Chapter 1

Software Engineering

Software Engineering

- The discipline devoted to the design, production, and maintenance of computer programs that are developed on time and within cost estimates, using tools that help to manage the size and complexity of the resulting software products.

The Software Life Cycle

- Problem analysis
- Requirements elicitation
- Software specification
- High- and low-level design
- Implementation

The Software Life Cycle *(Cont'd)*

- Testing and Verification
- Delivery
- Operation
- Maintenance

Programmer Toolboxes

- Hardware—the computers and their peripheral devices
- Software—operating systems, editors, compilers, interpreters, debugging systems, test-data generators, and so on

Programmer Toolboxes *(Con't)*

- Ideaware
  - Algorithms
  - Data structures
  - Programming methodologies
- Design aids
  - CRC cards
  - Unified Modeling Language
  - Scenarios

Programmer Toolboxes *(Con't)*

- Software Concepts
  - Information hiding
  - Data encapsulating
  - Abstraction

Goals of Quality Software

- It works.
- It can be modified without excessive time and effort.
- It is reusable.
- It is completed on time and within budget.

Recommended Program Specification

- Processing requirements
- Sample inputs with expected outputs
- Assumptions
- Definition of terms
- A test plan

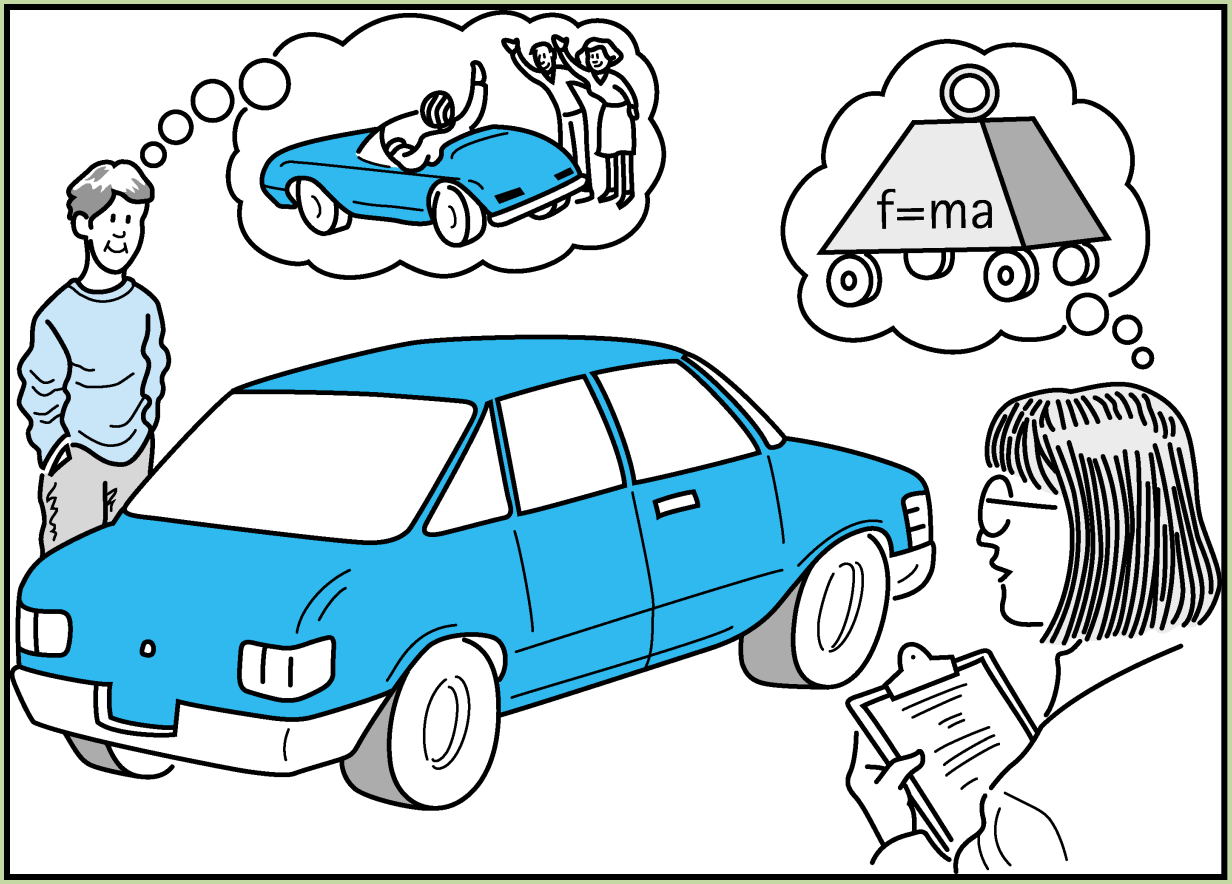
Detailed Program Specification

- Tells *what* the program must do, but *not how* it does it.
- Is written documentation about the program.

Abstraction

- A model of a complex system that includes only the details essential to the perspective of the viewer of the system.
- Programs are abstractions.

Abstraction *(cont'd)*



Information Hiding

- The practice of hiding the details of a module with the goal of controlling access to the details from the rest of the system.
- A programmer can concentrate on one module at a time.
- Each module should have a single purpose or identity.

Stepwise Refinement

- A problem is approached in stages. Similar steps are followed during each stage, with the only difference being the level of detail involved. Some variations:
  - Top-down
  - Bottom-up
  - Functional decomposition
  - Round-trip gestalt design

Visual Aids—CRC cards

Class Name:	Superclass:	Subclassess:
Primary Responsibility		
Responsibilities	Collaborations	

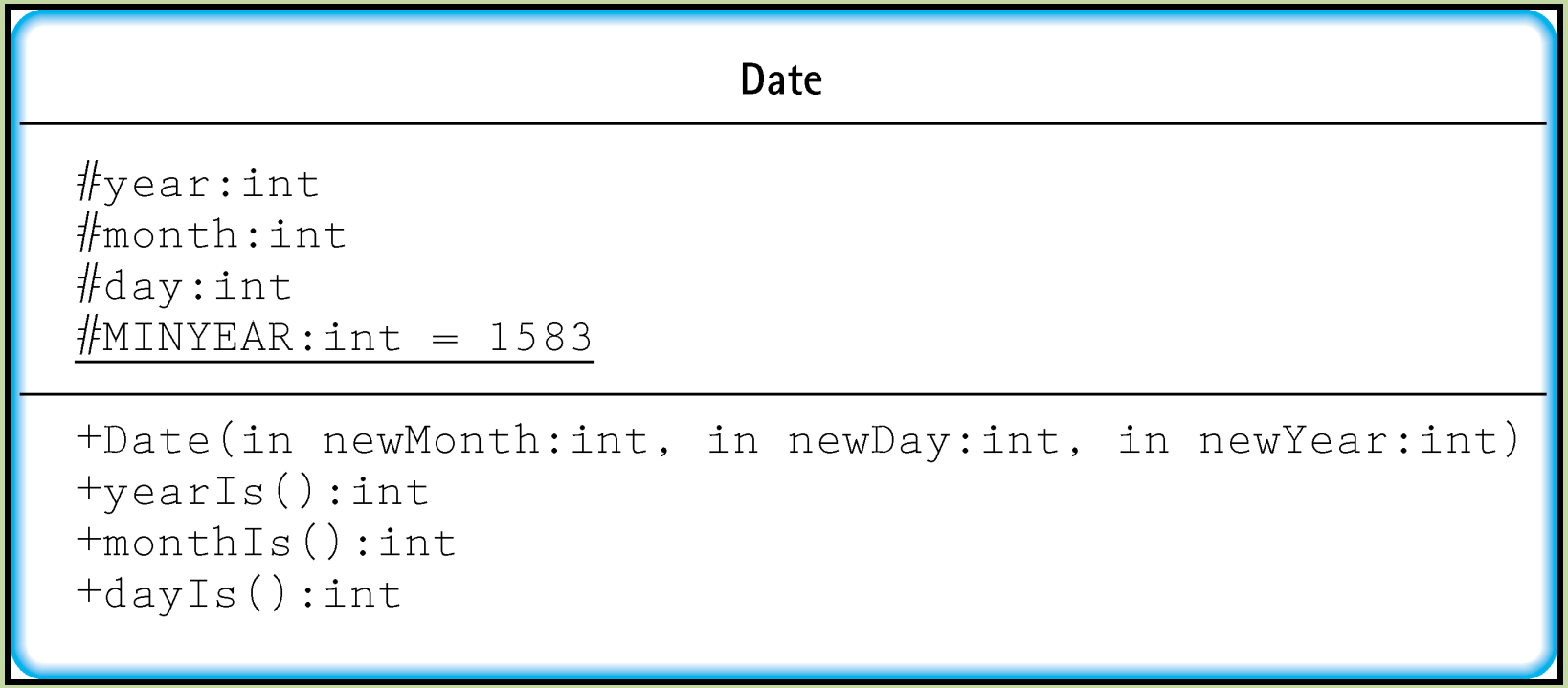
A Short Review of Object-Oriented Programming

- Three inter-related constructs: classes, objects, and inheritance
- Objects are the basic run-time entities in an object-oriented system.
- A class defines the structure of its objects.
- Classes are organized in an “is-a” hierarchy defined by inheritance.

Sample Java Code defines a `Date` Class

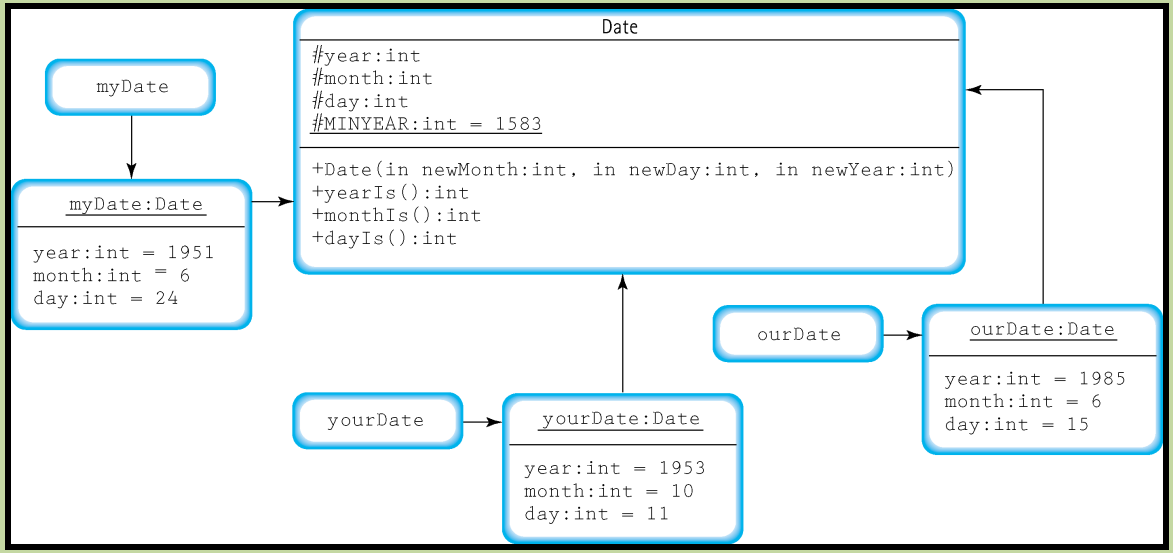
```
public class Date
{
  protected int year;
  protected int month;
  protected int day;
  protected static final int MINYEAR = 1583;

  public Date(int newMonth, int newDay, int newYear)
  // Initializes this Date with the parameter values
{
  month = newMonth;
  day = newDay;
  year = newYear;
 }

public int yearIs()
// Returns the year value of this Date
{
  return year;
}

public int monthIs()
// Returns the month value of this Date
{
  return month;
}

public int dayIs()
// Returns the day value of this Date
{
  return day;
}
```

UML class diagram for the `Date` class



Extended UML class diagram showing `Date` objects

- Date myDate = new Date (6, 24, 1951);
- Date yourDate = new date (10, 11, 1953);
- Date ourDate = new Date (6, 15, 1985);



Inheritance

1. Allows programmers to create a new class that is a specialization of an existing class.
2. The new class is called a subclass of the existing class; the existing class is the superclass of the new class.

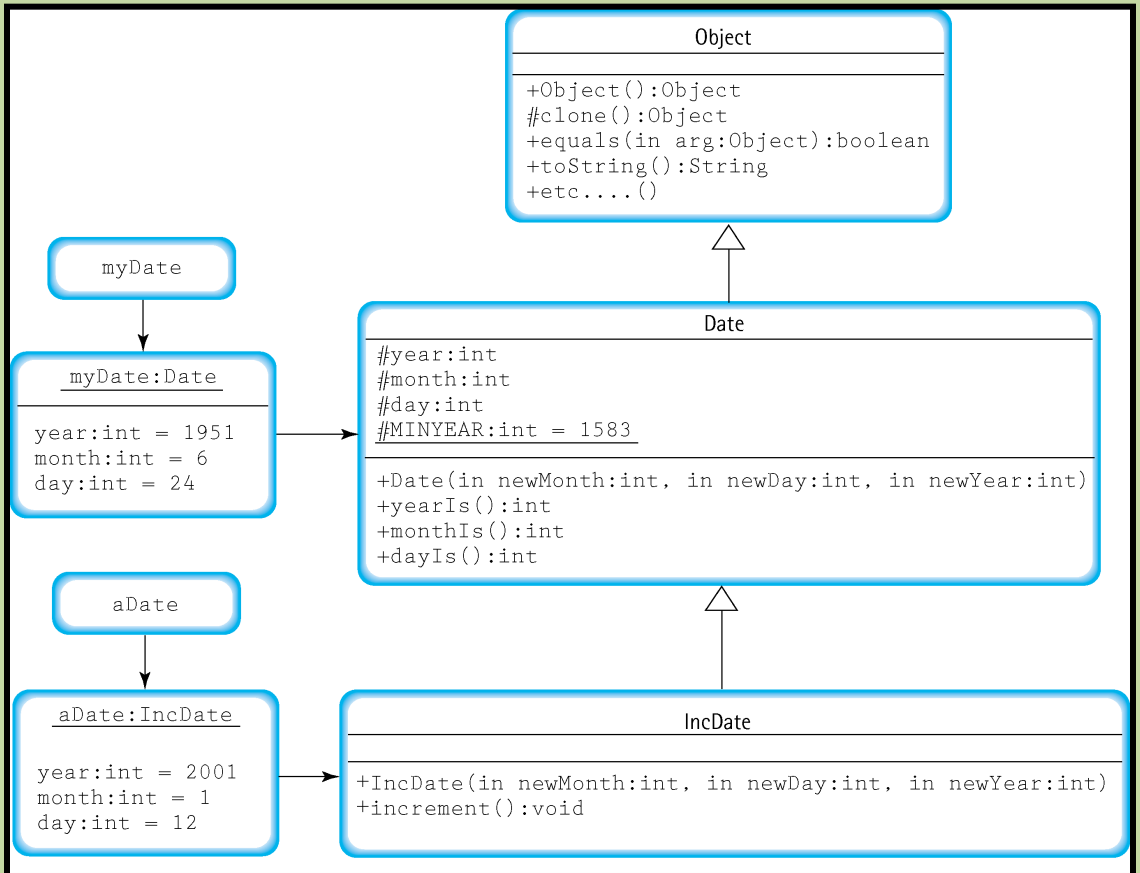
Code to Create `IncDate` in Java

```
public class IncDate extends Date
{
  public IncDate(int newMonth, int newDay, int newYear)
  // Initializes this IncDate with the parameter values
  {
    super(newMonth, newDay, newYear);
   }

  public void increment()
  // Increments this IncDate to represent the next day, i.e.,
  // this = (day after this)
  // For example if this = 6/30/2003 then this becomes 7/1/2003
  {
    // Increment algorithm goes here
  }
}
```

Program Segment Using `Date` and `IncDate`

```
Date myDate = new Date(6, 24, 1951);
IncDate aDate = new IncDate(1, 11, 2001);

output.println("mydate day is:    " + myDate.dayIs());
output.println("aDate day is:     " + aDate.dayIs());

aDate.increment();
output.println("the day after is: " + aDate.dayIs());
```

Extended UML Class Diagram Showing Inheritance



Object-Oriented Design

- Identify classes
- Organize classes in an inheritance hierarchy
- Find prewritten classes

Identifying Classes

1. Start brainstorming ideas.
2. Identify objects in the problem.
3. Objects are usually nouns and operations are usually verbs.
4. Filter the classes.
5. Consider some scenarios in which the objects interact to accomplish a task.
6. CRC cards help us enact such scenarios.

Summary: Approaches to Identifying Classes

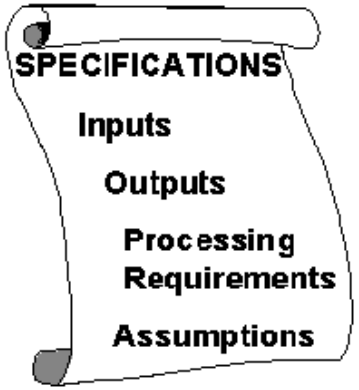
1. Start with the major classes and refine the design.
2. Hide important design decisions and requirements likely to change within a class.
3. Brainstorm with a group of programmers.
4. Make sure each class has one main responsibility.

Summary Continued

5. Use CRC cards to organize classes and identify holes in the design.
6. Walk through user scenarios.
7. Look for nouns and verbs in the problem description.

Program Verification

- Program Verification is the process of determining the degree to which a software product fulfills its specifications.



Verification vs. Validation

Program verification asks,

"Are we doing the job right?"

Program validation asks,

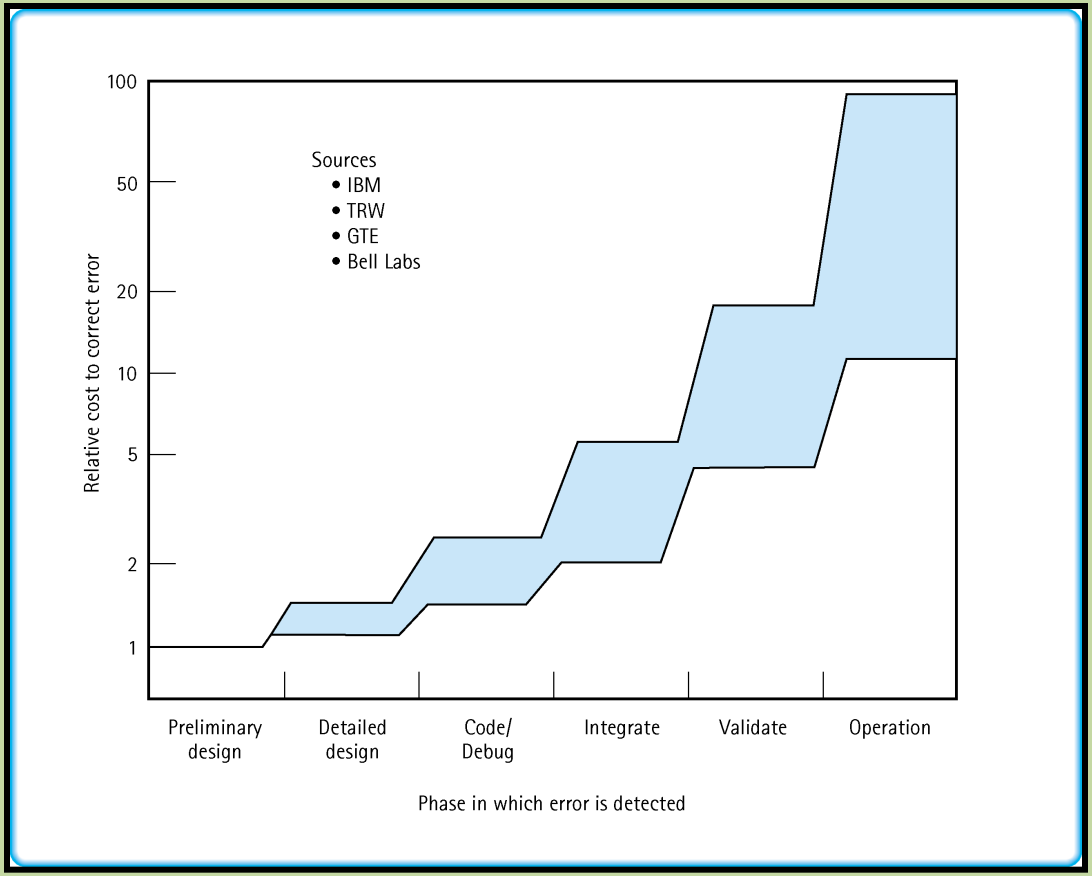
"Are we doing the right job?"

B.W. Boehm, Software Engineering Economics, 1981.

Types of Errors

- Specification
- Design
- Coding
- Input

Cost of a Specification Error Based on When It Is Discovered



Controlling Errors

Robustness The ability of a program to recover following an error; the ability of a program to continue to operate within its environment

Preconditions Assumptions that must be true on entry into an operation or method for the postconditions to be guaranteed.

Postconditions Statements that describe what results are to be expected at the exit of an operation or method, assuming that the preconditions are true.

Design Review Activities

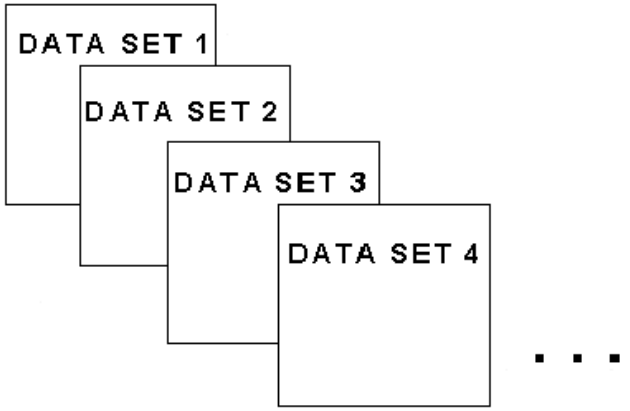
Deskchecking Tracing an execution of a design or program on paper

Walk-through A verification method in which a *team* performs a manual simulation of the program or design

Inspection A verification method in which one member of a team reads the program or design line by line and others point out errors

Program Testing

- **Testing is the process of executing a program with various data sets designed to discover errors.**



For Each Test Case

- Determine inputs.
- Determine the expected behavior of the program.
- Run the program and observe the resulting behavior.
- Compare the expected behavior and the actual behavior.

Types of Testing

Unit testing Testing a class or method by itself

Black-box testing Testing a program or method based on the possible input values, treating the code as a "black box"

Clear (white) box testing Testing a program or method based on covering all of the branches or paths of the code

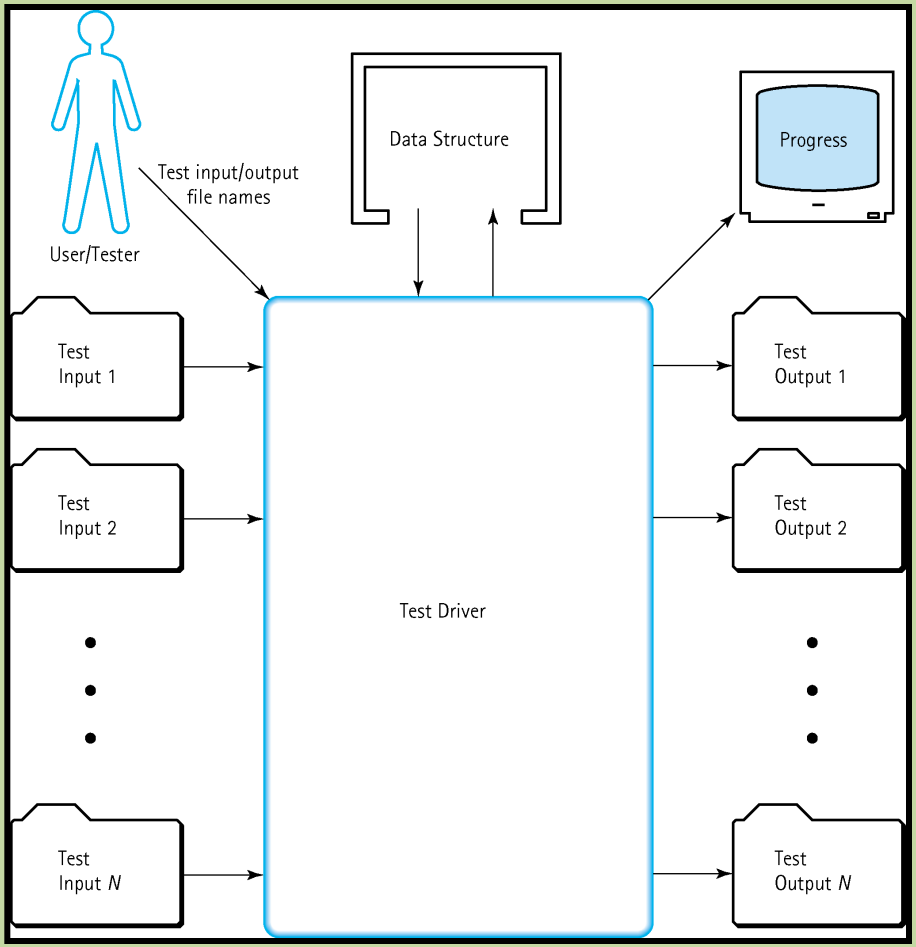
Test Plans

- Document showing the test cases planned for a program or module, their purposes, inputs, expected outputs, and criteria for success
- For program testing to be effective, *it must be planned*.
- Start planning for testing before writing a single line of code.

Operation to be Tested and Description of Action	Input Values	Expected Output
Constructor		
`IncDate`	5, 6, 2000	
print		5/6/2000
Observers		
print `monthIs`		5
print `dayIs`		6
print `yearIs`		2000
Transformer		
`increment` and print		5/7/2000
`IncDate`	5,30,2000	
`increment` and print		5/31/2000
`IncDate`	5,31,2000	
`increment` and print		6/1/2000
`IncDate`	6,30,2000	
`increment` and print		7/1/2000
`IncDate`	2,28,2002	
`increment` and print		3/1/2002
etc.		

file on our web site

Testing Java Data Structures



Obtain the names of the input and output files from the command line
Open the input file for reading and the output file for writing
Read the first line from the input file
Print "Results " plus the first line of the input file to the output file
Print a blank line to the output file
Read a command line from the input file
Set numCommands to 0
While the command read is not 'quit'
    Execute the command by invoking the public methods of the data structure
    Print the results to the output file
    Print the data structure to the output file (if appropriate)
    Increment numCommands by 1
    Print "Command " + numCommands + " completed" to the screen
    Read the next command from the input file
Close the input and output files.
Print "Testing completed" to the screen

method

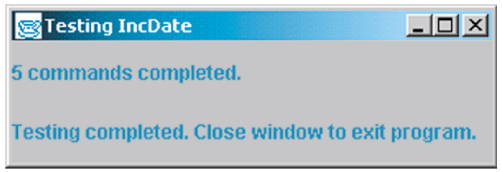
Life-Cycle Verification Activities

```
IncDate Test Data A
IncDate
5
6
2000
monthIs
dayIs
increment
dayIs
quit
```

File: TestDataA

```
Results IncDate Test Data A

Constructor invoked with 5 6 2000
theDate: 5/6/2000
Month is 5
theDate: 5/6/2000
Day is 6
theDate: 5/6/2000
increment invoked
theDate: 5/7/2000
Day is 7
theDate: 5/7/2000
```

File: TestOutputA



Screen

constructs.

Command: java TDIncDate TestDataA TestOutputA

Life-Cycle Verification Activities

Analysis	Make sure that requirements are completely understood.
	Understand testing requirements.
Specification	Verify the identified requirements.
	Perform requirements inspections with your client.
Design	Design for correctness (using assertions such as preconditions and postconditions).
	Perform design inspections.
	Plan testing approach.
Code	Understand programming language well.
	Perform code inspections.
	Add debugging output statements to the program.
	Write test plan.
	Construct test drivers.
Test	Unit test according to test plan.
	Debug as necessary.
	Integrate tested modules.
	Retest after corrections.
Delivery	Execute acceptance tests of complete product.
Maintenance	Execute regression test whenever delivered product is changed to add new functionality or to correct detected problems.