CHAPTER 4 Object Oriented Design & Principles

The essence of Object Oriented Programming will be known only if it is applied appropriately. Object Oriented Analysis and Design is a paradigm on which most modern software systems are built. They provide modularity and flexibility to architect huge systems from small pieces of code. Java and C++ are languages that emulate this paradigm. To program in Object Oriented language, one should think and visualize the software application as a collection of related objects. Believe me, any candidate with Object Oriented Programming knowledge is always rated higher than the others. This chapter provides basic interview questions and answers usually asked on Object Oriented technology along with simple practical problems and solutions.

CONCEPTS

☑ Q1. What is Object Oriented Programming (OOP)?

In a **structure or action oriented** programming language like C, a car is represented by a set of **functions** (named bodies of code) for starting, braking, parking, accelerating and so on. A separate set of variables defines the car's color, number of doors, make, model and so on. You initialize the car's variables and call some functions that operate on those variables to have a car.

An **OOP** language sees a car as an integration of its functions and the variables that hold the car's data (information/detail). The integration of data and function results in an **object**. You create that object simply, at the same time initializing the variables. At any time, the object-oriented program identifies the object it wants to work with and calls the object's functions. Essentially, the object-oriented program thinks in terms of objects (e.g. cars)–not in terms of separate functions (e.g. parking) and variables (e.g. number of doors).

☑ Q2. What are the salient features of Object Oriented Programming (OOP)?

OOP supports following salient features:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

☑ Q3. What is a class and its instance?

A class is a blueprint, or prototype, that defines the variables and methods common to all objects of a certain kind.

An instance is an object of a particular class. The term instance and object are interchangeable. An object has state, behavior and identity where:

- State defines the attribute (data member) of an object.
- Behavior defines method or function used to modify the state of an object.
- Identity is name of an object.

```
class Pen
{
  private:
    Color inkColor;
    Color penColor;
  public:
    void setPenColor(Color);
    void setInkColor(Color);
};

void main()
{
  Pen renold; // Object of Pen
  renold.setPenColor(Blue); // renold calls its behavior
  renold.setInkColor(Green);
}
```

In the above example, **renold** is an object/instance of the **Pen** class. It has

- **State**: inkColor = Green and penColor = Blue
- **Behaviors**: setPenColor(Color) and setInkColor(Color).
- **Identity**: renold

☑ Q4. Explain Abstraction and Encapsulation.

Using the Abstraction principle, only the essential behavior of an object, which distinguishes it from all other objects, is exposed. This means that abstraction allows a designer of an object to ask questions such as 'What an object can do?' rather than 'How it is going to do it?'. In C++, an object's essential behavior will be exposed using public member functions of that object's class.

```
// Incomplete class

class Pen
{
  public:
    void setPenColor(Color );
    Color getPenColor();
    void setInkColor (Color);
    Color getInkColor();
    Bool  isInkAvailable();
};
```

The above example exposes only **essential behavior** of the Pen class's, ignoring its implementation details.

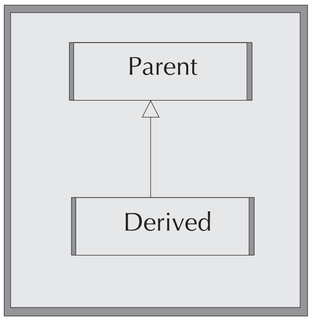
Encapsulation is also known as **Information Hiding**. It is used to describe the process of defining both code and data which form an object. In C++, private data and member functions allow us to implement the encapsulation. The goal of encapsulation is to ensure that the state (data member) of an object can be changed via its public member functions**.**

```
class Pen
{
  private:
    Color penColor;
    Color inkColor;
    int inkLevel;


private:
    int getInkLevel();


public:
    void setPenColor(Color c)
    { penColor = c;};
    Color getPenColor() const;
    {return penColor;};


    void setInkColor(Color c)
    { inkColor = c;};
    Color getInkColor() const;
    { return inkColor;};


Bool isInkAvaliable()
  {
    if(getInkLevel() >=2)
    return true;
    else
    return false;
  }
};
```

In the above example, Encapsulation is achieved by hiding the **state** of any Pen object from the outside world by giving **private access** to it**.** The state must only be obtained/modified using **public** member functions

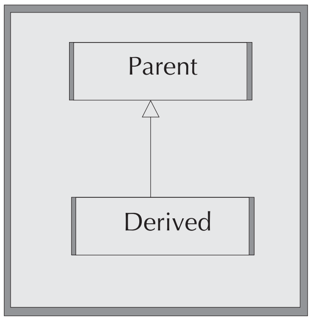
☑ Q5. Explain Inheritance.

The mechanism of deriving a new class from existing class is called inheritance. The existing class is known as base class, super class or parent class; the new class is known as sub class, derived class or child class



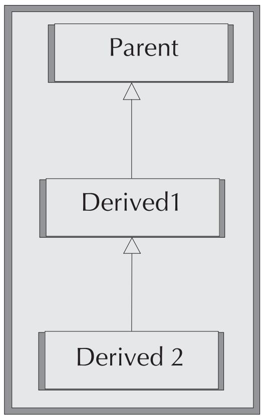
There are different kinds of inheritance namely single, multi-level and multiple inheritance.

Single inheritance means a class derived from only one of the existing class.



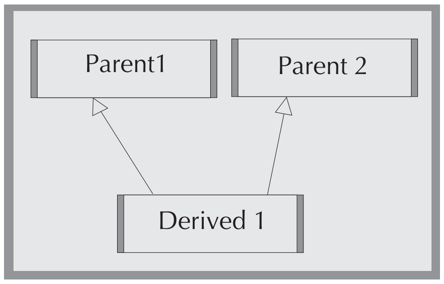
```
class Derived : public Parent
{
};
```

Multi-level inheritance means a class derived from another derived class.



```
class Derived1: public Parent
{
};

class Derived2: public Derived1
{
};
```

Multiple inheritance means a class derived from more than one of the existing classes.



```
class Derived1 : public Parent1, public Parent2
{
};
```

☑ Q6. Explain Polymorphism.

Polymorphism refers to the ability of an object to:

- invoke its function based on the function's signature. This concept is known as **Overloading** and**;**
- invoke its function based on its class type. This concept is known as **Overriding.**

To explain these concepts, let us take a problem:

Design a Graphical User Interface (GUI) which plots a square or rectangle depending on the dimension given. GUI should allow the user to create a maximum of two squares and two rectangles of different dimensions; allow the user to provide a name for each of the newly created squares or rectangles; allow the user to list all the created squares and rectangles by their names and allow the user to view any shape graphically as shown in Figure 4.1.

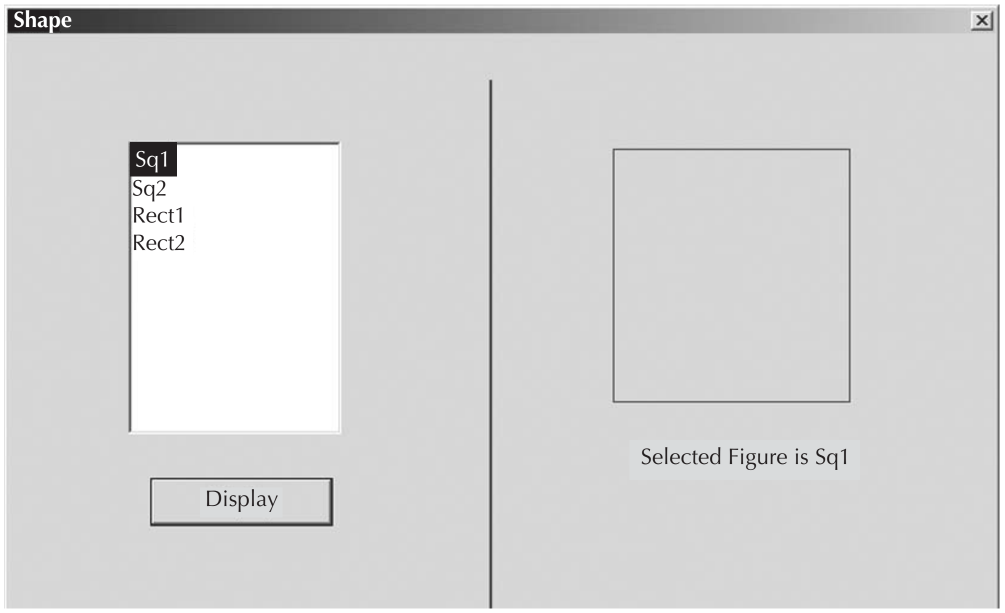


Figure 4.1 Sample View

Chapter 4.qxd 8/10/05 1:03 PM Page 159

Solution 1: Overloading approach

```
struct SQUARE
{
  int side;
}square;
struct RECT
{
  int length;
  int breadth;
}rect;

class  Shape
{
  public:
    void draw (square s)
    {
      //draw the square
    }
    void draw (rect r)
    {
      //draw the rect
    }
};


class GUI
{
    square sq[2];
    rect rec[2];
    ListBox listBox;
    int squareCount, rectCount;
    char * squareNames[2];
    char * rectNames[2];

public:
    void listAdd(char *name,square s)
    {
      listBox.append(name);
      this->sq[squareCount] = s;
      strcpy(squareNames[squareCount++],name);
    }
    void listAdd(char *name,rect r)
    {
      listBox.append(name);
      this->rec[rectCount] = r;
      strcpy(rectNames[rectCount++],name);
    }
    void onDraw()
    {
      char *name = getSelectName();
      /*find the name in the NameList (from both squareNames
      and rectNames list) and get its index */
      /* Using the index value search the square/rect array
      for the selected object and call its corresponding
      draw method. */
    }
};

void main()
{
    square sq1,sq2;
    rect rect1,rect2;
    GUI gui = new GUI();
    gui.listAdd("Sq1", sq1);
    gui.listAdd("Sq2", sq2);
    gui.listAdd("Rect1", rect1);
    gui.listAdd("Rect2", rect2);
}
```

The above GUI class allows user to create two square objects of name "Sq1" and "Sq2"; two rectangle objects of name "Rect1" and "Rect2" using their respective listAdd function. It stores each object into their respective structures. The resultant GUI will look as shown in Figure 4.1.

To plot "Sq1" object, the user selects "Sq1" option from the list and clicks the display button. In response, the onDraw() function of the GUI class is called. The onDraw() function performs the following tasks:

- gets the selected list element name
- verifies the squareNames and rectNames list to find the **index** of selected element. In case of "Sq1", index is 0
- Using this index value, it collects the structure info from square or rect array. In case of "Sq1" it is s[0].
- Now the s[0]->draw() method is invoked

Note: Whenever a new shape (say circle) needs to be added, we need to add new listAdd method, circleName list and circle array member variable in the GUI class. We can avoid this by using **virtual function** as described in the next solution

Solution 2: Overriding approach

```
class Shape
{
  public:
```

```
  virtual void draw() =0;
};

class Square : public shape
{
  public:
    void draw()
    {
      //logic to draw square
    }
};

class Rectangle : public shape
{
  public:
    void draw()
    {
      //logic to draw rectangle
    }
};

class GUI
{
  Shape *shape;
  int shapeCount;
  ListBox listBox;

  public:
    GUI()
    {
      shape = new Shape[4];
      shapeCount = 0;
    }

    //Call to add new list entry
    void listAdd(char *name,Shape *s)
    {
      shape[shapeCount++] = s;
      listBox.append (name);
    }

    void onDraw()
    {
      int index = listBox.getSelectedIndex();
      shape[index]->draw();
    }
    ~GUI()
    {
      delete []shape;
    }
};

void main()
{
    GUI gui = new GUI();
    gui.listAdd("Sq1", new Square(10));
    gui.listAdd("Sq2", new Square(11));
    gui.listAdd("Rect1", new Rectangle(14,12));
    gui.listAdd("Rect2", new Rectangle(114,12));
}
```

The above GUI class allows the user to create two square objects of name "Sq1" and "Sq2"; two rectangle objects of name "Rect1" and "Rect2" using the listAdd() function. It stores all the object pointers into the shape array. GUI will look as shown in Figure 4.1.

To plot "Sq1" object, user selects the "Sq1" option from the list and then clicks the display button. In response, the onDraw() function of the GUI class is invoked which obtains the selected list element's index (which is 0 in the present case). Using this index value, the appropriate object pointer from the **shape** array is obtained and then its draw () method is invoked to draw the corresponding shape.

In this solution, to support a new shape like circle, we don't need to change the GUI class as in the first solution.

Note: This example is given to explain the various polymorphic concepts and not to state that overriding is a better approach that overloading. The approach to be taken is totally dependent on the problem

☑ Q7. What are mutator and accessor method in a class?

Mutators are the method in a class that are used to set the state of an object.

Accessors are the method in a class used to get the state of an object. In C++, accessor functions need to be declared as 'const' methods

In the class Pen for example, the getPenColor() method is called Accessor and setPenColor() method is called Mutator.

☑ Q8. When is an abstract class needed?

Abstract class is a class for which an instance can't be created. For example, a flower in the real world. Have you ever seen an instance of a flower? No. What you see instead are instances of rose or jasmine. 'Flower' represents the abstract concept of things that we all can smell. It doesn't make sense for an instance of flower to exist. However, a rose or jasmine will have certain common behaviors. These common data and behaviors among the rose or jasmine can be grouped under a class named **Flower** and all their specific data and behaviors can be grouped in their respective class named **Rose** or **Jasmine**. Flower class can't be instantiated as it is an abstract concept for the instances of class Rose or Jasmine**.**

UML

☑ Q9. What is an Object Model?

In Object Oriented Programming, all the real world things will be modeled as a collection of Objects with a relationship between them. This representation is called Object Model. A model plays the analogous role in software development that the blueprints and other plans (like site maps, elevations) play in the building of a skyscraper.

☑ Q10. What is UML?

UML stands for Unified Modeling Language. It is used in designing object models for software application. UML helps to specify, visualize and document models of software systems.

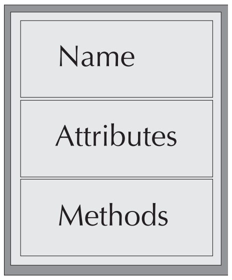
☑ Q11. Mention the different UML diagrams?

UML defines twelve types of diagrams that can be divided into three categories:

- ***Structural Diagrams*** include the Class Diagram, Object Diagram, Component Diagram and Deployment Diagram.
- ***Behavior Diagrams*** include the Use Case Diagram (used during requirements gathering), Sequence Diagram, Activity Diagram and Collaboration Diagram.
- ***Model Management Diagrams*** include Packages, Subsystems and Models.

☑ Q12. Draw UML Class element?

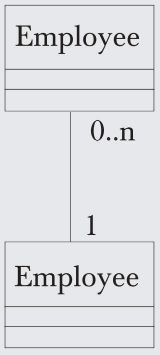
UML class element comprises class name, attributes (data), and methods.

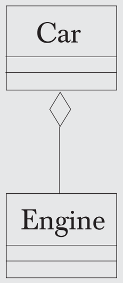
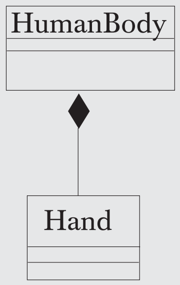
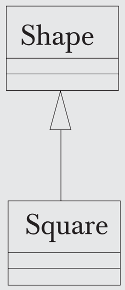
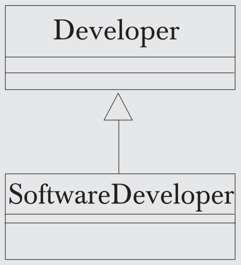


Note: Name cannot be empty while attributes and methods can be.

☑ Q13. How can you represent the relationship between classes in UML?

There are different kinds of relationship possible between any two classes in UML.

Relation	UML representation	Description
Association		If any two classes are related to each other in any way then an association relationship is established between them. In this example, the "Employee works for an Employer" association could be established. The '1' and '0..n' denotes the multiplicity saying there can be more than one employee for an employer

Aggregation		The Aggregation model shows containment relationship where an object is part of another object. In this example, the Engine Object is "**part of**" the Car object. We can also say the Car "**has a**" Engine.
Composition		The Composition model is strong form of aggregation where the contained object will be created and destroyed by the container object. In this example, the hand object will be created by a HumanBody object and used only by that object.
Inheritance/Generalization		The Inheritance model shows "**is a**" relationship. The square object ***is a*** kind of shape.
Realization		Realization is a way in which a set of behaviors from an interface can be used by concrete classes. In this example, Developer interface can be realized by SoftwareDeveloper class.

DESIGN PRINCIPLES

☑ Q14. What are Design Principles?

Design principles provide certain rules that an Object Oriented software designer should consider during the software design phase in order to make:

- each module (class) zero or less dependent on other modules and:
- the module less effected by frequent requirement change

☑ Q15. Explain Open Closed Principle (OCP)?

The OCP states, "A Module should be open for extension, but closed for modification". Here, a module can be a class or a method.

```
//File: Shape.h
enum ShapeType { isCircle, isSquare };

typedef struct Shape
{
   enum ShapeType type;
} shape;

//File: Circle.h


typedef struct Circle
{
   enum ShapeType type;
   double radius;
   Point center;
} circle;
```

```
void drawCircle( circle* );

//File: Square.h

typedef struct Square
{
    enum ShapeType type;
    double side;
    Point topleft;
} square;

void drawSquare( square* );

//File: drawShapes.c

#include "Shape.h"
#include "Circle.h"
#include "Square.h"

void drawSquare( square* s)
{
  printf("Square!\n");
}
void drawCircle( circle* c)
{
  printf("Circle!\n");
}
void drawShapes( shape* s )
{
  switch( s->type )
```

```
   {
     case isSquare:
      drawSquare( (square*)s );
     break;

     case isCircle:
      drawCircle( (circle*)s );
     break;
   }
}
void main()
{
     square s1;
     s1.type = isCircle;
     drawShapes(&s1);
}
```

Output

```
   Circle!
```

In the above example, the function drawShapes is not a closed one because;

- It must be changed every time a new kind of shape (say rectangle) is added to the software.
- Worse, since each different type of shape depends upon the Shape::ShapeType enumeration that is part of all the shape structure, each shape file must be recompiled every time a new kind of shape is added.

Clearly, such structures make the system much harder to maintain and are prone to error. This problem can be solved by several techniques like the two mentioned here.

Dynamic polymorphism

```
class Shape
{
   public:
   virtual void draw () =0;
};

class Square : public shape
{
   public:
   void draw ()
   {
      //logic to draw square
   }
};

class Rectangle : public shape
{
  public:
  void draw ()
  {
      //logic to draw rectangle
  }
};

class GUI
{
  public:
    void drawShapes( shape* s)
    {
      s->draw();
    }
};
```

Here the **GUI::drawShapes** function depends only upon the shape interface. Additional shapes will not cause the **GUI::drawShapes** function to change. Thus, we have created a module (class) that is not closed and can be extended, with new shapes, without requiring modification.

Static Polymorphism

```
template <typename Shape>
void drawShapes(Shape * m)
{
  m->draw ();
}
```

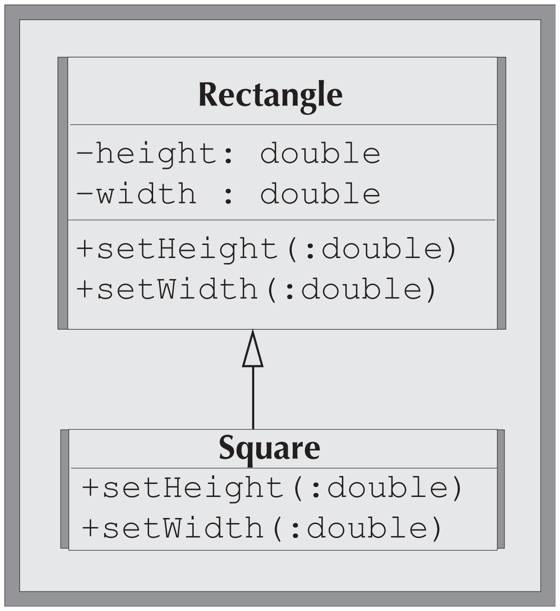
Another technique for conforming to the OCP is through the use of templates or generics. The above example snippet shows how this is done. The drawShapes function can be extended with many different types of shapes without requiring modification.

☑ Q16. Explain the Liskov Substitution Principle (LSP)?

This principle states, “Subclasses should be substitutable for their base classes”. Derived classes should be substitutable for their base classes. That is, a user of a base class should continue to function properly if a derivative of that base class is passed to it.

A Square is a form of a Rectangle. So, usually a Square is derived from a Rectangle as shown below. According to this principle, “Square should be substitutable for Rectangle”.

But here the problem is, the users of Rectangle expect to change height value without changing the width value (and vice versa). However, Square does not fulfill this expectation since height and width of a Square should always be same. This issue could be solved by keeping Square’s width and height as same value i.e. whenever height or width changes, the other one should implicitly need to be changed as shown in the snippet below.



```
void Square::setWidth(double w)
{
  width = w;
  height = w;
}


void Square::setHeight(double h)
{
  width = h;
  height = h;
}
```

Though the above solution looks satisfactory to prove Square is derivative of Rectangle there may exist some other unknown problems like the one mentioned below.

```
void GUI::changeShape(Shape *sp)
{
  sp->setWidth(12);
  sp->setHeight(22);
  sp->draw();
}
```

In the above the snippet, the GUI::changeShape function tries to change the shape's width and height assuming the shape is rectangle. The whole logic will be wrong if the shape is a square. This heavily violates the Liskov principle. So, the derived class Square is not a substitutable for base class Rectangle.

This problem can also be solved using RTTI mechanism as follows:

```
void GUI::changeShape(Shape *sp)
{
  if(typeid(sp) == typeid(Rectangle)
  {
    sp->setWidth(12);
    sp->setHeight(22);
    sp->draw();
  }
}
```

But this solution indirectly violates the OCP principle. So the conclusion is, "Violations of LSP are latent violations of OCP".

DESIGN PATTERNS

☑ Q17. What are design patterns?

Design patterns are standard solutions for certain common problems that occur while designing a software system. For example, whenever we want to restrict an application to create only one instance of a class, we can make use of the Singleton Design Pattern.

☑ Q18. Explain Singleton Design Pattern?

If we need to design a class that can have only one instance then we make use of the Singleton Design Pattern. The rules for creating Singleton class are:

1. Constructor should be **protected**. (It could be made **private** to prevent inheritance)
2. Declare a **static** "self" pointer to hold a reference to the single instance of the class when it is created and initialize it to null.
3. Declare a **static** "instance" function which creates and returns the newly created instance of the class when the static "self" pointer is null, otherwise it returns the previously created instance.
4. Declare a **protected** destructor that deletes the "self" pointer

```
class Singleton
{
  static Singleton *instance;
  protected:
    Singleton ();
    ~ Singleton ( )
    {
      if(instance != NULL)
      {
        delete instance;
        instance = NULL;
      }
    }
  public:
    static Singleton * getInstance()
    {
      if(instance == NULL)
      {
         instance = new Singleton ();
      }
      return instance;
    }
};

Singleton* Singleton::instance = NULL;

void main()
{
  Singleton *single  = Singleton::getInstance();
}
```

PROBLEMS AND SOLUTIONS

☑ Q19. "A dog is an animal". Draw the object model for this statement.

1. The first step involves identifying nouns in the given statement. Here dog and animal are the nouns and all the nouns should be represented as classes in a model diagram.
2. Then identify the relationship among the classes. Here animal and dog have "is a" (inheritance) relationship.

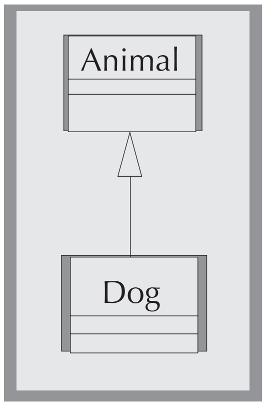


Figure 4.2 Object model

☑ Q20. "A car is a vehicle. It has an engine and four wheels". Draw the object model for this statement.

1. The first step involves identifying **nouns** in the given statement. Here **car, vehicle, engine and wheel** are nouns. All the nouns should be represented as classes in modeling.
2. Then identify the relationship among the nouns. Here:

   o Car and vehicle classes have **is a** (inheritance) relationship.

   o Car and engine classes have **has a** (aggregation) relationship.

   o Car and wheel classes have **has a** relationship. So wheel object is contained by car class. If you notice one more point, a car should have only 4 wheels. This is represented in the model using **multiplicity** notation between Car and Wheel class (1: n) where n should be 4.

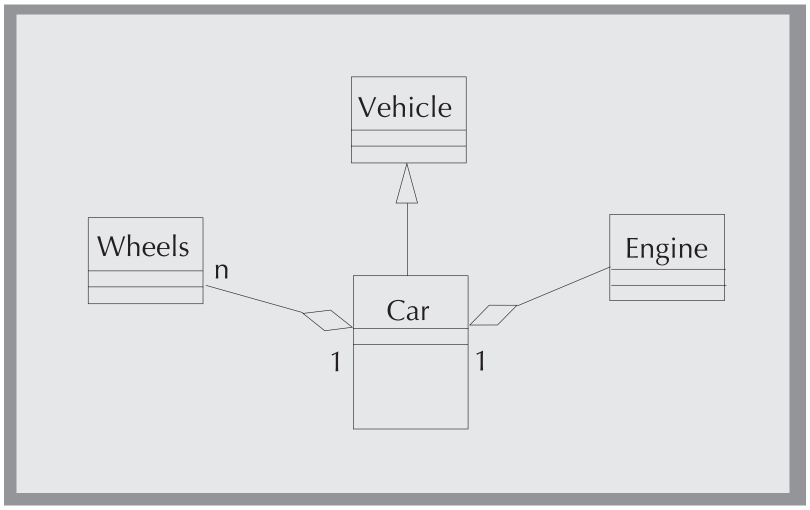


Figure 4.3 Object Model

THINKER'S CHOICE

☑ Q21. Write C++ and Java classes for all the problems given in `Problems and Solution' section.

☑ Q22. Explain Dependency Inversion Principle (DIP)?

☑ Q23. Explain Interface Segregation Principle (ISP)?

☑ Q24. What is adaptor design pattern? Where it will be used?

☑ Q25. What is factory design pattern? Where it will be used?

Additional References

1. Applying UML and Patterns–Craig Larman (Prentice Hall).

2. Design Patterns–by Erich Gamma, Richad Helm, Ralph Johnson, John Vlissides (Addison Wesley).

3. The Unified Modely Language user guide by Grady Booch, *et al* (Addison Wesley).

Day 23–25

Chapter 8

Telephoning

Track 40

Rao makes a phone call to a client's place.

Priya talks to her colleague over the phone.

Guru calls his Japanese friend's house.

Talking on the phone is difficult.

There are lots of polite telephoning expressions in Japanese.

Let's listen to the CD carefully and learn how Rao, Priya and Guru talk over the phone.

8.1 BUSINESS PHONE CALL TO CLIENT (OKYAKUSAMA)

8.1.1 Vocabulary

Irasshaimasu	be, exist, there is (polite of 'imasu')
mōshiwake gozaimasen	sorry (official expression for customer)
...chū	at, on, in, under…(continuous)
orikaeshi gorenraku itashimashō ka?	Would you like him to call you back?
deshō ka?	(polite of 'desu ka?')
… **to** omoimasu	I think…
dewa shitsureishimasu	(phrase saying to end the call)
omodori deshō ka?	Back to seat?
Kakunin	confirm
Eetto…	well, I see (thinking)
suiyōbi	Wednesday
yoroshii	ok, good, fine (polite of 'ii')

8.1.2 Conversation

		English Translation	
ABC:	ABC Info Tech desu.	**ABC:**	Hello, this is ABC Info Tech.
Rao	Indo-Fuji no Rao to mōshimasu ga, eigyō bu no Kimura san wa irasshaimasu ka?	Rao	This is Rao from Indo-Fuji, may I speak to Mr. Kimura from sales department?
ABC	Hai, shōshō omachi kudasai. Mōshiwake gozaimasen. Kimura wa tadaima kaigichū desu. Orikaeshi gorenraku itashimashō ka?	ABC	Yes, please hold on. Sorry Kimura is in a meeting right now. Shall I ask him call you back?
Rao	Lie, kochira kara mata denwa itashimasu. Kaigi wa nanji made deshō ka?	Rao	No, I'll call again. About what time will the meeting get over?
ABC	San ji goro ni wa owaru *to omoimasu*.	ABC	I think it'll finish about 3 o'clock.
Rao	Wakarimashita. Dewa, shitsureishimasu.	Rao	Thanks, bye.
ABC	ABC Info tech desu.	ABC	This is ABC Info tech.
Rao	Indo-Fuji no Rao to mōshimasu ga, Kimura san wa omodori deshō ka?	Rao	This is Rao from Indo-Fuji, is Mr Kimura back at his seat?
ABC	Hai, shōshō omachi kudasai.	ABC	Yes, just moment please.
Kimura	Hai, Kimura desu.	Kimura	Hello, this is Kimura.
Rao	Itsumo osewani natteorimasu Indo-Fuji no Rao desu.	Rao	Hello, this is Rao from Indo-Fuji.
Kimura	Aa Rao san, dōmo.	Kimura	Hello, Mr. Rao.

Rao	Raishū no kaigi ni tsuite kakunin shitai to omoimashite…	Rao	I would like to confirm about the meeting next week.
Kimura	Eetto, raish? no suiyōbi desu ne?	Kimura	Well…next Wednesday right?
Rao	Hai, jikan wa ni ji de yoroshii deshō ka?	Rao	Yes, is 2 o'clock convenient for you?
Kimura	Ni ji … Ee ii desu yo	Kimura	2 o'clock… yes fine.
Rao	Dewa ni ji ni ukagaimasu.	Rao	So I will visit at 2 o'clock.
Kimura	*Hai, wakarimashita.*	Kimura	Ok.
Rao	Dewa shitusreishimasu.	Rao	Thanks. Bye.

8.2 Business Phone Call at Office (NAISEN)

8.2.1 Vocabulary

otsukaresamadesu	hello (only used for person form same company)
seki **o** hazushiteimasu	not in one's seat
nani **ka** otsutae itashimashō ka?	Shall I take a message?
shiryō	material, paper, document
… **to** tsutaete (otsutae) itadakitain desuga	Could you tell …?

8.2.2 Conversation

		English Translation	
Yamada:	Hai, kaihatsu bu Yamada desu.	**Yamada:**	Hello, this is Yamada at development department.
Priya	Otsukaresamadesu, Priya desu ga Nishimura san onegaishimasu.	Priya	Hello, this is Priya. Mr.Nishimura please.
Yamada	Aa…. Nishimura san wa seki o hazushiteimasu.	Yamada	Well… Mr.Nishimura is not at his desk.
Priya	Sō desu ka….	Priya	I see…
Yamada	Nani ka otsutae itashimashō ka?	Yamada	Shall I take a message?
Priya	Ee, kaigi no shiryō o mail de okutte kudasai, *to tsutaete itadakitain desuga.*	Priya	Yes, Could you tell him to send me the meeting document by e-mail?
Yamada	Shiry? o mail de okuru… hai wakarimashita.	Yamada	To send the document by e-mail… Ok.
Priya	Yoroshiku onegaishimasu.	Priya	Thank you.

8.2.3 Special Pronunciation Notes

* mail - mēru

8.3 Phone Call to Friend (tomodachi)

8.3.1 Vocabulary

moshimoshi	hello (only on the phone while speaking to friend, family)
dochira sama desu ka?	Who are you? (polite form)
hatarakimashita (hatarakimasu)	to work
raigetsu	next month
keitaidenwa	mobile phone
bangō	number
osiete (oshiemasu / te form)	to teach
itsudemo	anytime

Cultural Note

Honne and Tatemae

There is a way things are and the way we'd like them to be. The reality and the facade. The real reason and the pretext. The substance and the form. Being direct and being diplomatic. And the truth and the white lie. In short, that is honne and tatemae, respectively.

Since avoiding conflict and trouble is extremely important in Japan, diplomatic language is often used rather than the direct approach. It's said that in formal situations a direct "No" is avoided and there are a thousand nicer alternatives that may be offered. Which can be true, but it depends a lot on the situation and social status of the parties involved. Some westerners unfairly call this deceptive, but this shows more ignorance of how the culture and language are intertwined. The Japanese may say things very politely and vaguely, but if the meaning is not clear it's perfectly acceptable to ask for clarification. But while we in the west judge tatemae to be hypocritical, the Japanese have elevated it to an art. Sometimes, anyway. Like blocking European ski equipment from the Japanese market because "Japanese snow is different". In fact, almost every "reason" for not importing foreign goods is crammed full of tatemae. While many so-called Japan "experts" tell the world about how much the Japanese stress on "harmony", the reality is that they push THE IMAGE of harmony. What lies beneath may be completely different.

8.3.2 Conversation

		English Translation	
Mother:	Hai, moshimoshi Kusano desu.	**Mother:**	Hello, this is Kusano.
Guru	*Moshimoshi* Guru *to mōshimasu ga* Tomoko san wa irasshaimasu ka?	Guru	Hello, this is Guru. May I speak to Ms. Tomoko?
Mother	E? dochira sama desu ka?	Mother	Sorry? Who is calling?
Guru	Indo no Guru desu. Bangalore de Tomoko san to issho ni hataraiteimashita.	Guru	I'm Guru from India. I worked with Ms. Tomoko in Bangalore.
Mother	Aa, sō desu ka. Shitsureishimashita. Chotto matte kudasai ne.	Mother	Oh I see. I'm sorry. Please hold on.
Tomoko	*Moshimoshi*, Guru san?	Tomoko	Hello, are you Mr. Guru?
Guru	Hai. Tomoko san ogenki desu ka?	Guru	Yes, how are you Tomoko?
Tomoko	Hai, genki desu yo. Guru san wa?	Tomoko	I'm fine. How about you?
Guru	Hai, genki desu. Raiget su Tō kyō e ikimasu.	Guru	Yes, I'm fine. I'm going to Tokyo next month.
Tomoko	Hontō? Zehi aimashō ! Watashi no keitaidenwa no bangō o shitteimasu ka?	Tomoko	Really? Let's meet! Do you have my mobile phone number?
Guru	Iie shirimasen. Oshiete kudasai.	Guru	No, I don't. Please tell me.
Tomoko	Zero kyū zero ichi ichi go nana kyū san hachi yon desu.	Tomoko	090 115 7984.
Guru	Wakarimashita. Kondo kara keitaidenwa ni denwashimasu.	Guru	Ok. From next time onwards I'll call on your mobile phone.
Tomoko	Hai itsudemo denwashite kudasai.	Tomoko	Yes, please call me anytime.
Guru	Ja mata.	Guru	Ok bye.
Tomoko	Hai, ja mata ne.	Tomoko	Bye.

8.4 EXERCISE

1. **Moshimoshi _________ desu ga**, **_______ san wa irasshaimasu ka?**

Rei) Moshimoshi Guru desu ga Kimura san wa irasshaimasu ka?

... Hello, this is Guru. May I speak to Mr. Kimura?

1) (your name) / Tanaka

2) Indo-Fuji no (your name) / Priya

3) Indo no (your name) / Watanabe

4) Kaihatsu bu no (your name) / Kondo kachō (Chief Kondo)

5) Bangalore no (your name) / Sakurai buchō (Manager Sakurai)

8.4.1 Special Pronunciation Notes

* shunin (leader), kachō (chief), buchō (manager) … don't call with 'san'

2. ________**to omoimasu**. (Verb **dictionary** form + to omoimasu " I think ….")

Rei) Owarimasu / owaru

San ji goro owaru to omoimasu ... I think it'll be finish around 3 o'clock.

1) Ni ji goro Ikimasu / iku

2) Nagoya de hatarakimasu / hataraku

3) Raishū kimasu / kuru

4) Roku ji ni Kaerimasu / kaeru

5) Rao san wa uchi ni imasu / iru

3. ____________ **to otsutae (tsutaete) itadakitain desuga** (leave message)

Rei) Denwa o kudasai to otsutae itadakitain desuga

..... Please tell him to call me.

1) Shiryō o FAX shitekudasai

2) Ashita yasumimasu (take leave tomorrow)

3) Yo ji ni ukagaimasu

4) Sukoshi okuremasu

5) Kaigi wa go ji kara desu

8.4.2 Special Pronunciation Notes

* FAX - fakkusu

SDLC Process Adoption Scorecard

BHARANIDHARAN R
BHUVANESHWARI M
OMAR FAROOK
SANJU K P
SANKAR JAYARAMAN
GLOBAL eXCHANGE SERVICES INDIA TECHNOLOGY CENTRE PVT LTD

ABSTRACT

Measuring the performance of a project is critical for success. Processes are key areas that help teams to work in a structured and defined way and thereby curtail the scope for mismanagement. Hence adherence to processes is vital to check the entry of chaos and to detect defects early during the lifecycle of any project. It may sometimes be cumbersome for the senior management to look at the project plan for the progress of each project. On the other hand, a scorecard that can give details of the process adherence and the current status will enable the senior management to visualise with a bird's eye view of where the project stands in terms of each of the phases.

Using the scorecard an organisation can achieve a level of standardisation in terms of process across projects that get executed. The paper brings out an understanding of a similar project management measurement tool, which helps organisations to maintain the senior management visibility of each project in terms of process adherence and project status.

INTRODUCTION

Project management is concerned with the overall planning and co-ordination of a project, from inception to completion, aimed at meeting the client's requirements and ensuring completion on time, within cost and to required quality standards.[1]

Success or failure of a software project is directly proportional to the extent of software processes that are followed or adopted. Process adoption means the extent to which a particular project has adhered to the software processes laid out by an organisation. A good insight into the process adoption capability of projects can be achieved by quantitatively measuring the process adoption scores by different projects in an organisation. The process adoption score, in turn, shows the process capability as well as project capability of the organisation. Process adoption scores also show the scope for improvement in areas where they are needed, be it a single project or the entire organisation.

SDLC PROCESS ADOPTION IN PROJECT MANAGEMENT

When an organisation introduces a new process, a change takes place. While the new process may not be an innovation, its novelty in the organisation, undoubtedly, classifies it as an innovation and sometimes, attracts refusals for acceptance. Hence this change must be managed carefully from conception to maturity. When implementing a Software Process Improvement (SPI) program, just as any other change initiative, many organisations focus on the process and technical issues, arriving at the need for a simple and a well-defined process.

There are many areas in project management that are important to achieve a successful project output. One of the areas that can be helpful in providing a smooth flow during project execution is the extent to which a structured process is followed. Process as such evolves differently for different organisations and depends on what is important to the organisation so as to achieve a valuable output. An organisation has to define the organisation process clearly. According to SEI, (Software Engineering Institute, USA) the purpose of Organisation Process Definition as defined in the SW-CMM terminology section is:

“The purpose of Organisation Process Definition is to develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organisation.” [2]

Using the organisation process definition, an organisation can drive process adoption across projects and ensure that it is measurable.

How to measure SDLC process adoption

Process adoption of a software project can be measured by reviewing the processes followed at each phase of the SDLC or at each milestone of a software project and thereby an accumulative process adoption score can be derived at the end of each project.

The scores of each phase can be derived dynamically through the scorecard, which can be updated for each phase. This paper outlines a five-phase approach to measure the process adoption scores for any given project, which is explained in the next few paragraphs.

SDLC Process Adoption Scorecard^{TM}

SDLC Process Adoption Scorecard^{TM} is a Microsoft Excel^{TM} based process measurement system, which calculates the process adoption score of each project in an organisation. This scorecard should get updated at regular intervals during the software development cycle.

The operational definitions for this scorecard mainly includes the evaluation criteria and the completion expediency. An organisation should define an evaluation criteria and completion expediency for each task. Based on the evaluation criteria and completion expediency of the tasks, the process adoption score of the activities and the phases of the project are measured and thereby the overall score of a project is obtained.

A sample set of activities for each phase of the SDLC is shown in Figure 1.

The procedure for calculating the SDLC process adoption score of a project is illustrated in Figure 2 along with the score calculations.

Example of score calculation for each phase

The requirements phase is broadly divided into three activities, namely, business requirements, project plan and technical requirements (see Figure 3).

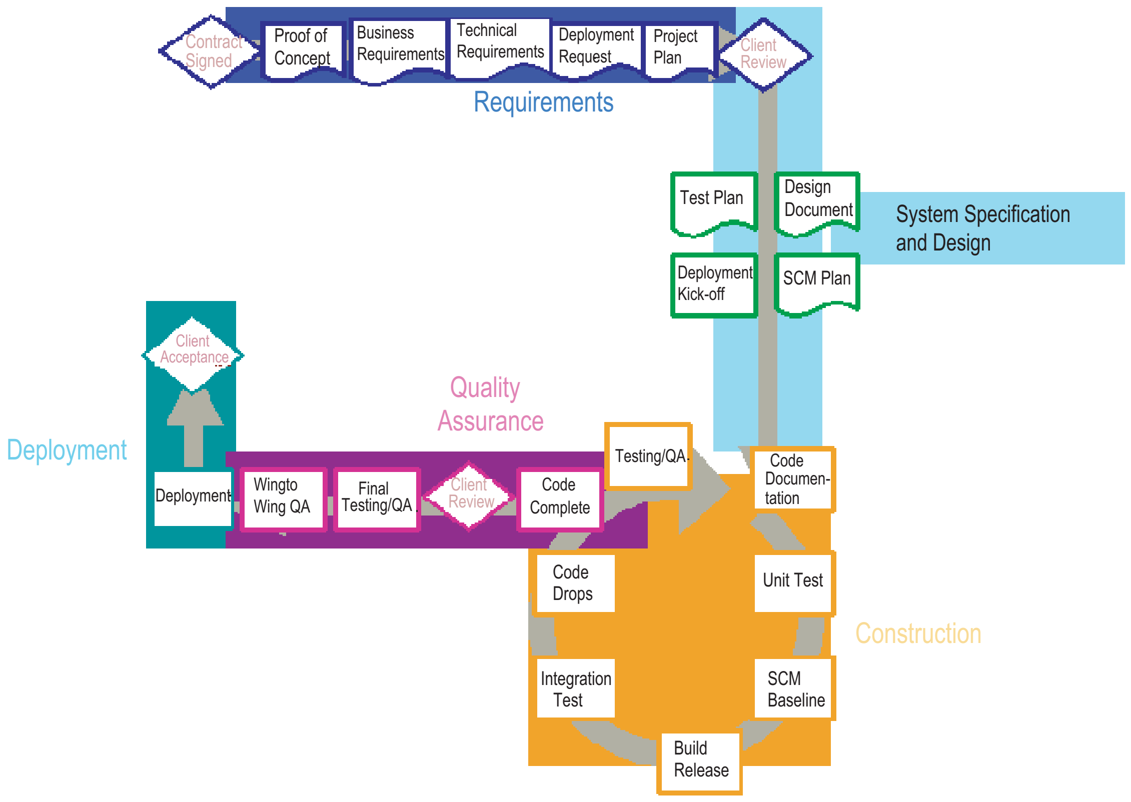


Figure 1 Phases and activities

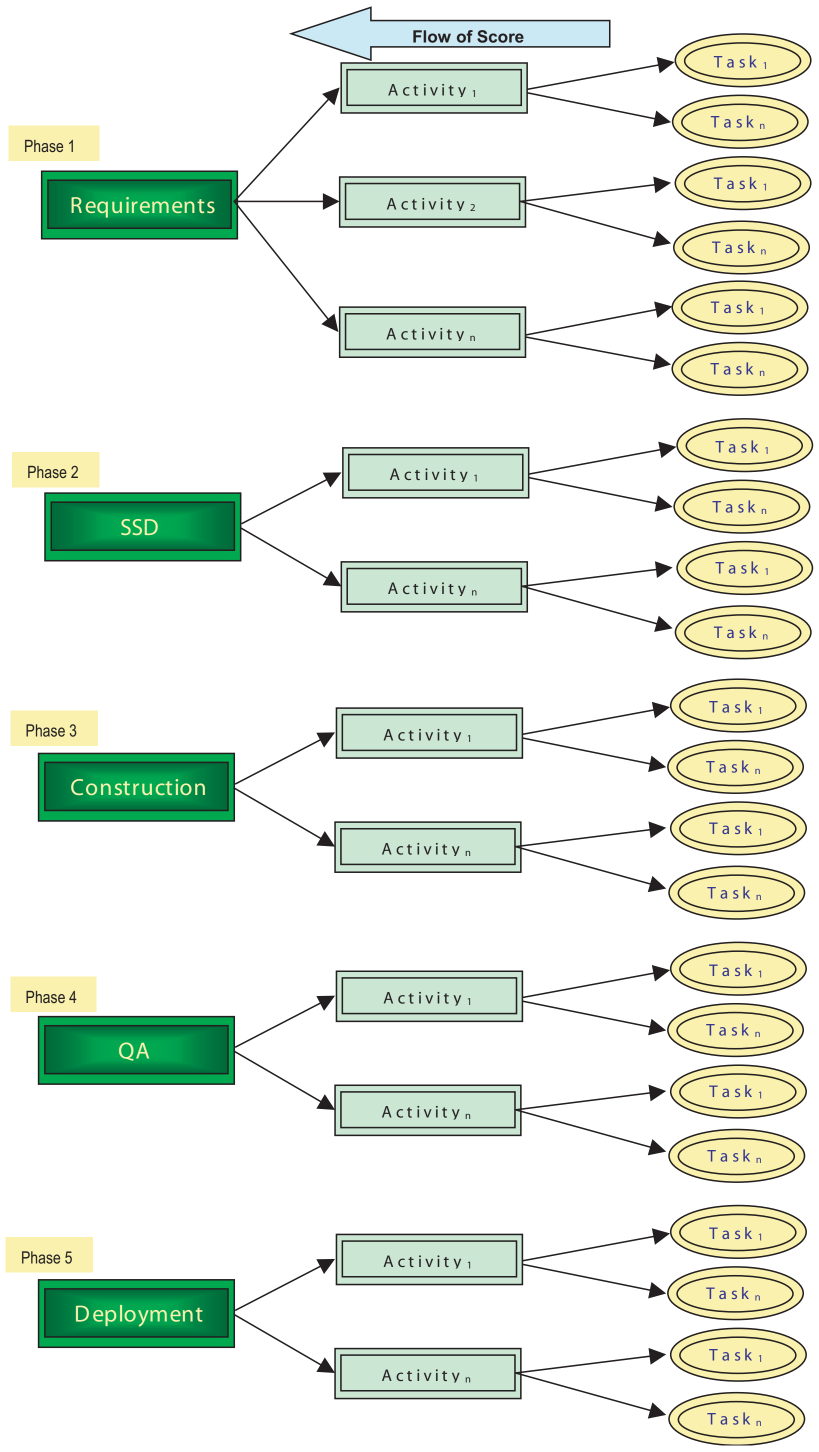
A scale of 1 to 10 is defined for each of the tasks. In case a particular task is not applicable to the activity, the Not Applicable (N/A) option should be selected. A maximum value of 10 should be selected as the score for a task when the evaluation criteria and the completion expediency are fully met. When no defined process is followed a value of zero should be selected.

As illustrated in Figure 3, the scores for tasks under the activity 'Business Requirements' is 5, 5, 6, N/A, 2 and 2. The score for this activity is calculated as explained below:

Score of each activity = Scores [(Task 1...Task n)/(n*10)]*100

Score of business requirements = [(5+5+6+2+2)/(5*10)]*100 = 40%

Chp4.qxd 3/10/2008 11:43 PM Page 5



Definitions:

Activity_1 = 1st Activity
Activity_2 = 2nd Activity
Activity_n = nth Activity
Task_1 = 1st Task
Task_2 = 2nd Task
Task_n = nth Task
SSD = System Specification and Design

SDLC Process Adoption Score calculations:

Process adoption = Function of (Phase 1... Phase 5)

Each phase = Function of (Activity 1 … Activity n)

Each activity = Function of (Task 1...Task n)

Score of each activity = ΣScores ((Task 1...Task n)/(n*10))*100

Score of each phase = Σ Scores (Activity 1 … Activity n)/n

Process adoption score of the project = $$\frac{\sum_{i=1}^{n} \text{Scores}(T_i)}{\sum_{i=1}^{n} T_i - \sum_{i=1}^{n} T_{n/ai}} * 10$$

where $\sum_{i=1}^{n} \text{Scores}(T_i)$ = *Summation of scores of all tasks*

$\sum_{i=1}^{n} T_i$ = *Summation of all tasks*

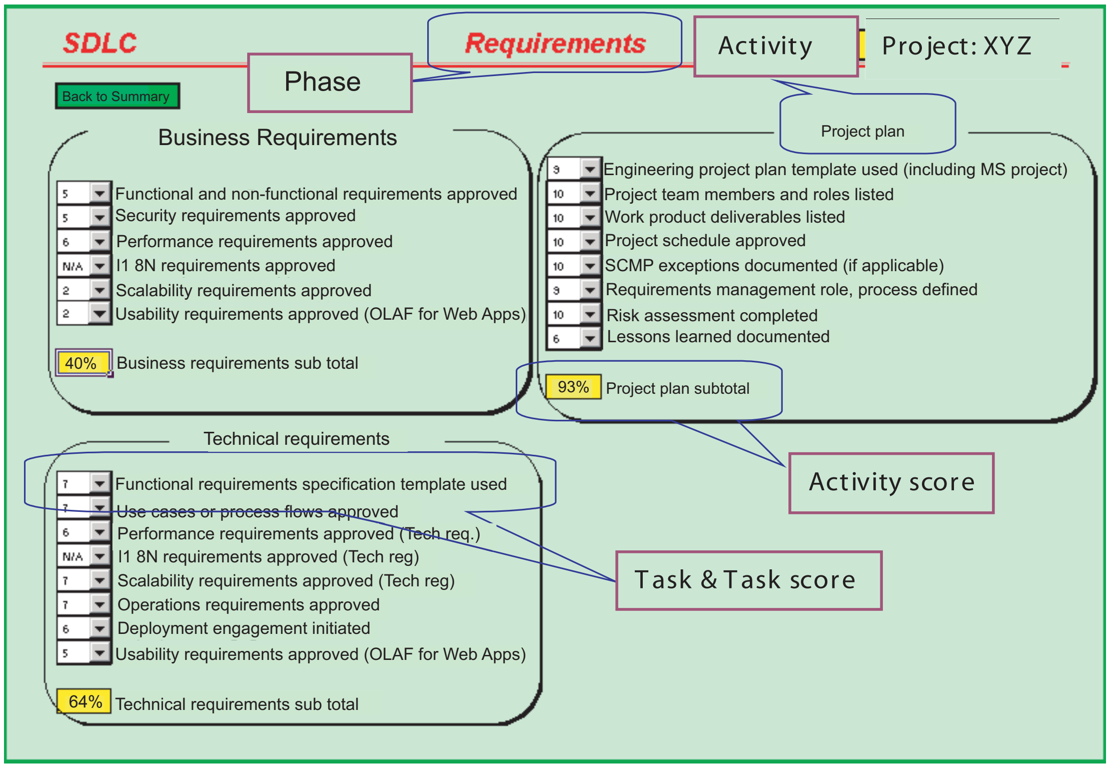
$\sum^{n} T_{n/ai}$ = *Summation of N/A tasks*

Figure 2 Process flow chart and calculatins

Table1 shows a sample set of activities and tasks for the other four phases such as System Specification and Design, Construction, Quality Assurance and Deployment.

Final score of the project

Figure 4 is the example scorecard for a project 'XYZ'. The tool will calculate the overall score for a project based on the phase-wise score. For example, for project 'XYZ'



SDLC Requirements Project: XYZ

Project plan

9	Engineering project plan template used (including MS project)
10	Project team members and roles listed
10	Work product deliverables listed
10	Project schedule approved
10	SCMP exceptions documented (if applicable)
9	Requirements management role, process defined
10	Risk assessment completed
6	Lessons learned documented
93%	Project plan subtotal

Business Requirements

5	Functional and non-functional requirements approved
5	Security requirements approved
6	Performance requirements approved
N/A	I1 8N requirements approved
2	Scalability requirements approved
2	Usability requirements approved (OLAF for Web Apps)
40%	Business requirements sub total

Technical requirements

7	Functional requirements specification template used
7	Use cases or process flows approved
6	Performance requirements approved (Tech req.)
N/A	I1 8N requirements approved (Tech reg)
7	Scalability requirements approved (Tech reg)
7	Operations requirements approved
6	Deployment engagement initiated
5	Usability requirements approved (OLAF for Web Apps)
64%	Technical requirements sub total

Figure 3 Sample requirements phase score details

the score is 74 per cent, i.e, project XYZ has adopted only 74 percent of the organisation-wide process. This score is the measure of the process adoption of the project 'XYZ'.

The score of 74 per cent is derived using the equation as below:

$$\text{Process adoption score of the project} = \frac{\sum_{i=1}^{n} \text{Scores}(T_i)}{\sum_{i=1}^{n} T_i - \sum_{i=1}^{n} T_{n/ai}} *10$$

Figure 4 illustrates the SDLC process adoption scorecard for project XYZ.

Chp4.qxd 3/10/2008 11:43 PM Page 8

Table 1 A sample set of activities and tasks

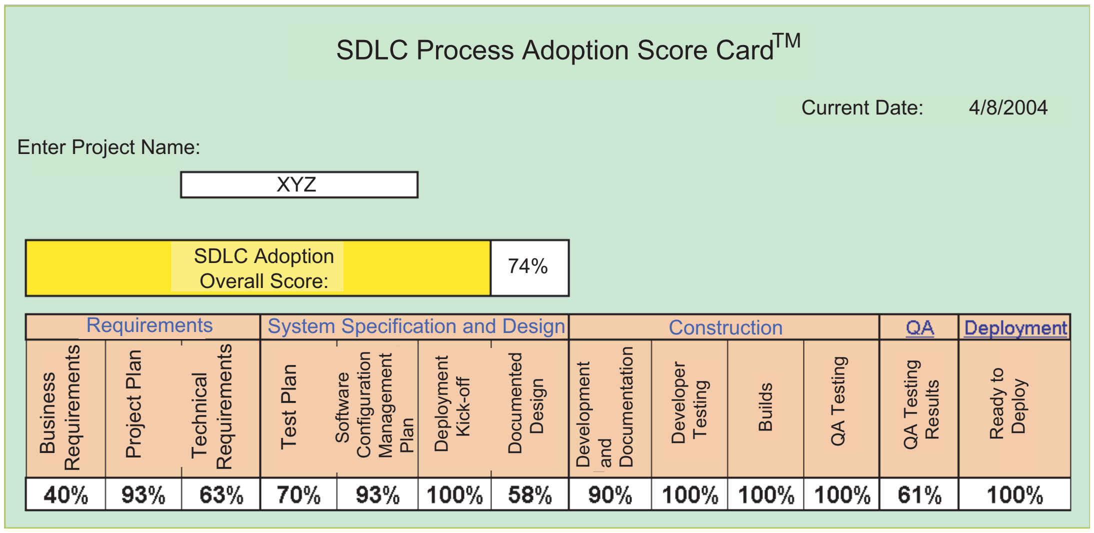
Phases	Activities	Tasks
System Specification and Design	1. Test plan 2. Software configuration Management plan 3. Deployment kick-off 4. Documented design	1. Test description document template used 2. Standard SCM plan followed 3. Deployment iss ues addressed 4. Software design document template used
Construction	1. Development and documentation 2. Developer testing 3. Builds 4. QA testing	1. Coding completed with comments 2. Unit test defined 3. Builds created through SCM tool 4. Change requests addressed within the build 5. System integration testing completed
Quality Assurance	1. QA testing results	1. Coding standard adherence 2. QA test metrics reported to Engineering Project Manager 3. System test res ults reported
Deployment	1. Ready to deploy	1. Deployment readiness review completed

BENEFITS OF SDLC PROCESS ADOPTION SCORECARD

The advantages of using the SDLC process adoption scorecard could be divided into the following categories:

- High visibility to senior management
- Brings a sense of ownership to those driving the project
- Relative ranking across projects
- Ease of use
- Consistent method for scoring various project activities.

Chp4.qxd 3/10/2008 11:43 PM Page 9


SDLC Process Adoption Score Card^{TM}

Current Date: 4/8/2004

Enter Project Name: XYZ

SDLC Adoption Overall Score: 74%

Requirements			System Specification and Design				Construction				QA	Deployment
Business Requirements	Project Plan	Technical Requirements	Test Plan	Software Configuration Management Plan	Deployment Kick-off	Documented Design	Development and Documentation	Developer Testing	Builds	QA Testing	QA Testing Results	Ready to Deploy
40%	93%	63%	70%	93%	100%	58%	90%	100%	100%	100%	61%	100%

Figure 4 A summarized view of the SDLC Process Adoption Scorecard

High visibility for senior management

The key to process management when there is a portfolio of products handled by multiple engineering teams is to have a means of visualising the adoption rate. This could be seemed as dashboard, digital cockpit or simply, scorecard. There can be quality reviews every month in which senior leadership participates and gets information from all project managers using the scorecard. During the reviews, senior leadership has an opportunity to get an understanding of how a particular process was implemented and why a particular process was not followed. On the other side, project managers have an opportunity to represent the project's capability in terms of process adoption by using the scorecard. A comparison for the same project across various releases helps the management decide on the areas that need improvement or areas that need to get under control.

Brings in a sense of ownership to those driving the project

The key here is that the project driver becomes the process driver if the scorecard is used as a benchmark to measure process adoption across various projects. Project managers have the responsibility to maintain scorecard results and ensure that necessary steps are taken to adopt the processes defined by an organisation.

Relative ranking across projects

There is positive competition built across various projects to make them adhere to the processes and enable the projects to align with the organisation, process definition. Every project team could be measured using the same yardstick and it will help project teams to identify the weak areas and quickly emerge out of the weakness by taking necessary actions. The organisation could instate a recognition system based on the process adoption score.

Ease of use

The scorecard is easy to use for anyone who is managing the project from the requirements stage till the release or deployment stage. It is built on various questions pertaining to a particular phase of the project and contains a Likert scale to answer the questions. There is no overhead in getting the scorecard implemented.

Consistent method for scoring various project activities

A common process understanding is built across the various project teams through the scorecard. The management, in effect, has less scope for misinterpretation of the results.

CONCLUSION

Success for a software organisation lies not only in meeting the deliverable dates but also in having a good process measurement capability. It becomes more important for organisations with a portfolio of products with each product consisting of various project teams to manage the software engineering process through the simple means of a scorecard. Global eXchange Services, as an organisation, has always been a keen driver for process adoption in order to ensure standardisation and easy measurement criteria across various projects. The score defined for the tasks is subjective and needs to be periodically audited in order to ensure that the score depicts the actual state of the process followed.

Acronyms

QA	– Quality Assurance
SCM	– Software Configuration Management

SEI	– Software Engineering Institute
SDLC	– Software Development Lifecycle
SM-CMM	– Capability Maturity Model for Software.

References

1. http://www.ecbp.org/glossary.htm issued by the International Association for Professional Management of Construction.

2. http://www.teraquest.com/SW-CMM/static/Model_ProcessArea_Organization%20Process%20Definition.html.