a list of expected outputs



S/W Testing

- Testing is the process of demonstrating that errors are not present
- The purpose of testing is to show that a program performs its intended functions correctly
- Testing is the process of establishing confidence that a program does what it is supposed to do
- **Testing is the process of executing a program with the intent of finding errors**



Verification

- Verification: *Are we building the product right?*
- Static and proactive in nature (QA).
- Checks if product (result of a particular phase of SDLC) conforms with its specifications.
- Is static in nature. Involves manual checking, doesn't include execution of code.
- The target documents are Req. Specs., HLD, DB Design
- The methods used are Inspection, Walkthrough etc.



Validation

- Validation: *Are we building the right product?*
- Dynamic and reactive in nature (QC).
- Checks the software to insure it meets customer's requirements.
- Is dynamic in nature. Involves execution of code.
- The target could be a component, module, a set of integrated modules or system as a whole.
- The methods used are Black box, white box testing etc.

Debugging & Certification

- a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected
- certification is intended to establish standards for initial qualification and provide direction for the testing function



Clean Room Software Engineering

- a software development process intended to produce software with a certifiable level of reliability
- originally developed by Harlan Mills and several of his colleagues including Alan Hevner at IBM
- Focus is on defect prevention, rather than defect removal.



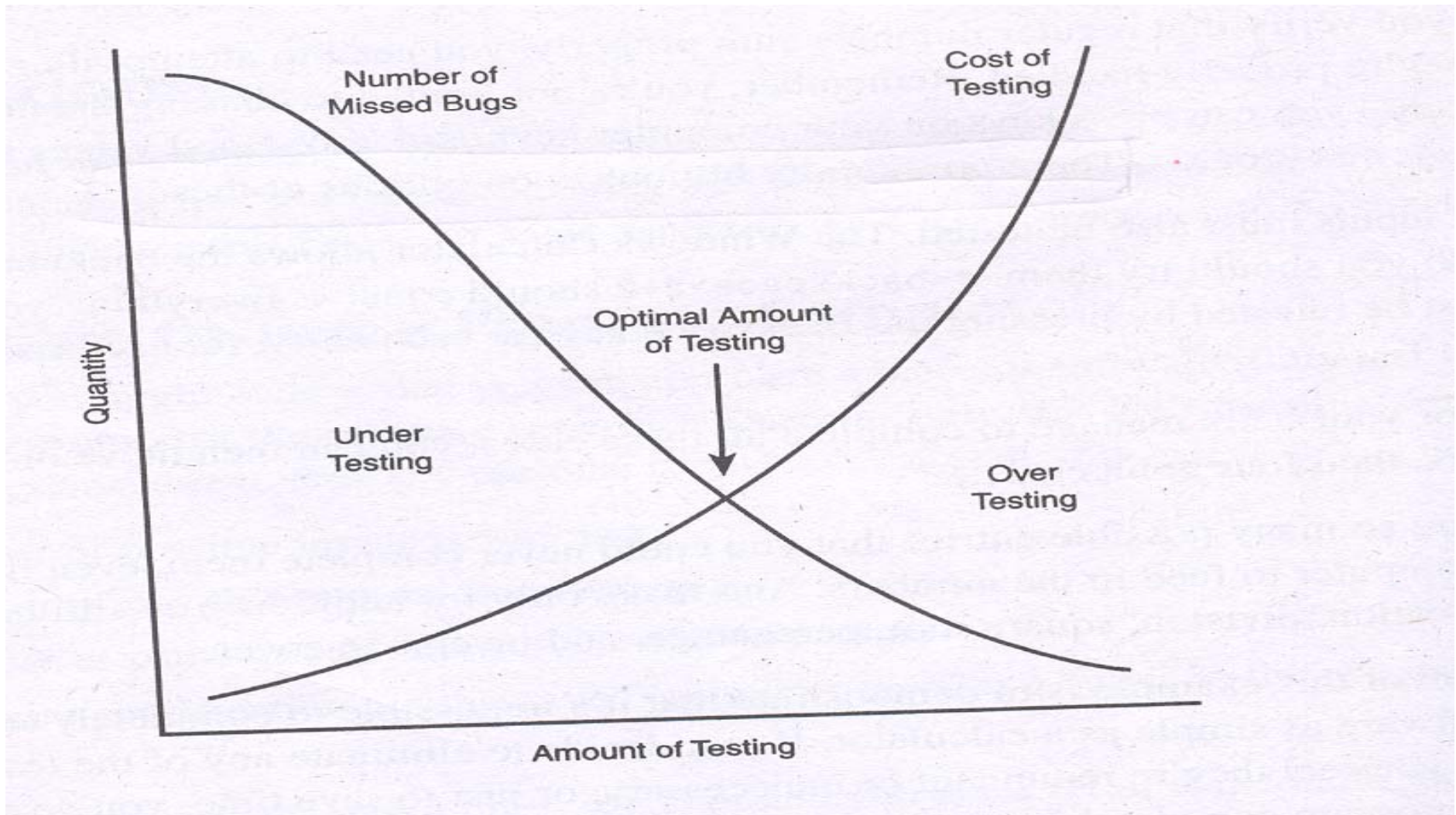
Testing Techniques

- Regression Test
- Operation test
- Full-scale test
- Stress test
- Overload test
- Negative test
- Test based on requirements
- Ergonomic tests
- Testing of the user documentation
- Acceptance testing

Object-Oriented Testing

- *When should testing begin?*
- Analysis and Design:
 - Testing begins by evaluating the OOA and OOD models
 - *How do we test OOA models (requirements and use cases)?*
 - *How do we test OOD models (class and sequence diagrams)?*
 - Structured walk-throughs, prototypes
 - Formal reviews of correctness, completeness and consistency
- Programming:
 - *How does OO make testing different from procedural programming?*
 - Concept of a 'unit' broadens due to class encapsulation
 - Integration focuses on *classes* and their context of a use case scenario
or their execution across a thread
 - Validation may still use conventional black box methods

Testing Cost Curve



Testing Strategies

- Construction

- System
- Subsystem
- Usecase
- Service Package
- Block
- Class

- Tesing

- System
- Subsystem
- Usecase
- Service Package
- Block
- Class



Testing Level

- Unit testing
- Integration Testing
- System Testing

Equivalent Partitioning

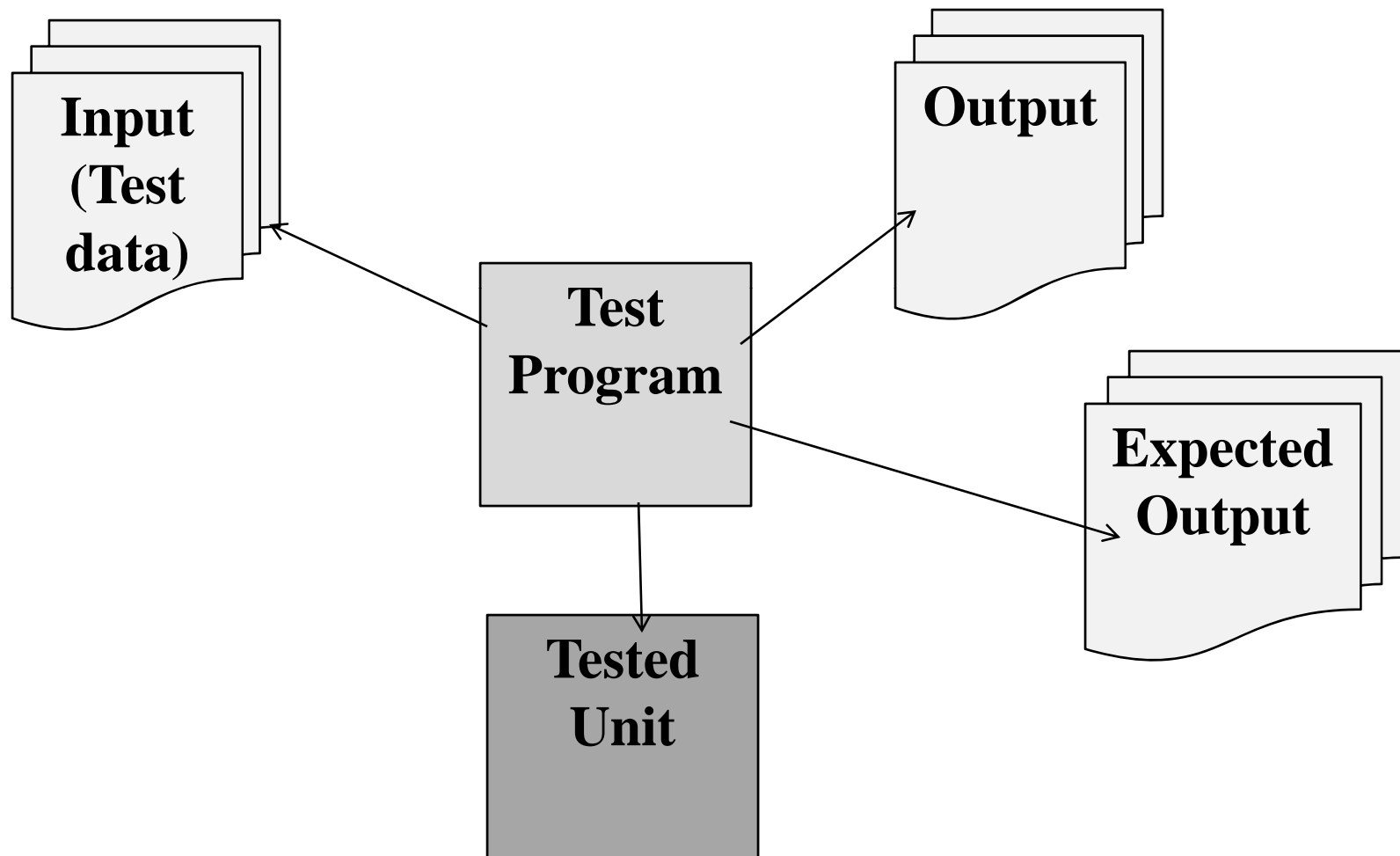
- Important aspect
- form a partition of a set, where partition refers to a collection of mutually disjoint subsets where the union is the entire set
- If the same result is expected from two test cases, consider them equivalent



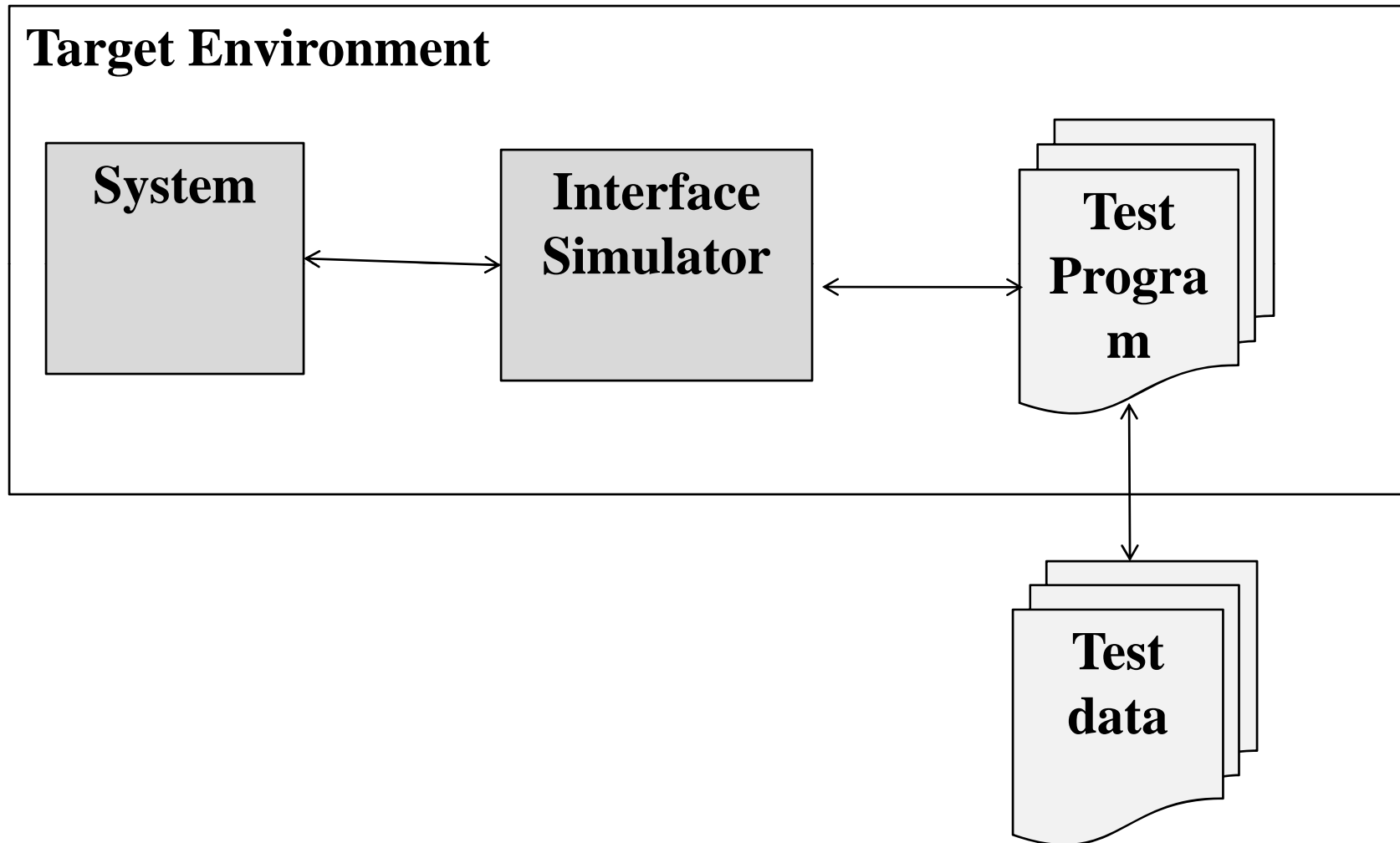
Automatic Testing

- Test need to be store
- Specially in case of Regression Test
- need to automate as much as possible

Principle of Automatic Testing



Use of an Interface Simulator





Unit Testing

- Lowest Level of testing
- Generally done by developer
- More Complex in case of OO Code
- Issues like
 - Inheritance
 - Polymorphism

Class (Unit) Testing

- Smallest testable unit is the encapsulated class
- Test each operation as part of a class hierarchy because its class hierarchy defines its context of use
- Approach:
 - Test each method (and constructor) within a class
 - Test the state behavior (attributes) of the class between methods
- *How is class testing different from conventional testing?*
- Conventional testing focuses on input-process-output, whereas class testing focuses on each method, then designing sequences of methods to exercise states of a class
- But white-box testing can still be applied

Unit Testing

- Consist of
 - Structural testing
 - Specification Testing
 - State Based Testing
- Suggested to perform Structural testing in the last
- Start with specification testing

Specification testing

- Verify specified behavior
- Test data based on the parameter of operations
- Techniques used
 - Boundary Testing
 - Equivalence Testing

State Based Testing

- Test the interactions between the operations of a class by monitoring the states
- Every state must be visited at least once and every transition must be traversed at least once
- Use tool state matrix
 - Represent the combination of states and stimuli



Structural Testing

- Test the internal structure
- Use measure of test coverage
- Least coverage is decision coverage

Challenges of Class Testing

- Encapsulation:
 - Difficult to obtain a snapshot of a class without building extra methods which display the classes' state
- Inheritance and polymorphism:
 - Each new context of use (subclass) requires re-testing because a method may be implemented differently (polymorphism).
 - Other unaltered methods within the subclass may use the redefined method and need to be tested
- White box tests:
 - Basis path, condition, data flow and loop tests can all apply to individual methods, but don't test interactions between methods

Random Class Testing

1. Identify methods applicable to a class
 2. Define constraints on their use – e.g. the class must always be initialized first
 3. Identify a minimum test sequence – an operation sequence that defines the minimum life history of the class
 4. Generate a variety of random (but valid) test sequences – this exercises more complex class instance life histories
- Example:
 1. An account class in a banking application has *open*, *setup*, *deposit*, *withdraw*, *balance*, *summarize* and *close* methods
 2. The account must be opened first and closed on completion
 3. *Open – setup – deposit – withdraw – close*
 4. *Open – setup – deposit –* [deposit | withdraw | balance | summarize] – withdraw – close*. Generate random test sequences using this template



Object Oriented Testing Concepts..

- Stubs
- Drivers
- Test bed
- Equivalence set
- Equivalence partitioning
- Automatic Testing
 - Test data
- Test program

Integration Testing

- OO does not have a hierarchical control structure so conventional top-down and bottom-up integration tests have little meaning
- Integration applied three different incremental strategies:
 - Thread-based testing: integrates classes required to respond to one input or event
 - **Use-based testing: integrates classes required by one use case**
 - Cluster testing: integrates classes required to demonstrate one collaboration
- Done by separate test team in target environment
- More formal documentation

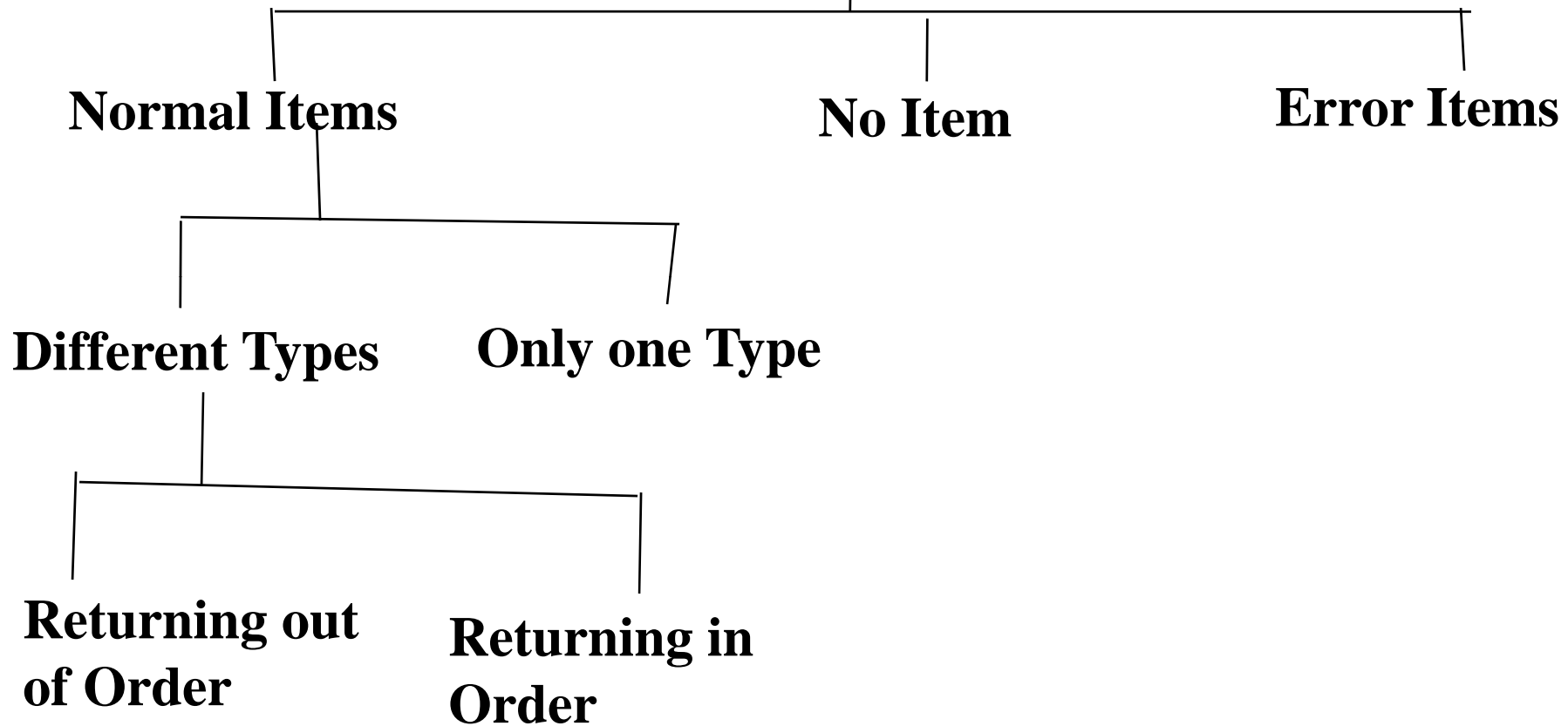


Use Case Testing

- Different test types
 - Basic Course Tests
 - Alternative course test
 - Tests of user documentation

Use Case Testing

Usecase “returning Item



Random Integration Testing

- Multiple Class Random Testing
 1. For each client class, use the list of class methods to generate a series of random test sequences.
Methods will send messages to other server classes.
 2. For each message that is generated, determine the collaborating class and the corresponding method in the server object.
 3. For each method in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits
 4. For each of the messages, determine the next level of methods that are invoked and incorporate these into the test sequence

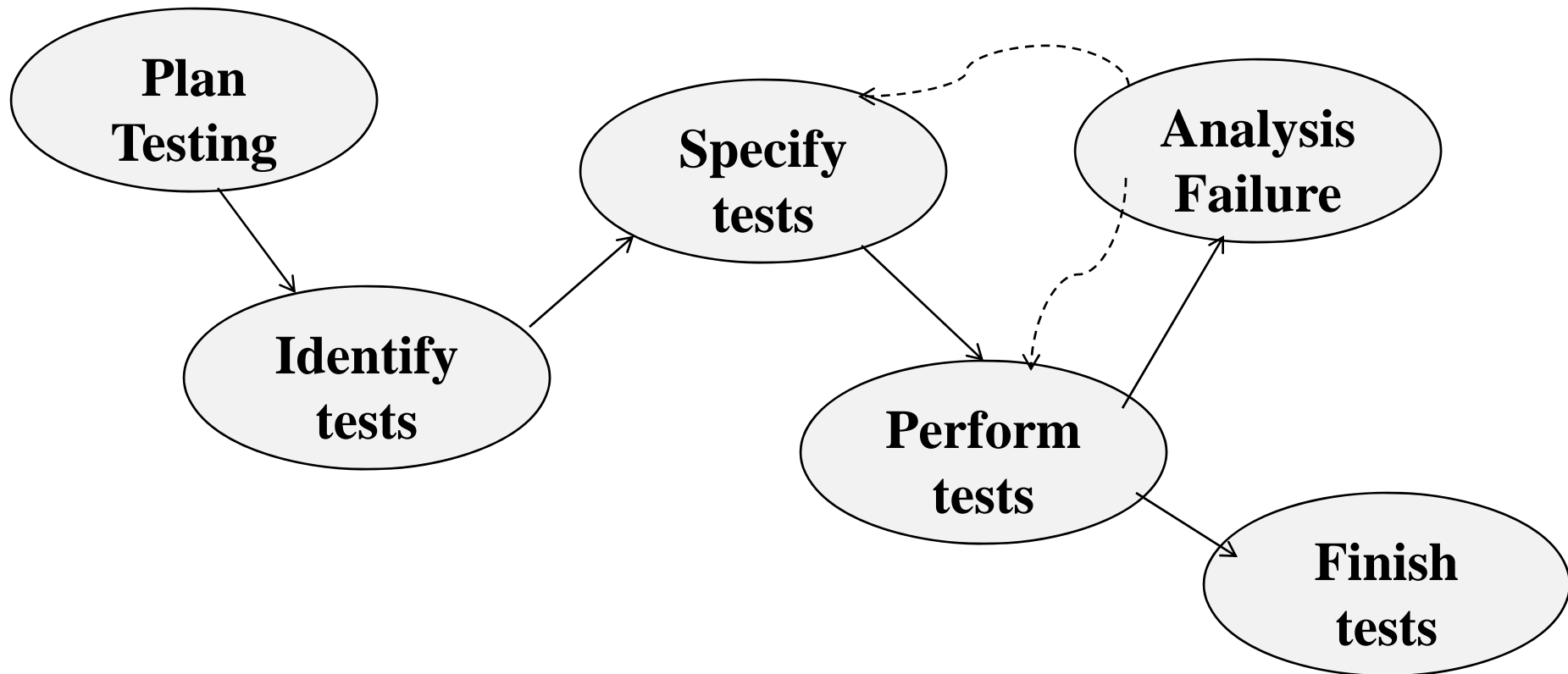
Validation Testing

- Are we building the right product?
- Validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- Focus on user-visible actions and user-recognizable outputs
- Details of class connections disappear at this level
- Apply:
 - Use-case scenarios from the software requirements spec
 - Black-box testing to create a deficiency list
 - Acceptance tests through alpha (at developer's site) and beta (at customer's site) testing with actual customers
- How will you validate your term product?

System Testing

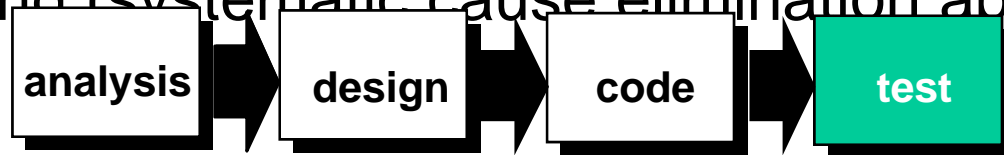
- Each usecase tested separately from an external point of view
- Entire system is tested as a whole
- Types of System Testing:
 - Operation Tests
 - Full Scale Tests
 - Negative Tests
 - Test based on requirement specifications
 - Tests of user documentation

Testing Process



Testing Summary

- Testing affects all stages of software engineering cycle
- One strategy is a bottom-up approach – class, integration, validation and system level testing
- XP advocates test-driven development: plan tests before you write any code, then test any changes
- Other techniques:
 - white box (look into technical internal details)
 - black box (view the external behaviour)
 - debugging (systematic cause elimination approach is best)





Object Oriented Testing Concepts..

- When the testing activities must be commence?
- Justify the statement “More the testing is integrated with software development phases it will be better”.
- Why the most important aspect of testing is actually the attitude adopted toward it?
- Why testing must be planned in the same manner as analysis and construction?
- Why it is not possible to arrive at high quality software just by testing and debugging?
- How testing can be done on analysis and design documents?

Object Oriented Testing Concepts..

- Why reviews and inspections should also be performed in parallel to validation
- Why reviews and inspections consider as costly method?
- How reviews and inspections produce high quality code?
- Justify the statement “Defect avoidance is more powerful than defect removal”.
- Discuss testing strategy. Why bottom up testing strategy is better than top down testing strategy.
- Discuss the principle of automatic testing
- Compare test program with test data
- Why interface simulator is use in automatic testing?



Object Oriented Testing Concepts..

- Whether the test programs are part of the tested product or they are a separate product?
- Why unit test of object-oriented code is more complex than testing ordinary (procedural) code?
- What is difference between ordinary (procedural) unit testing and object oriented unit testing?
- Describe state-based testing.
- What do you mean by state matrix for testing?
- Describe test coverage.
- How polymorphism impact the test coverage?

Object Oriented Testing Concepts..

- How inheritance effect the structural testing?
- If we test all units extensively, is there a need for integration testing, since all units will be correct as they are?
- What are test for a usecase?
- What are the different tests can be done during system test?
Why?
- What are the different activities in testing process?
- Why we need to kept test log during the entire test process?
- What is the purpose of test specification?



Object Oriented Testing Concepts..

- Why we need to prepare test reports while writing test specifications?
- When a test is done what are the different reasons of detected failure?
- What we need to do when testing has been completed?



UNIT III Learning's

Construction

- ✓ Introduction
- ✓ the design model
- ✓ block design
- ✓ working with construction

Testing

- ✓ Introduction
- ✓ on testing
- ✓ unit testing
- ✓ integration testing
- ✓ system testing
- ✓ the testing
- ✓ process

Objective Questions

- Q1. Define Block
- Q2. Define Design model
- Q3. Define Verification
- Q4. Define negative test with example.
- Q5. What is association class? Give example.
- Q6. Differentiate message and signal with example.
- Q7. Define Homogenization with example.
- Q8. Define probe in sequential diagram.
- Q9. Differentiate full-scale test and overload test.

Short Questions

- Q1. What are the reasons for having construction phase in object oriented software engineering?
- Q2. How is Object oriented testing different than procedural testing?
- Q3. What are the guidelines for defining stimuli in an interaction diagram?
- Q4. Write short note on Testing strategies, Automatic Testing and System Testing
- Q5. Define and differentiate full-scale test, performance test & overload test
- Q6. Differentiate Object Oriented Analysis (OOA) and Object Oriented Design (OOD)?
- Q7. What are the consequences of implementation environment? Why analysis objects are required before design objects? Discuss with example.

Long Questions

- Q1. The identification of classes and objects is the hardest part of object oriented analysis and design. Discuss with the help of an example.
- Q2. Explain the procedure of converting analysis model into construction model.
- Q3. Explain different types of System Testing
- Q4. What are the various testing levels in an object oriented system? Explain.
- Q5. Differentiate between implementation and test model of OO system. Take suitable examples.
- Q6. What are the limitations of State Transition Table? How are they overcome?
- Q7. Explain unit and integration testing in the object oriented context. Differentiate between thread based and use case based strategies for integration testing.
- Q8. How is object oriented integration testing different from structural integration testing?
- Q9. Compare and contrast various object oriented integration testing techniques.
- Q10. Discuss the state based testing and system testing.
- Q11. Describe the purpose of the construction phase? What are the main steps to develop design model?
- Q12. Describe the Activities in testing process.
- Q13. Discuss the structure of use cases in the interaction diagram.

Research Problems

Consider the following Library Management System(LIS) :

- The Librarian can create new member records by entering the member's name and address.
- LIS assigns a unique membership number to each new library member. The Librarian can also delete a membership by entering the membership number.
- LIS registers each book issued to a member. When a member returns a book, LIS deletes the book from the member's account and make the book available for future issue.
- When a member returns an overdue book, LIS software computes the penalty charge and prints a bill towards the fine payable by the member.
- A member can input either the name of a book or the name of the author of the book and query about the availability of the book. If available, LIS displays the following :
 - rack number in which book is located,
 - the number of copies of books available for issue
 - number of copies of books already issued

Perform the following. You can make suitable assumptions regarding the details of various features of LIS software but you must clearly write down the assumptions you make.

- a) Draw the use case diagram and give the use case description of issue and return of books.
- b) For implementing the LIS software identify the classes and their interrelationships and represent them in class diagram.
- c) Draw Sequence Diagram for Cash Withdrawal from ATM Machine.

References

1. Ivar Jacobson, "Object Oriented Software Engineering", Pearson, 2004.
2. Grady Booch, James Runbaugh, Ivar Jacobson, "The UML User Guide", Pearson, 2004
3. R. Fairley, "Software Engineering Concepts", Tata McGraw Hill, 1997.
4. P. Jalote, "An Integrated approach to Software Engineering", Narosa, 1991.
5. Stephen R. Schach, "Classical & Object Oriented Software Engineering", IRWIN, 1996.
6. James Peter, W Pedrycz, "Software Engineering", John Wiley & Sons
7. Sommerville, "Software Engineering", Addison Wesley, 1999.
8. http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/userguide.html
9. http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/statemachinediagram.html
10. <http://www.developer.com/design/article.php/2238131/State-Diagram-in-UML.htm>