

Chapter 2 Elementary Programming



Motivations

Suppose, for example, that you need to take out a student loan. Given the loan amount, loan term, and annual interest rate, can you write a program to compute the monthly payment and total payment? This chapter shows you how to write programs like this. Along the way, you learn the basic steps that go into analyzing a problem, designing a solution, and implementing the solution by creating a program.



Objectives

- To write programs that perform simple computations (§2.2).
- To obtain input from a program's user by using the **input** function (§2.3).
- To use identifiers to name variables (§2.4).
- To assign data to variables (§2.5).
- To define named constants (§2.6).
- To use the operators **+**, **-**, *****, **/**, **//**, **%**, and ****** (§2.7).
- To write and evaluate numeric expressions (§2.8).
- To use augmented assignment operators to simplify coding (§2.9).
- To perform numeric type conversion and rounding with the **int** and **round** functions (§2.10).
- To obtain the current system time by using **time.time()** (§2.11).
- To describe the software development process and apply it to develop the loan payment program (§2.12).
- To compute and display the distance between two points (§2.13).

Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea



Trace a Program Execution

Assign a radius

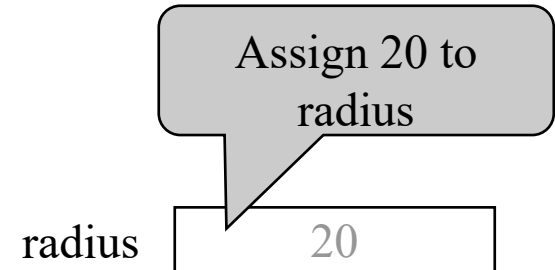
```
radius = 20 # radius is now 20
```

Compute area

```
area = radius * radius * 3.14159
```

Display results

```
print("The area for the circle of radius",  
      radius, "is", area)
```



Trace a Program Execution

Assign a radius

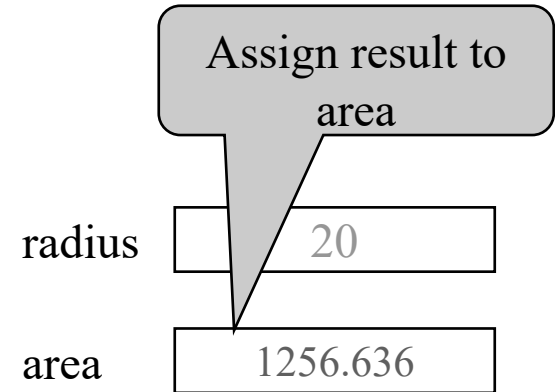
radius = 20 # radius is now 20

Compute area

```
area = radius * radius * 3.14159
```

Display results

```
print("The area for the circle of radius",  
      radius, "is", area)
```



Trace a Program Execution

```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
```

```
print("The area for the circle of radius",
      radius, "is", area)
```

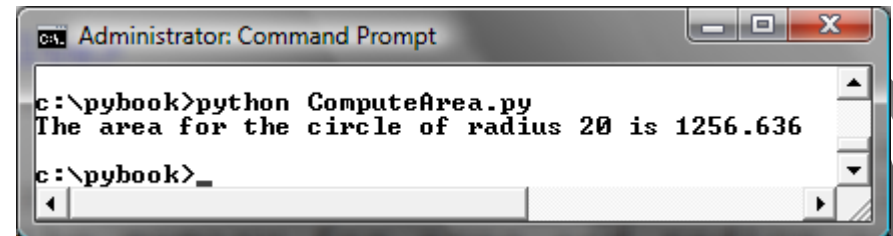
print a message to
the console

radius

20

area

1256.636



```
C:\> Administrator: Command Prompt

c:\pybook>python ComputeArea.py
The area for the circle of radius 20 is 1256.636
c:\pybook>
```

Reading Input from the Console

1. Use the input function

```
variable = input("Enter a string: ")
```

2. Use the float and int function to convert a string to a float or int.

```
var = float(stringVariable)
```

```
var = int(stringVariable)
```

ComputeAreaWithConsoleInput

ComputeAverage



Identifiers

- ❑ An identifier is a sequence of characters that consists of letters, digits, underscores (`_`), and asterisk (`*`).
- ❑ An identifier must start with a letter or an underscore. It cannot start with a digit.
- ❑ An identifier cannot be a reserved word. (See Appendix A, "Python Keywords," for a list of reserved words.) Reserved words have special meanings in Python, which we will later discuss.
- ❑ An identifier can be of any length.



Variables

```
# Compute the first area
radius = 1.0
area = radius * radius * 3.14159
print("The area is ", area,
      " for radius ", radius)

# Compute the second area
radius = 2.0
area = radius * radius * 3.14159
print("The area is ", area,
      " for radius ", radius)
```



Expression

```
x = 1                # Assign 1 to variable x  
radius = 1.0         # Assign 1.0 to variable radius
```

```
# Assign the value of the expression to x  
x = 5 * (3 / 2) + 3 * 2
```

```
x = y + 1           # Assign the addition of y and 1 to x  
area = radius * radius * 3.14159 # Compute area
```



Assignment Statements

`x = 1` `# Assign 1 to x`

`x = x + 1`

`i = j = k = 1`



Simultaneous Assignment

`var1, var2, ..., varn = exp1, exp2, ..., expn`

`x, y = y, x # Swap x with y`

ComputeAverageWithSimultaneousAssignment



Named Constants

The value of a variable may change during the execution of a program, but a *named constant* or simply *constant* represents permanent data that never changes. Python does not have a special syntax for naming constants. You can simply create a variable to denote a constant. To distinguish a constant from a variable, use all uppercase letters to name a constant.



Numerical Data Types

- integer: e.g., 3, 4
- float: e.g., 3.0, 4.0



Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2



The % Operator

$$\begin{array}{r} 2 \\ 3 \overline{) 7} \\ \underline{6} \\ 1 \end{array}$$

$$\begin{array}{r} 3 \\ 4 \overline{) 12} \\ \underline{12} \\ 0 \end{array}$$

$$\begin{array}{r} 3 \\ 8 \overline{) 26} \\ \underline{24} \\ 2 \end{array}$$

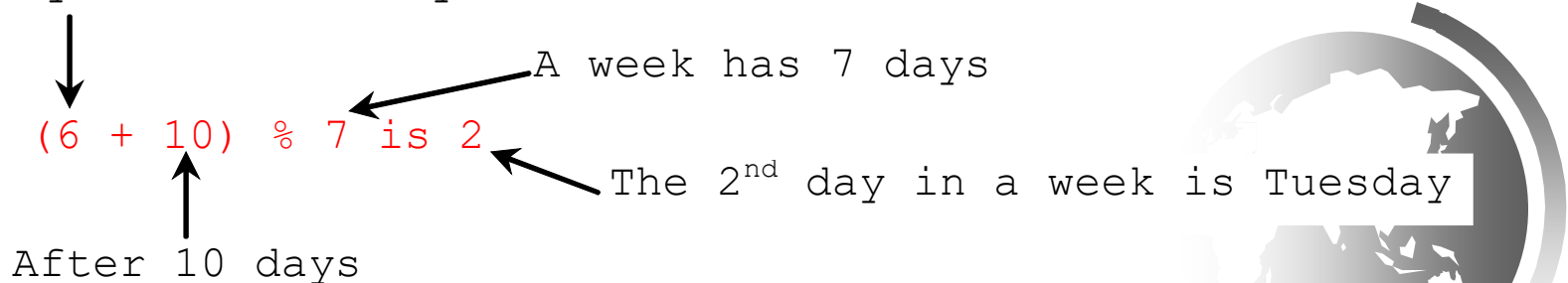
$$\begin{array}{r} 1 \\ \text{Divisor} \rightarrow 13 \overline{) 20} \leftarrow \text{Quotient} \\ \underline{13} \leftarrow \text{Dividend} \\ 7 \leftarrow \text{Remainder} \end{array}$$



Remainder Operator

Remainder is very useful in programming. For example, an even number $\% 2$ is always 0 and an odd number $\% 2$ is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

Saturday is the 6th day in a week



Problem: Displaying Time

Write a program that obtains minutes and remaining seconds from seconds.

DisplayTime



Overflow

When a variable is assigned a value that is too large (*in size*) to be stored, it causes *overflow*. For example, executing the following statement causes *overflow*.

```
>>> 245.0 ** 1000
```

```
OverflowError: 'Result too large'
```



Underflow

When a floating-point number is too small (i.e., too close to zero) to be stored, it causes *underflow*. Python approximates it to zero. So normally you should not be concerned with underflow.



Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, $1.23456e+2$, same as $1.23456e2$, is equivalent to 123.456, and $1.23456e-2$ is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.



Arithmetic Expressions

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

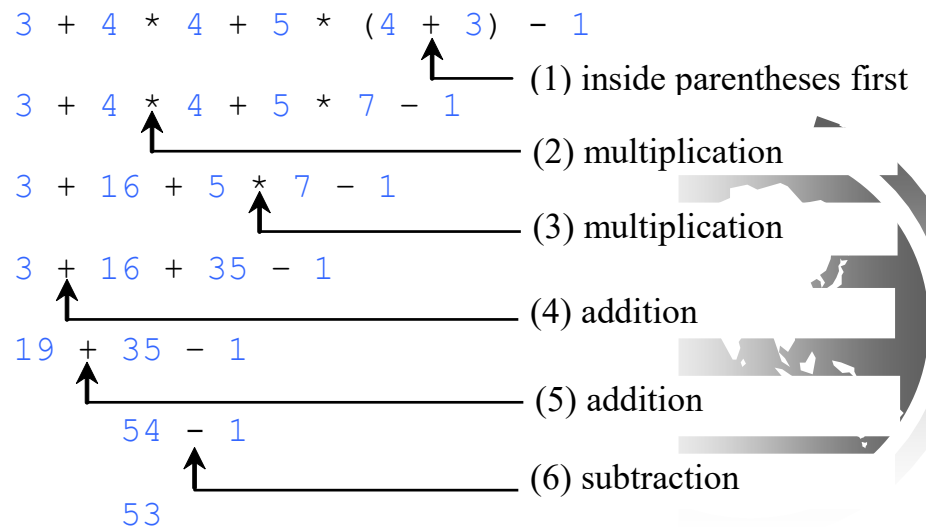
is translated to

$$(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)$$



How to Evaluate an Expression

Though Python has its own way to evaluate an expression behind the scene, the result of a Python expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Python expression.



Augmented Assignment Operators

<i>Operator</i>	<i>Example</i>	<i>Equivalent</i>
<code>+=</code>	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	<code>f -= 8.0</code>	<code>f = f - 8.0</code>
<code>*=</code>	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	<code>i /= 8</code>	<code>i = i / 8</code>
<code>//=</code>	<code>i //= 8</code>	<code>i = i // 8</code>
<code>%=</code>	<code>i %= 8</code>	<code>i = i % 8</code>
<code>**=</code>	<code>i **= 8</code>	<code>i = i ** 8</code>



Type Conversion and Rounding

`datatype(value)`

i.e., `int(4.5) => 4`

`float(4) => 4.0`



Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

SalesTax



Problem: Displaying Current Time

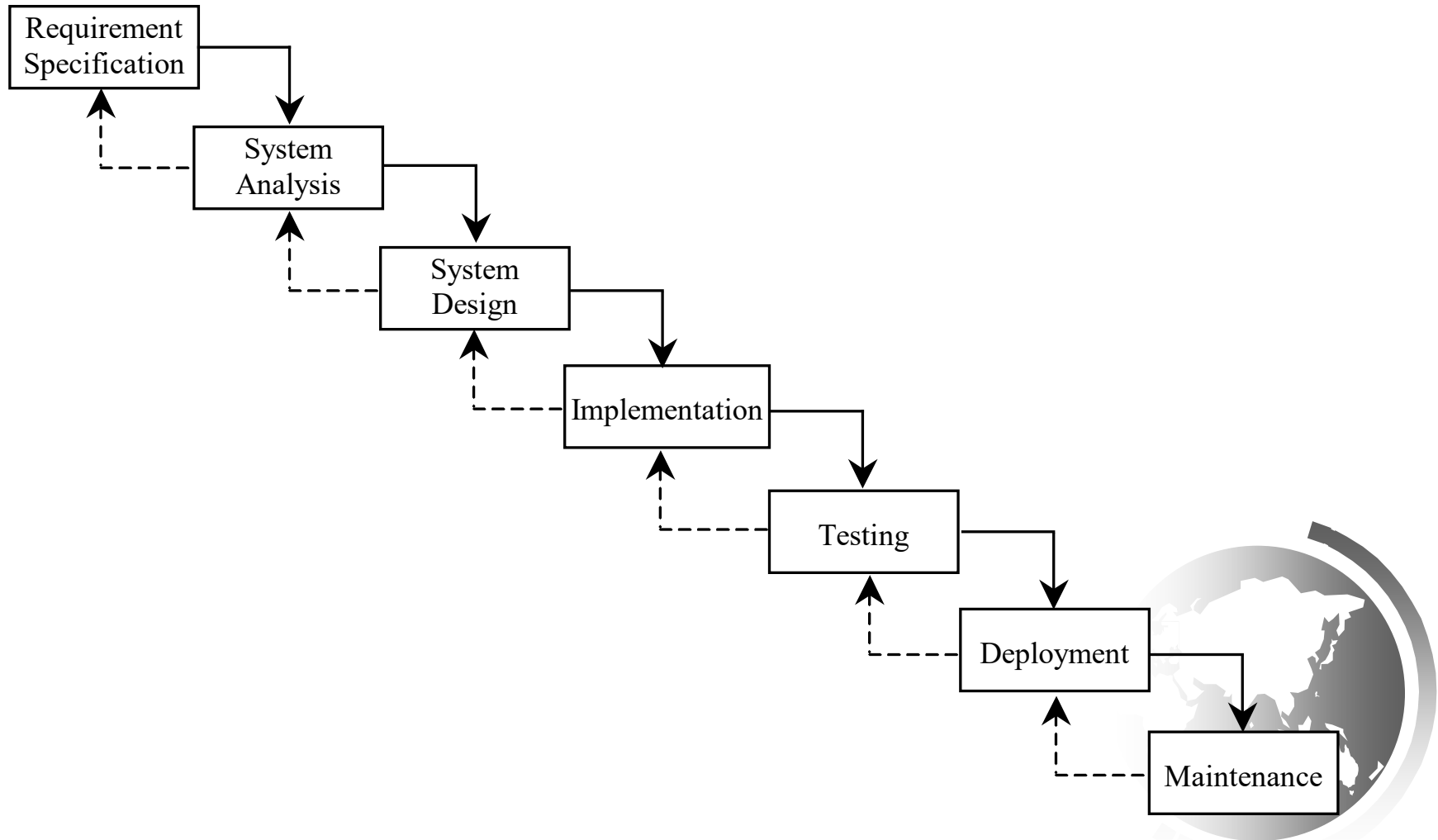
Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.

The `time.time()` function returns the current time in seconds with millisecond precision since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this function to obtain the current time, and then compute the current second, minute, and hour as follows.



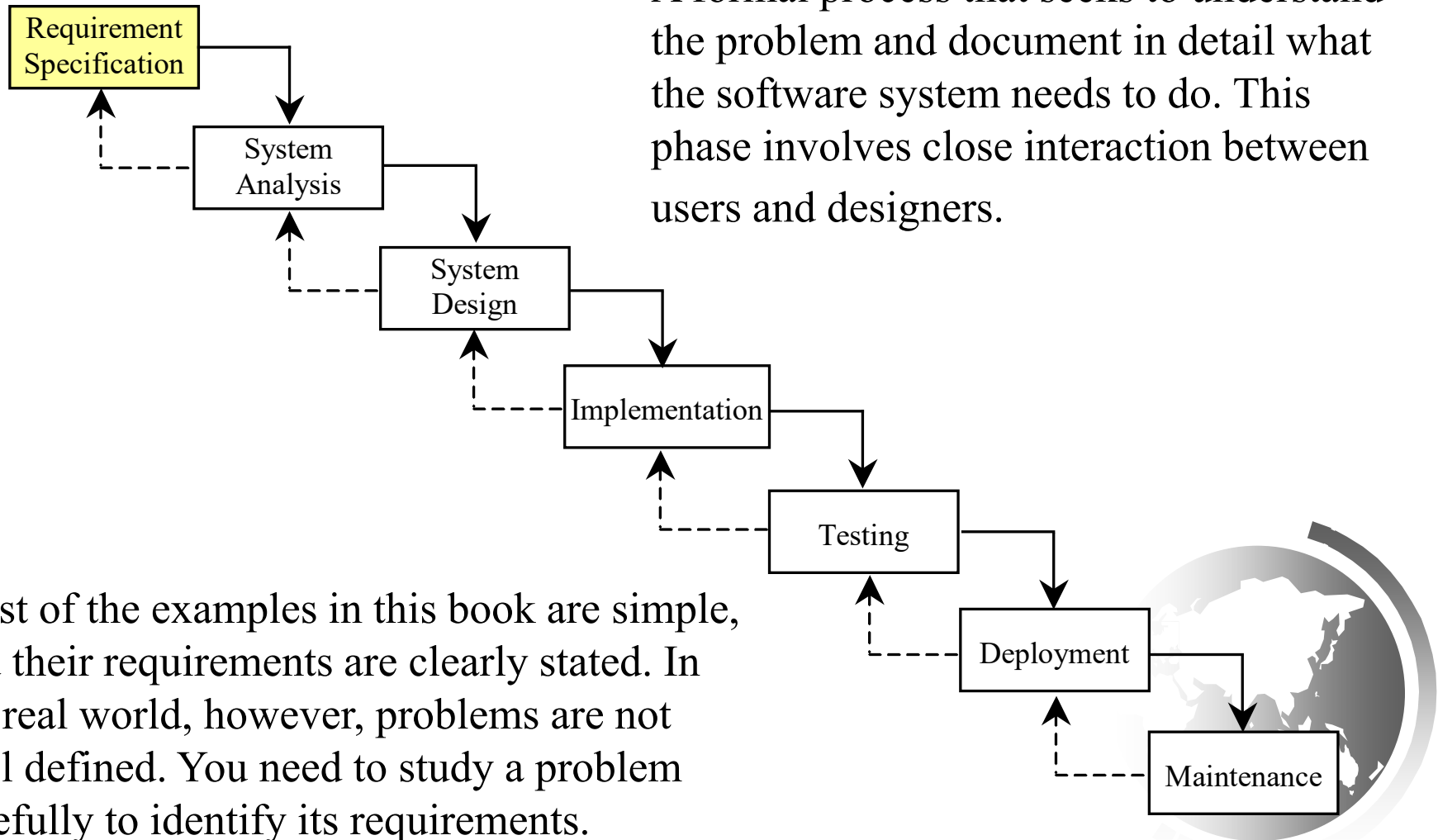
ShowCurrentTime

Software Development Process

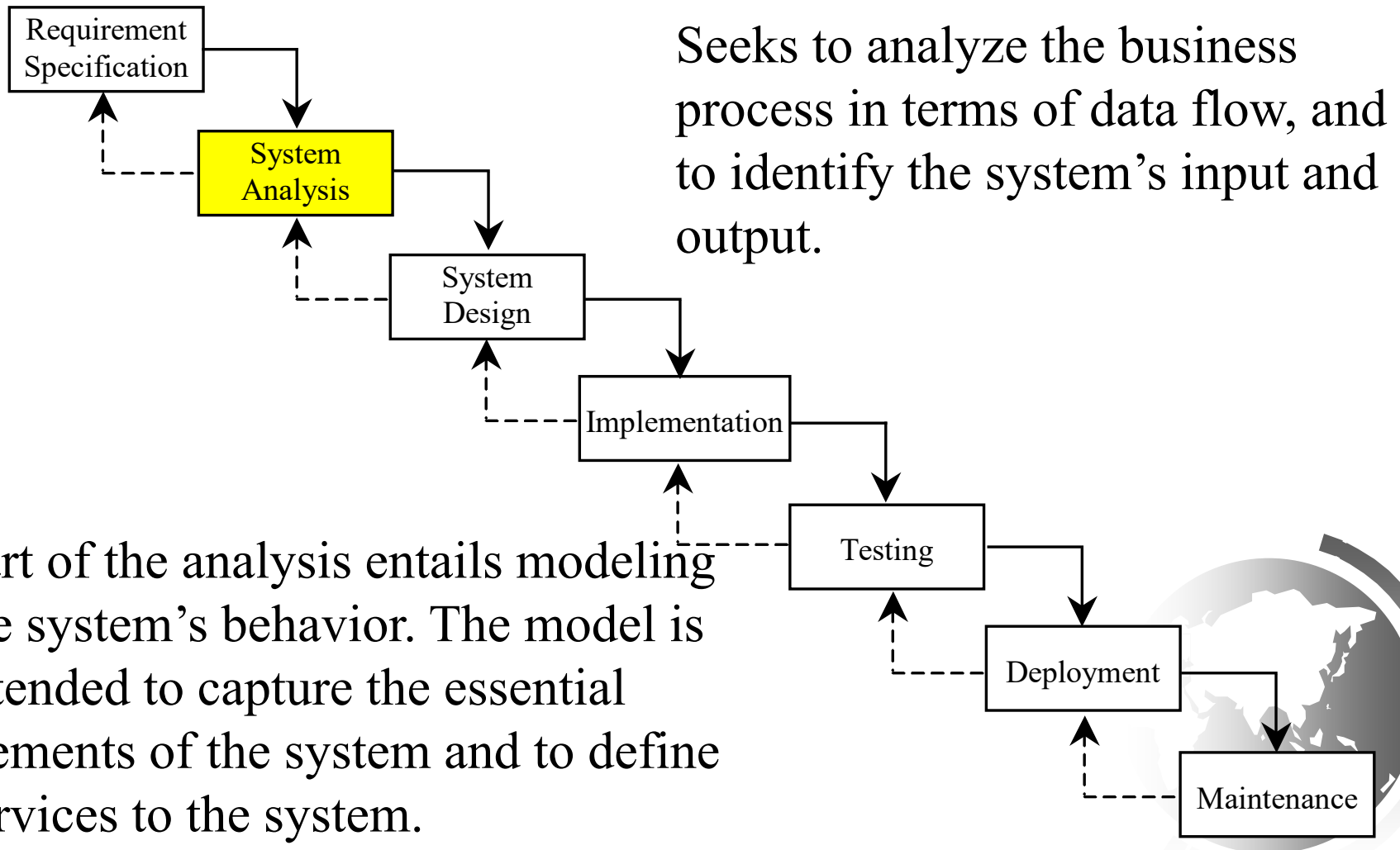


Requirement Specification

A formal process that seeks to understand the problem and document in detail what the software system needs to do. This phase involves close interaction between users and designers.

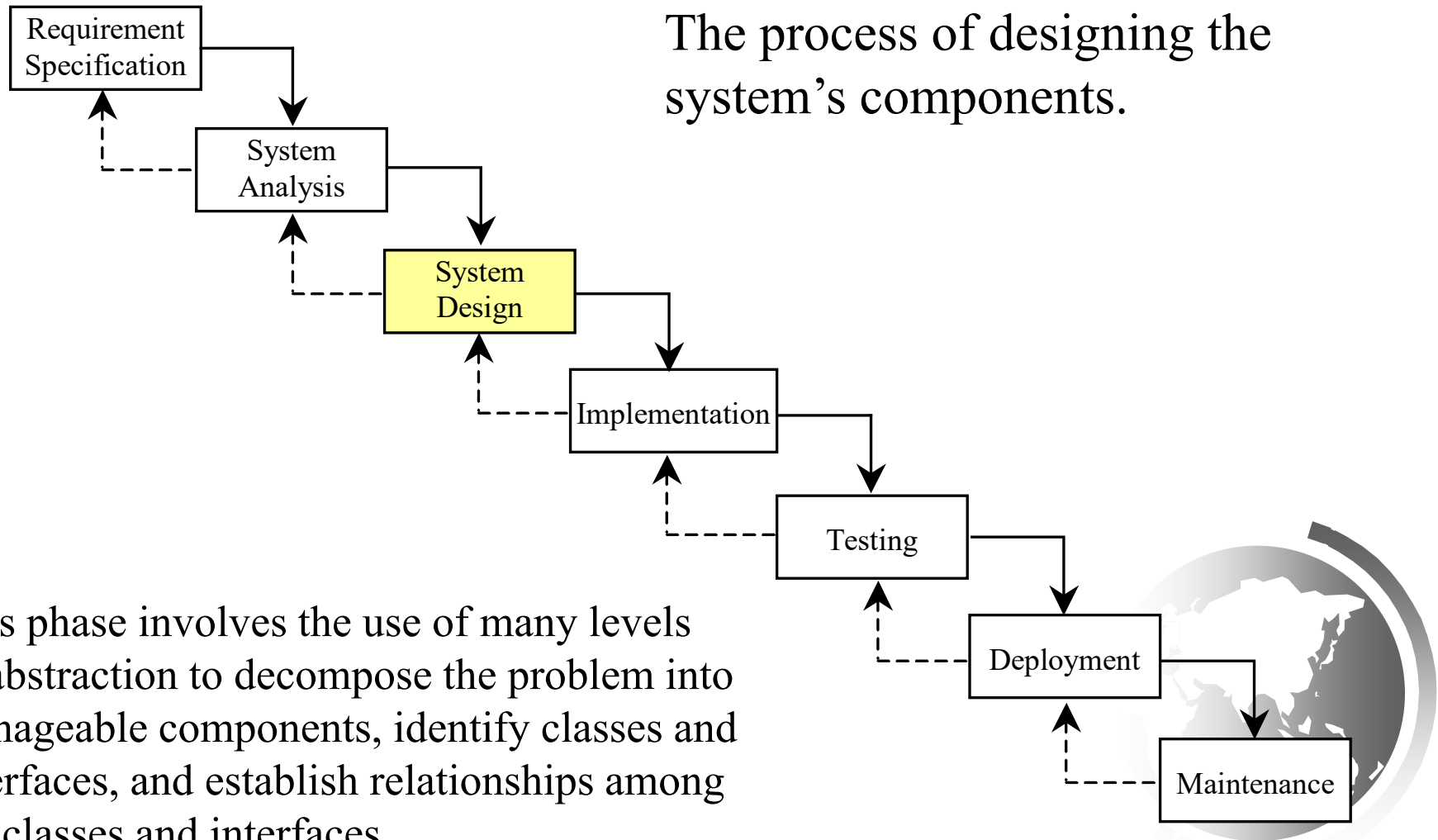


System Analysis



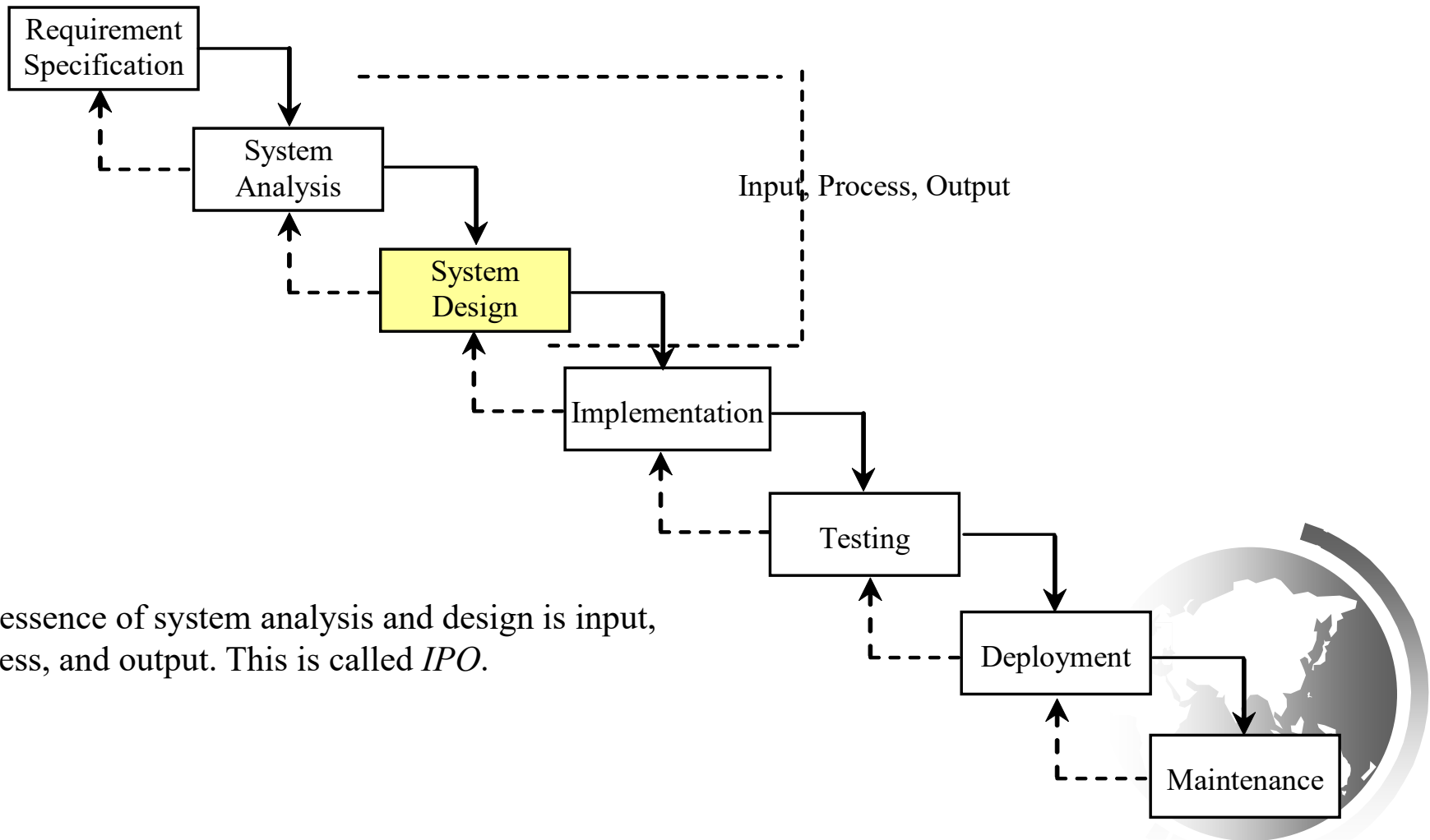
System Design

The process of designing the system's components.



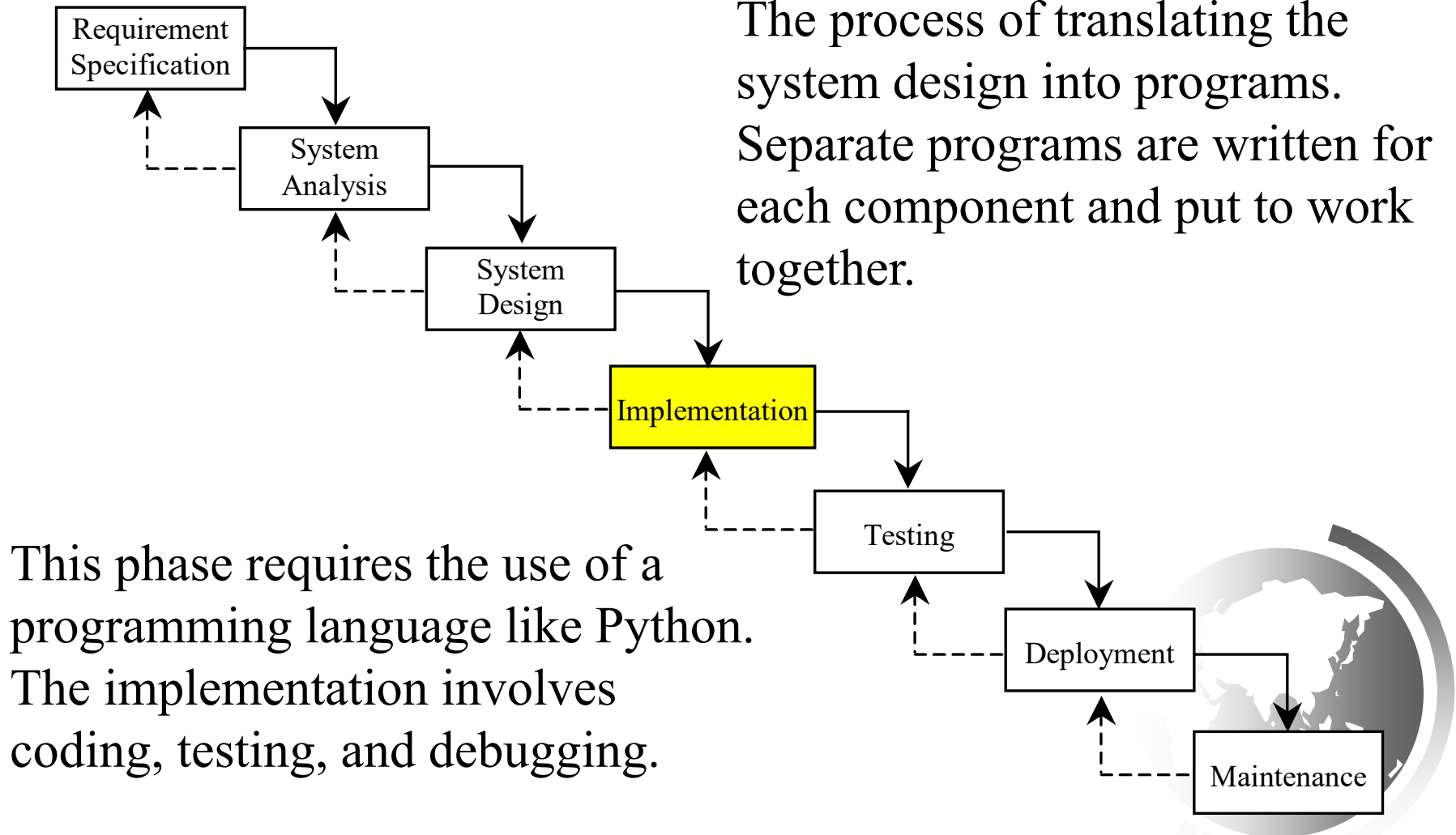
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

IPO



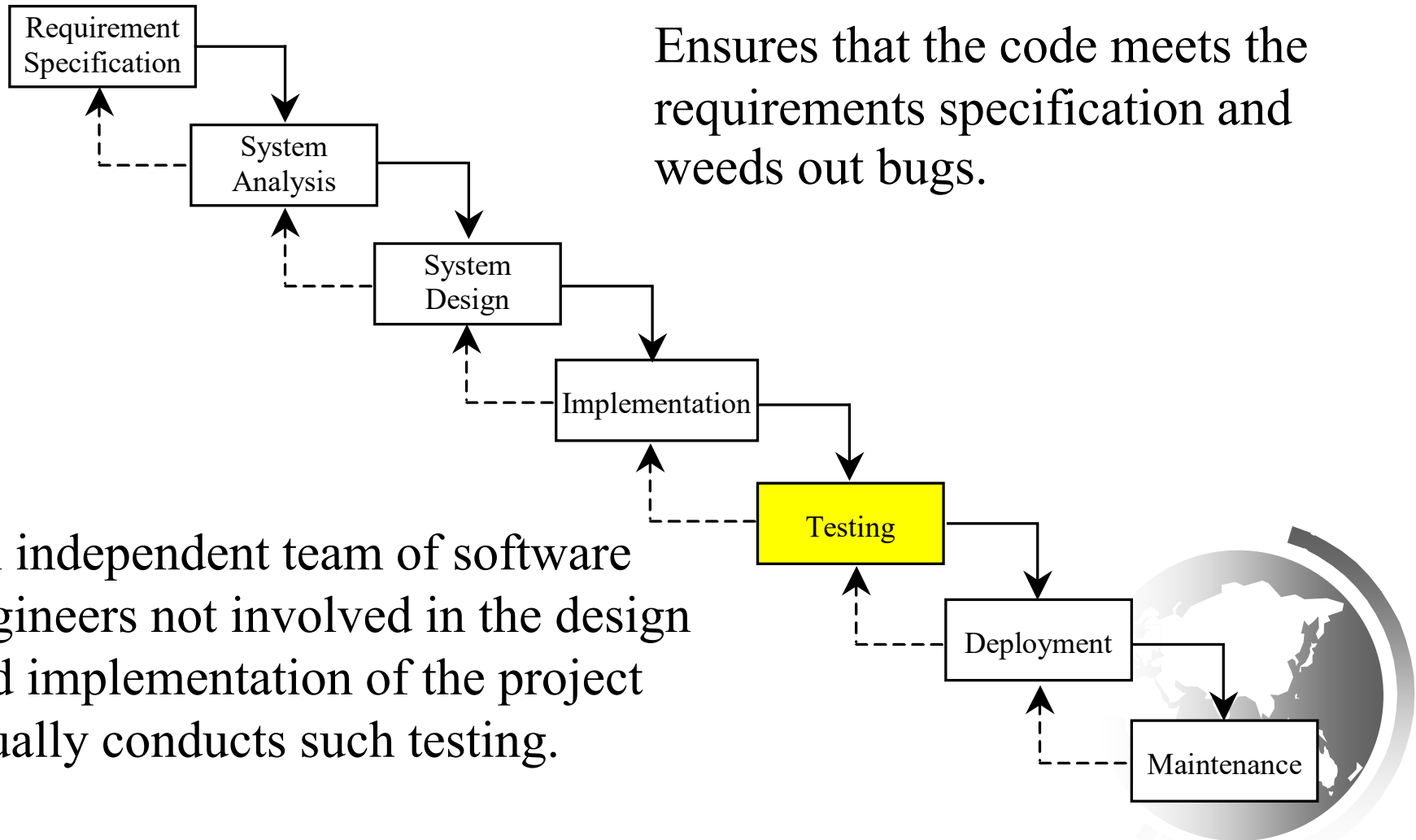
Implementation

The process of translating the system design into programs. Separate programs are written for each component and put to work together.



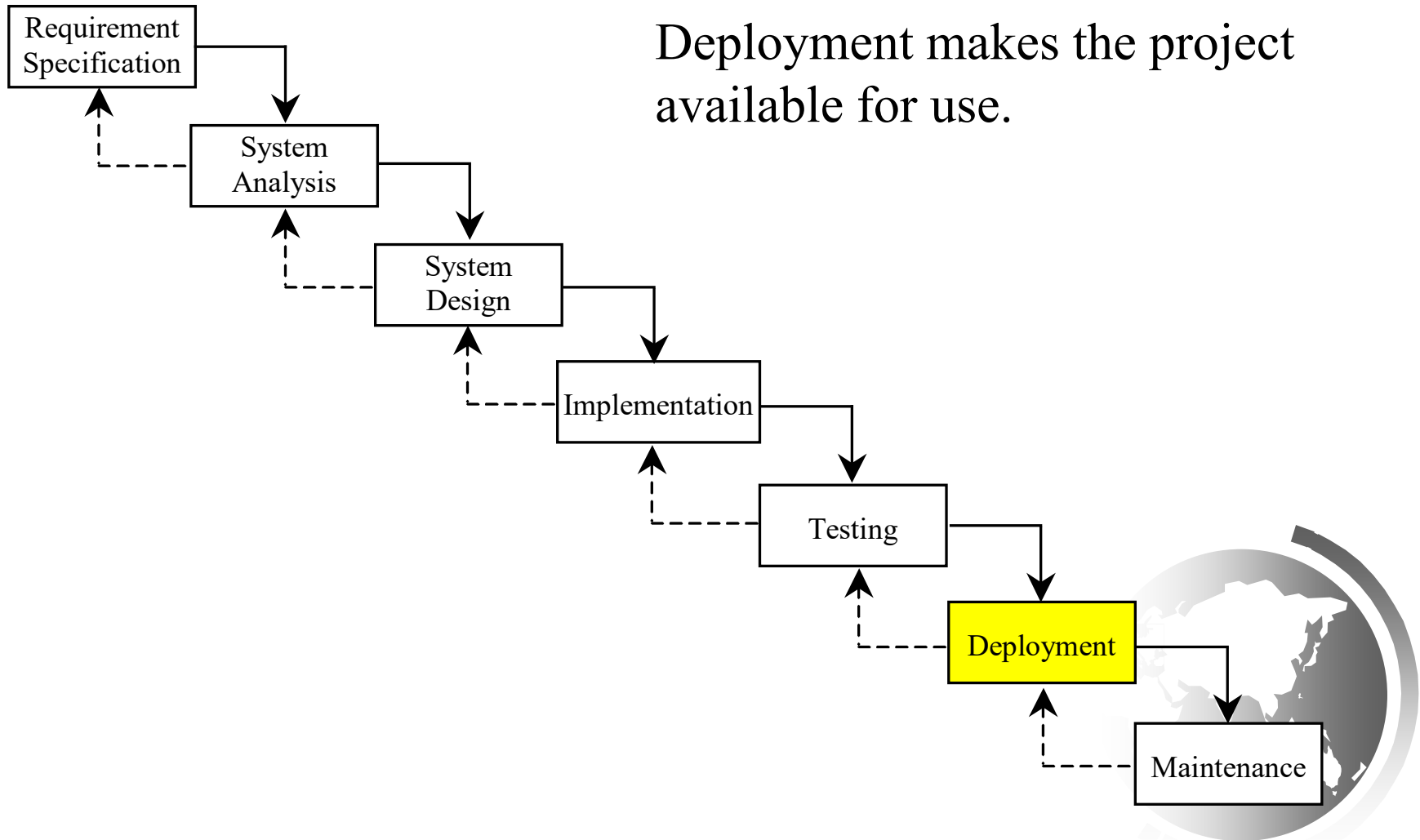
Testing

Ensures that the code meets the requirements specification and weeds out bugs.



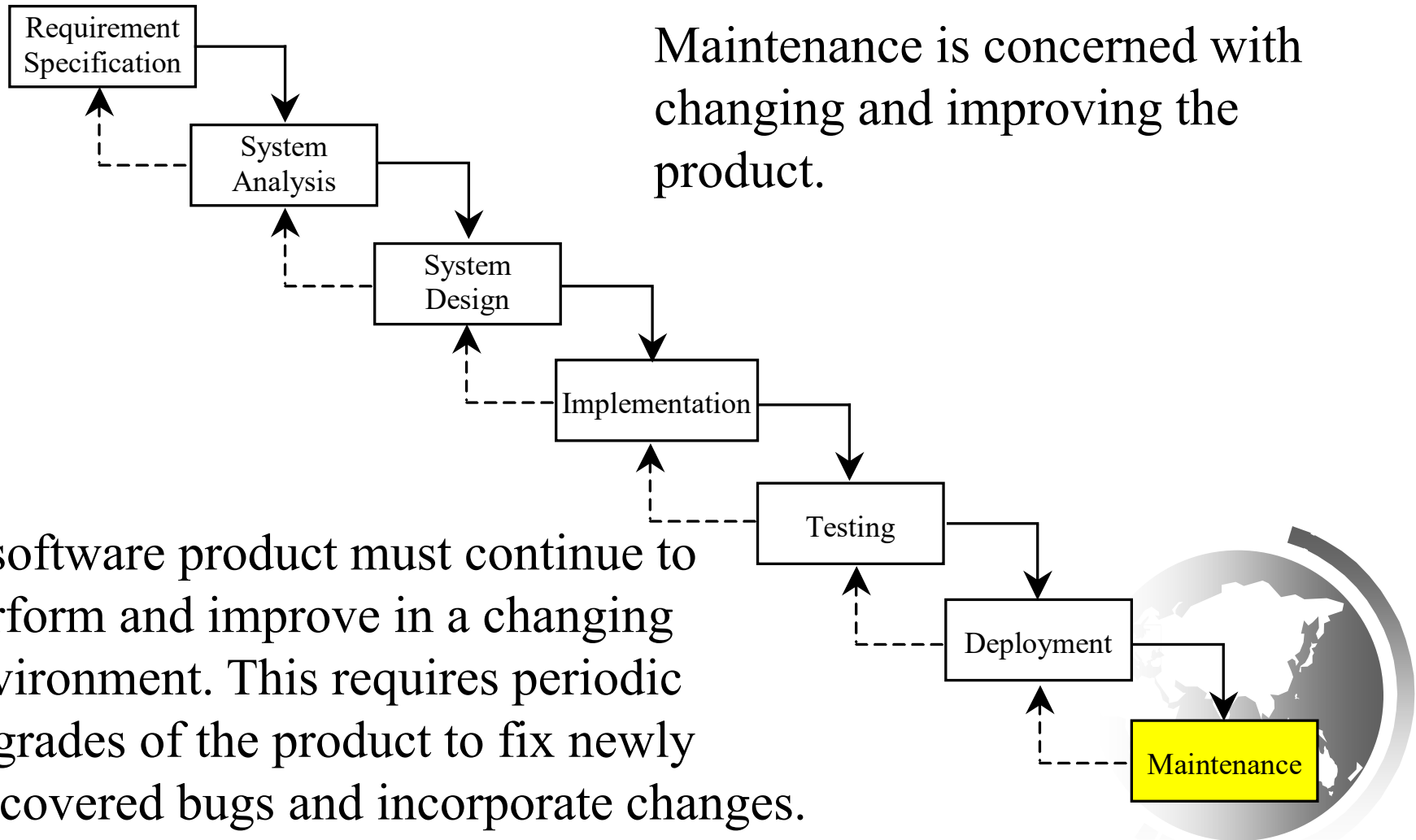
Deployment

Deployment makes the project available for use.



Maintenance

Maintenance is concerned with changing and improving the product.



Problem:

Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

$$\text{monthlyPayment} = \frac{\text{loanAmount} \times \text{monthlyInterestRate}}{1 - \frac{1}{(1 + \text{monthlyInterestRate})^{\text{numberOfYears} \times 12}}}$$

ComputeLoan

Case Study: Computing Distances

This program prompts the user to enter two points, computes their distance, and displays the distance.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

ComputeDistance

