Software Engineering :

10 : Software Architecture Part 3

Professor Imed Bouchrika

National School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

- **Part 1:**
  - Introduction to Software Architecture
  - Software Architecture Process
    - Selecting Technological Stack
  - Architecture Quality Attributes
  - Software Architectural Styles
- **Part 2 :**
  - Programming Questions
  - Design Patterns
  - Architecture Principles : SOLID
- **Part 3 :**
  - Revision : Design Patterns
  - Structured Design : History
  - Decoupling & Cohesion
  - Documenting the Architecture

Best Practices : SOLID

- **SOLID Practices**
  - S : Single Responsibility Principle
  - O : Open/Closed Principles
  - L : Liskov Substitution Principle
  - I : Interface Segregation Principle
  - D : Dependency Inversion Principle

Software Engineering : First Week

- This course is not programming, but the use of best practices in software engineering to develop a software product meeting the quality attributes.
- The aim is not code/implement the functional requirements completed, but to have :
  - Maintainable Software
  - Extensible
  - Scalable…
- Remember, there is no “simple software” or “simple/easy code”.

Case Study 0 : Show Hello World

- **Example from the first Lecture :**

Hello.java

```
1  class Hello{
2
3      public static void main(String [] args){
4          System.out.println("Hello World ! ");
5      }
6  }
7
```

Revision : Architecture Styles

- **Case 1 : Hello World Specification :**
  - Print a Given message that can be “Hello World” but there will be text processing being applied depending on the region:
    - Users of Algiers : would be shown as Capital letters.
    - Users of Annaba, letters will be duplicated.
    - Users of Oran, letters would be reversed.
    - **Other regions, we don’t have yet the details, will be available in the future.**
  - For simplicity, users would be asked the region code (16, …)

Revision : Architecture Styles

- **Case 2 : Hello World Specification :**
  - Print a Given message that can be “Hello World” but there will be a **sequence of** text processing operations being applied before printing:
    - Trimming the text.
    - Converting text to Capital letters.
    - Removing ponctuations.
    - Replacing Spaces with a hyphen (-)
    - Append a number at the end reflecting the number of times run
    - Other operations…..(we don’t know…)

Revision : Architecture Styles

- **Case 3 : Hello World Specification :**
  - Print a Given message that can be “Hello World” and inform users based on their preferences that the user printed the hello word.
    - Users can be notified by SMS or Email.

Revision : Architecture Styles

Not how to code, but What

Software architecture to use ?

- **Case 1 : Hello World Specification :**
  - Print a Given message that can be “Hell
    processing being applied depending on the region:
    - Users of Algiers : would be shown as Capital letters.
    - Users of Annaba, letters will be duplicated.
    - Users of Oran, letters would be reversed.
    - **Other regions, we don’t have yet the details, will be available in the future.**
  - For simplicity, users would be asked what the region code (16, …)

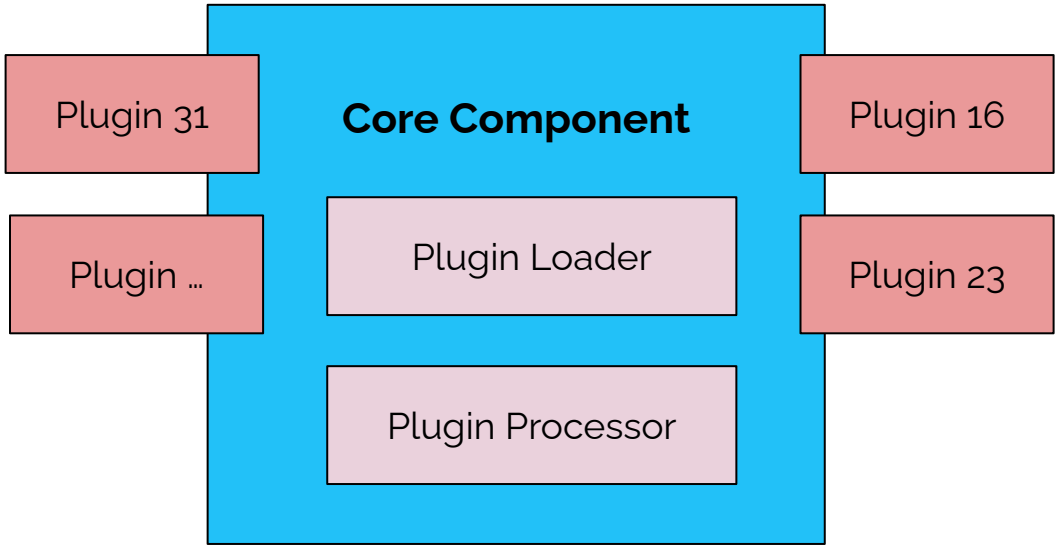
Revision : Architecture Styles

Best to use :

Microkernel Architecture

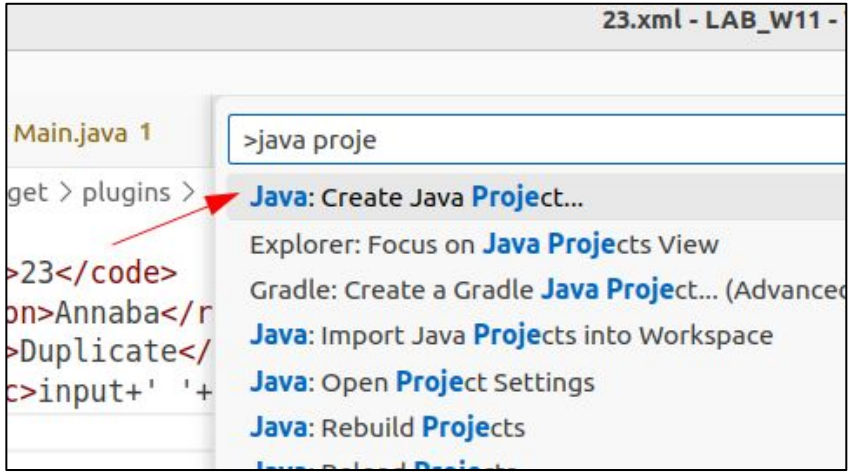
- **Case 1 : Hello World Specification :**
  - Print a Given message that can be “Hell
    processing being applied depending on the region:
    - Users of Algiers : would be shown as Capital letters.
    - Users of Annaba, letters will be duplicated.
    - Users of Oran, letters would be reversed.
    - **Other regions, we don’t have yet the details, will be available in the future.**
  - For simplicity, users would be asked what the region code (16, …)

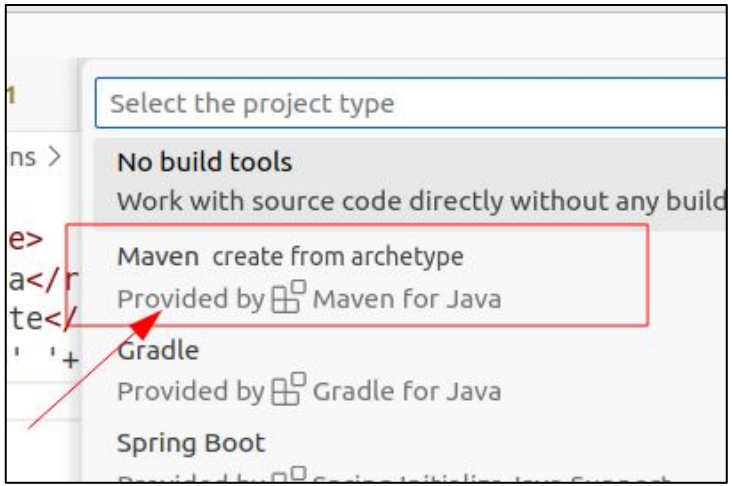
Revision : Architecture Styles



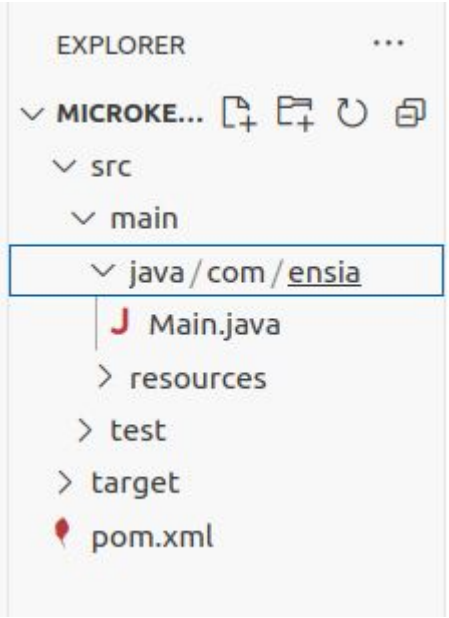
Revision : Architecture Styles

- **Case 1 : Hello World Specification :**
  - Create a New Java Project inside your VS.
    - Choose **Maven** as the Building utility





Revision : Architecture Styles



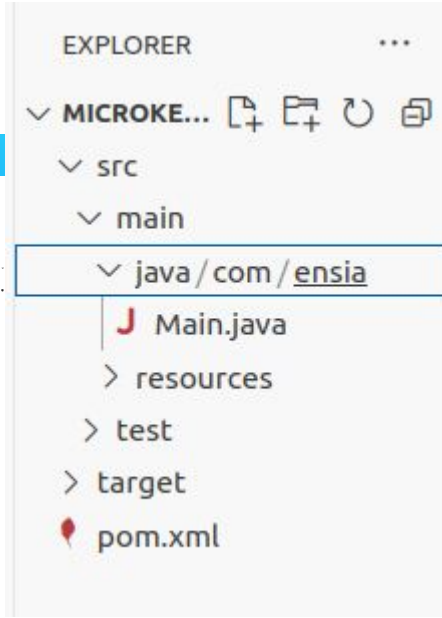
- **Case 1 : Hello World Specification :**
  - Create a New Java Project inside your VS.
    - Choose **Maven** as the Building utility :

      Maven / Ant / CMake / Gradle ...:

      - Are Build Automation tools with various features to facilitate compiling, building, packaging, testing and even installing software systems and components

Revision : Architecture Styles

- **Case 1 : Hello World Specification :**
  - **Compiling and Packaging : maven**
    - Install maven on your windows or Ubuntu machine
    - Structure your project in the following structure :
      - Under : src/main/java place your java files
      - The pom.xml contains the maven config for compiling and building



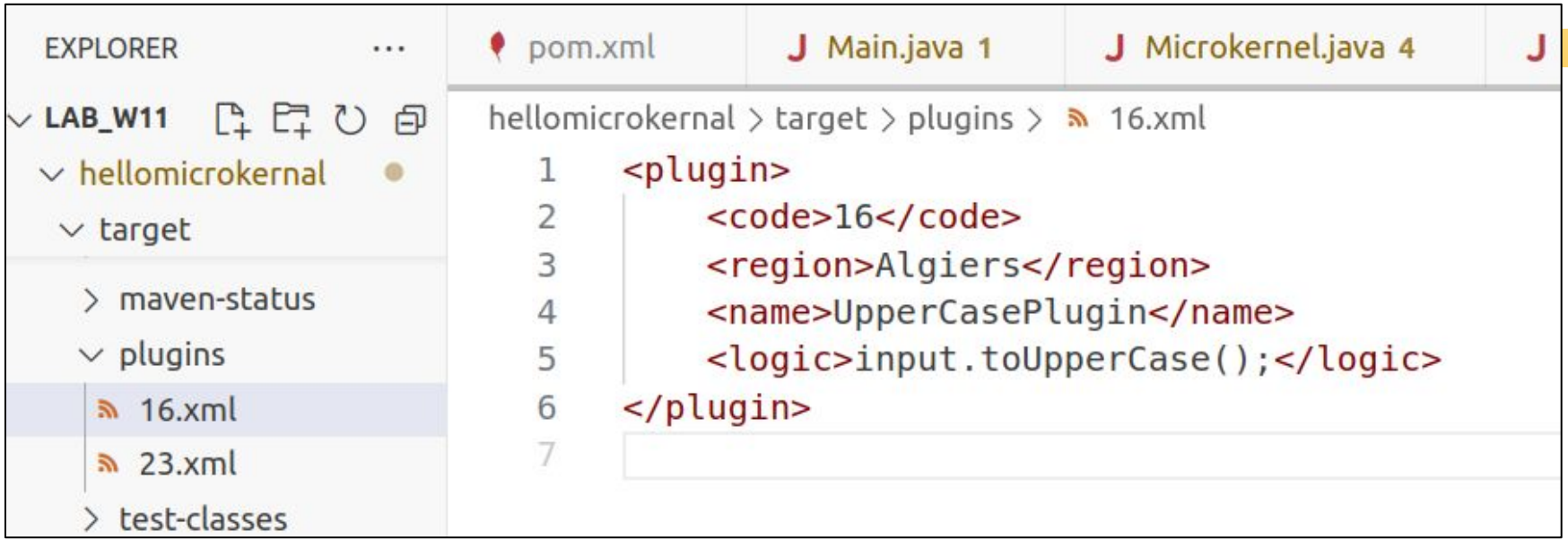
Revision : Architecture Styles

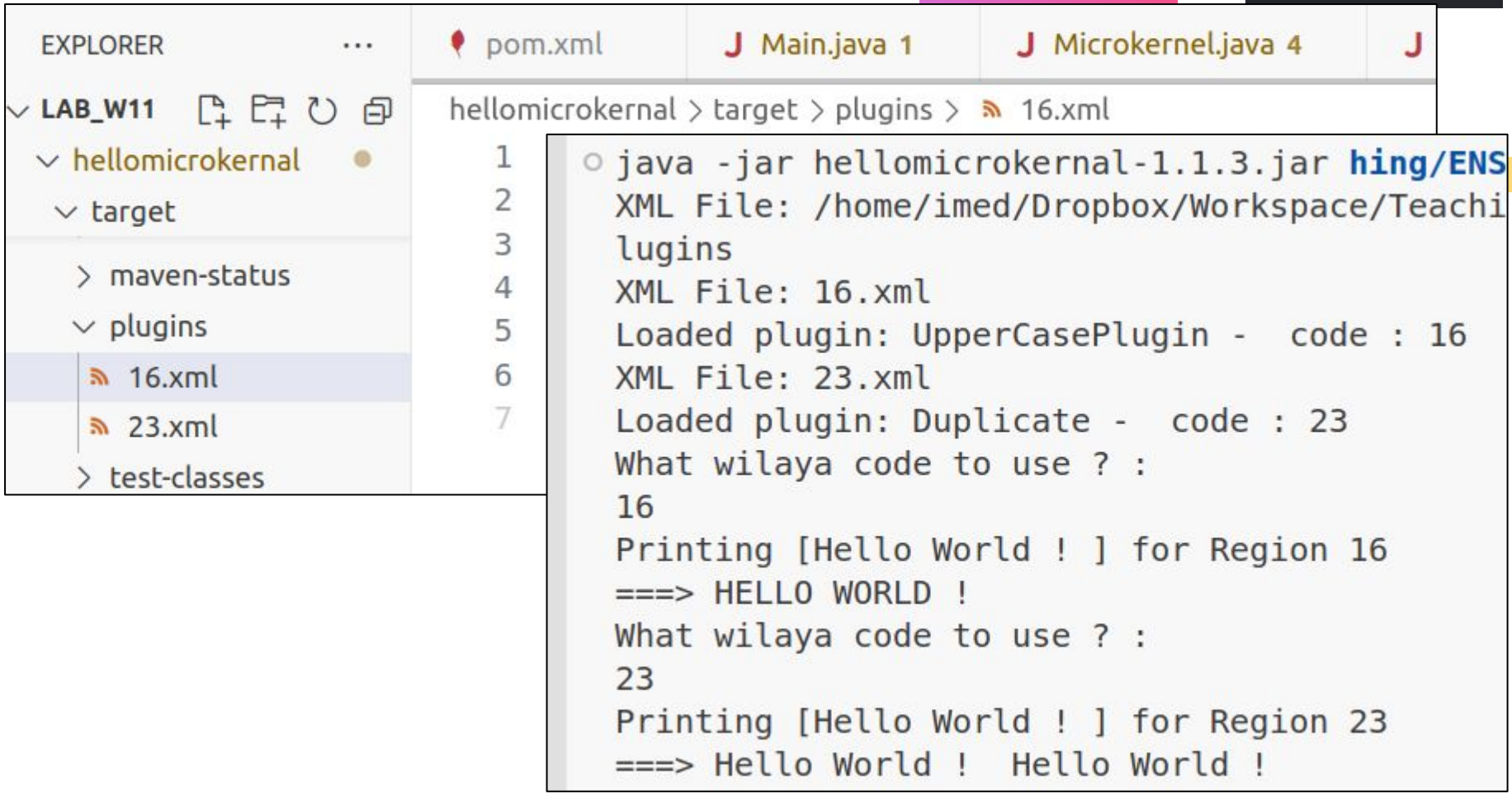
- **Compiling and Packaging : pom.xml file**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>dev.startsoftware</groupId>
   <groupId>com.ensia</groupId>
   <artifactId>hellomicrokernal</artifactId>
   <version>1.1.3</version>
   <packaging>jar</packaging>
   <properties>
       <maven.compiler.source>17</maven.compiler.source>
       <maven.compiler.target>17</maven.compiler.target>
   </properties>
</project>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ensia</groupId>
    <artifactId>hellomicrokernal</artifactId>
    <version>1.1.3</version>
    <packaging>jar</packaging>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
    <build>
      <plugins>
        <plugin>
          <!-- Build an executable JAR -->
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.1.0</version>
          <configuration>
            <archive>
              <manifest>
                <classpathPrefix>lib/</classpathPrefix>
                <addClasspath>true</addClasspath>
                <mainClass>com.ensia.Main</mainClass>
              </manifest>
            </archive>
          </configuration>
        </plugin>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <executions>
                    <execution>
                        <id>copy</id>
                        <phase>package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                        <configuration>
                            <outputDirectory>
                                ${project.build.directory}/lib
                            </outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>

      </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>org.mozilla</groupId>
            <artifactId>rhino</artifactId>
            <version>1.7.14</version>
        </dependency>
    </dependencies>
</project>
```

```java
package com.ensia;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner;
        Microkernel microkernel = new Microkernel();
        microkernel.loadPlugins("plugins");
        while (true) {
            try {
                String message = "Hello World ! ";
                System.out.println("What wilaya code to use ? : " );
                scanner = new Scanner(System.in);
                String region = scanner.nextLine().trim();

                System.out.println("Printing ["+message+"] for Region "+region);
                String new_message = microkernel.processText(region, message);
                System.out.println("===> "+new_message);
            } catch (Exception e) {
                System.out.println("Error...."+e.getStackTrace());
            }
        }
    }
}
```


EXPLORER
pom.xml
Main.java 1
Microkernel.java 4
LAB_W11
hellomicrokernal
target
maven-status
plugins
16.xml
23.xml
test-classes
hellomicrokernal > target > plugins > 16.xml
1 <plugin>
2 <code>16</code>
3 <region>Algiers</region>
4 <name>UpperCasePlugin</name>
5 <logic>input.toUpperCase();</logic>
6 </plugin>
7


EXPLORER
pom.xml
Main.java 1
Microkernel.java 4
LAB_W11
hellomicrokernal
target
maven-status
plugins
16.xml
23.xml
test-classes
hellomicrokernal > target > plugins > 16.xml
1
2
3
4
5
6
7
java -jar hellomicrokernal-1.1.3.jar hing/ENS
XML File: /home/imed/Dropbox/Workspace/Teachi
lugins
XML File: 16.xml
Loaded plugin: UpperCasePlugin - code : 16
XML File: 23.xml
Loaded plugin: Duplicate - code : 23
What wilaya code to use ? :
16
Printing [Hello World !] for Region 16
===> HELLO WORLD !
What wilaya code to use ? :
23
Printing [Hello World !] for Region 23
===> Hello World ! Hello World !

Revision : Architecture Styles

- **Case 2 : Hello World Specification :**
  - Print a Given message that can be “Hello World” but there will be a **sequence of** text processing operations being applied before printing:
    - Trimming the text.
    - Converting text to Capital letters.
    - Removing ponctuations.
    - Replacing Spaces with a hyphen (-)
    - Append a number at the end reflecting the number of times run
    - Other operations…..(we don’t know…)

Revision : Architecture Styles

- **Architecture :**

Modularization and encapsulation inside a class

Facade Pattern is used where a single facade method is provided calling all other private functions

Hello.java | HelloMessage.java

```
1
2  class HelloMessage{
3
4      private String messageToPrint;
5
6      public HelloMessage(String msg){
7          this.messageToPrint=msg;
8      }
9
10     public String getMessageToPrint(){
11         String message=trimMessage(messageToPrint);
12         //more business logic can be applied.
13         return message;
14     }
15     public static String trimMessage(String msg){
16         if (msg==null)return "";
17
18         String message=msg.trim();
19         return message;
20     }
21 }
```

Hello.java | HelloMessage.java

```
1  class Hello{
2
3      public static void main(String [] args){
4          HelloMessage helloObj=new HelloMessage("Hello World ! ");
5
6          System.out.println(helloObj.getMessageToPrint());
7      }
8  }
```

Revision : Archite

- Architec

```java
interface IHelloMessage{
    public String getMessageToPrint();
}
```

```java
class HelloMessage implements IHelloMessage{

    private String messageToPrint;

    public HelloMessage(String msg){
        this.messageToPrint=msg;
```

Hello.java | HelloMessage.java

```java
class HelloMessage{
```

```
{
(){
        String message=trimMessage(messageToPrint);
e applied.
e(String msg){
    }
}
```

Hello.java | HelloMessage.java

```java
class Hello{

    public static void main(String [] args){
        HelloMessage helloObj=new HelloMessage("Hello World ! ");

        System.out.println(helloObj.getMessageToPrint());
    }
}
```

We work always with Abstraction, provide an interface with public method

- Architec

```
interface IHelloMessage{
    public String getMessageToPrint();
}
```

```
class HelloMessage implements IHelloMessage{

    private String messageToPrint;

    public HelloMessage(String msg){
        this.messageToPrint=msg;
```

Hello.java | HelloMessage.java

```
class HelloMessage{
```

```
{
```

```
(){
        String message=trimMessage(messageToPrint);
e applied.

e(String msg){
```

```
    }
}
```

Hello.java | HelloMessage.java

```
class Hello{

        HelloMessage helloObj=new HelloMessage("Hello World ! ");

        System.out.println(helloObj.getMessageToPrint());
    }
}
```

There is some glue here…what if we want to change the concretion,,,,

Factory Pattern method is created

Inside the main method, we don't need to know how it is created nor its actual real class.

```
1
2
3   class HelloMessageFactory{
4
5       public static IHelloMessage createHelloMessage(String type){
6           if (type==null || type.equalsIgnoreCase("normal")){
7               IHelloMessage ret_object=new HelloMessage("Hello World ! ");
8               return ret_object;
9           }
10          return null;
11      }
12
13  }
14
```

```
2   class Hello{
3
4       public static void main(String [] args){
5           IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
6           System.out.println(helloObj.getMessageToPrint());
7       }
8   }
```

Extensibility ? easy now

```
3  class HelloMessageFactory{
4
5      public static IHelloMessage createHelloMessage(String type){
6          if (type==null || type.equalsIgnoreCase("normal")){
7              IHelloMessage ret_object=new HelloMessage("Hello World ! ");
8              return ret_object;
9          }
10         if (type.equalsIgnoreCase("custom")){
11             //some logic here..
12         }
13         if (type.equalsIgnoreCase("translated")){
14             //some logic here..
15         }
16         if (type.equalsIgnoreCase("counter")){
17             //some logic here..
18         }
19         return null;
20     }
21
22 }
```

- **Architecture :**

Different Levels of Abstraction :

Creating some complex object vs. printing to the console ?

Let's apply more modularisation:

Creation + Display :

<u>which pattern ?</u>

```
2  class Hello{
3
4      public static void main(String [] args){
5          IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
6          System.out.println(helloObj.getMessageToPrint());
7      }
8  }
```

- **Architecture :**

Different Levels of Abstraction :

Creating some complex object vs. printing to the console ?

Let's apply more modularisation:

Creation + Display :

<u>which pattern ?</u>

```
2  class Hello{
3
4      public static void main(String [] args){
5          IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
6          System.out.println(helloObj.getMessageToPrint());
7      }
8  }
```

```java
interface IDisplay{

    public void printMessage(String format);

}
```

```java
class Display implements IDisplay{

    private String message;
    public Display(String msg){
        this.message=msg;
    }
    public void printMessage(String format){
        if (format==null || format.equalsIgnoreCase("console")){
            System.out.println(message);
            return;
        }
    }
}
```

Extensibility ? Other formats ? Gui ? PDF

```
class Display implements IDisplay{

    private String message;
    public Display(String msg){
        this.message=msg;
    }
    public void printMessage(String format){
        if (format==null || format.equalsIgnoreCase("console")){
            System.out.println(message);
            return;
        }
    }
}
```

```
interface IDisplay{

    public void printMessage(String format);

}
```

```
interface IDisplayGUI{

    public void showMessageOnGUI(String msg);

}
```

Adapter Pattern

```
import javax.swing.*;
import java.awt.*;

class DisplayGUI implements IDisplayGUI{

    public void showMessageOnGUI(String message){
        JFrame frame = new JFrame(message);

        frame.setMinimumSize(new Dimension(800, 600));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel lb_msg = new JLabel(message, SwingConstants.CENTER);

        frame.getContentPane().add(lb_msg);

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
class Display implements IDisplay{

    private String message;
    public Display(String msg){
        this.message=msg;
    }
    public void printMessage(String format){
        if (format==null || format.equalsIgnoreCase("console")){
            System.out.println(message);
            return;
        }

        if (format.equalsIgnoreCase("gui")){
            IDisplayGUI gui=new DisplayGUI();
            gui.showMessageOnGUI(message);
        }
        if (format.equalsIgnoreCase("pdf")){
            //some business logic for the PDF file...
        }
    }

```

```
class Display implements IDisplay{

    private String message;
    public Display(String msg){
        this.message=msg;
    }
    public void printMessage(String format){
        if (format==null || format.equalsIgnoreCase("console")){
            System.out.println(m
            return;
        }
    }
}
```

```
interface IDisplay{

    public void printMessage(String format);

}
```

```
interface IDisplayGUI{
```

```
class Hello{

    public static void main(String [] args){
        IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
        String message=helloObj.getMessageToPrint();

        IDisplay diplay=new Display(message);
        String format="console";
        if (args.length > 0){
            format=args[0];
        }
        diplay.printMessage(format);
    }
}
```

Adapter Pattern

```
import javax.swing.*;
import java.awt.*;

class DisplayGUI implements IDisp

    public void showMessageOnGUI(String message){
        JFrame frame = new JFrame(message);

        frame.setMinimumSize(new Dimension(800, 600));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel lb_msg = new JLabel(message, SwingConstants.CENTER);

        frame.getContentPane().add(lb_msg);

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
        }

        if (format.equalsIgnoreCase("gui")){
            IDisplayGUI gui=new DisplayGUI();
            gui.showMessageOnGUI(message);
        }
        if (format.equalsIgnoreCase("pdf")){
            //some business logic for the PDF file...
        }
    }
```

```
class Display implements IDisplay{

    private String message;
    public Display(String msg){
        this.message=msg;
    }
    public void printMessage(String format){
        if (format==null || format.equalsIgnoreCase("console")){
            System.out.println(m
            return;
        }
    }
}
```

```
interface IDisplay{

    public void printMessage(String format);

}
```

```
interface IDisplayGUI{
```

```
class Hello{

    public static void main(String [] args){
        IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
        String message=helloObj.getMessageToPrint();

        IDisplay diplay=new Display(message);

        String format="console";
        if (args.length > 0){
            format=args[0];
        }
        diplay.printMessage(format);
    }
}
```

This is a mess ?

```
import javax.swing.*;
import java.awt.*;

class DisplayGUI implements IDisp

    public void showMessageOnGUI(String message){
        JFrame frame = new JFrame(message);

        frame.setMinimumSize(new Dimension(800, 600));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel lb_msg = new JLabel(message, SwingConstants.CENTER);

        frame.getContentPane().add(lb_msg);

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
        }

        if (format.equalsIgnoreCase("gui")){
            IDisplayGUI gui=new DisplayGUI();
            gui.showMessageOnGUI(message);
        }
        if (format.equalsIgnoreCase("pdf")){
            //some business logic for the PDF file...
        }
    }
```

We create another module to parse command line arguments

Who knows, in case of a change in the future, it won't impact all modules

Ex, program -o gui -d 900x600 ?

```
class Display implements IDisplay{

    private String message;
    public Display(String msg){
        this.message=msg;
    }
```

```
interface IDisplay{

    public void printMessage(String format);

}
```

```
interface IDisplayGUI{
```

```
class Hello{

    public static void main(String [] args){
        IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
        String message=helloObj.getMessageToPrint();

        IDisplay diplay=new Display(message);
        String format="console";
        if (args.length > 0){
            format=args[0];
        }
        diplay.printMessage(format);
    }
}
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel lb_msg = new JLabel(message, SwingConstants.CENTER);

        frame.getContentPane().add(lb_msg);

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
        }

        if (format.equalsIgnoreCase("gui")){
            IDisplayGUI gui=new DisplayGUI();
            gui.showMessageOnGUI(message);
        }
        if (format.equalsIgnoreCase("pdf")){
            //some business logic for the PDF file...
        }
    }
```

Revision : Architecture Styles

- **Using the principles of Software Engineering :**
  - Any feature to be added, it is easy.
  - The modules are fully reusable. I can use the Display Module with other software.
  - The code is very readable and can be maintained with ease.
  - New Feature ?

Revision : Architecture Styles

- **Case 2 : Hello World Specification :**
  - Print a Given message that can be “Hello World” but there will be a **sequence of** text processing operations being applied before printing:
    - Trimming the text.
    - Converting text to Capital letters.
    - Removing ponctuations.
    - Replacing Spaces with a hyphen (-)
    - Append a number at the end reflecting the number of times run
    - Other operations…..(we don’t know…)

Revision : Architecture Styles

- **Case 2 : Hello World Specification :**
  - Print a Given message that can be “Hello World” but there will be a **sequence of** text processing operations being applied before printing:
    - Trimming the text.
    - Converting text to Capital letters.
    - Removing ponctuations.
    - Replacing Spaces with a hyphen (-)
    - Append a number at the end reflecting the number of times run
    - Other operations…..(we don’t know…)

Doable ?

Which design Pattern ?

Case Study 1 : Show Hello World

- Invoking a number of action i

Command Pattern

```
interface IPostprocessingCommand{

    public String executePostprocessing(String message);
}
```

```
class MessageConvertUpper implements IPostprocessingCommand{

    public String executePostprocessing(String message){
        if (message==null) return "";
        String postMessage=message.toUpperCase();
        return postMessage;
    }

}
```

```
class MessageRemovePunctuation implements IPostprocessingCommand{

    public String executePostprocessing(String message){
        if (message==null) return "";
        String postMessage=message.replaceAll("\\p{Punct}", "");
        return postMessage;
    }

}
```

Case Study 1 : Show Hello World

- Invoking a number of action i

```
interface IPostprocessingCommand{

    public String executePostprocessing(String message);
}
```

```
class MessageConvertUpper implements IPostprocessingCommand{

g executePostprocessing(String message){
age==null) return "";
ostMessage=message.toUpperCase();
ostMessage;
```

```
vePunctuation implements IPostprocessingCommand{

 executePostprocessing(String message){
ge==null) return "";
stMessage=message.replaceAll("\\p{Punct}", "");
stMessage;
```

```
import java.util.*;

class Hello{

    public static void main(String [] args){
        IHelloMessage helloObj=HelloMessageFactory.createHelloMessage("normal");
        String message=helloObj.getMessageToPrint();

        Vector <IPostprocessingCommand>commands=new Vector();
        commands.add(new MessageConvertUpper());
        commands.add(new MessageRemovePunctuation());
        //.. More can be added....
        for (int i=0;i<commands.size();i++){
            IPostprocessingCommand command=commands.get(i);
            message=command.executePostprocessing(message);
        }

        IDisplay diplay=new Display(message);
        String format="console";
        if (args.length > 0){
            format=args[0];
        }
        diplay.printMessage(format);
    }
}
```

Structured Design

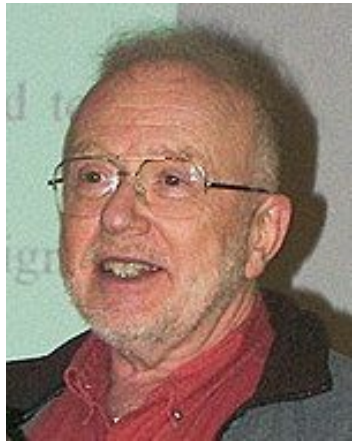
- **Software Crisis :**
  - a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time due to
    - Existing methods were inadequate.
    - Complexity of software systems.
    - The systems became very complicated to maintain.
  - The term "software crisis" was coined by some attendees at the first NATO Software Engineering Conference in **1968** in Germany

Structured Design

- **From Structured Programming and Modular Programming**
  - **Structured programming** (Use of Subroutines + Control flow blocks) began in 1950s.
  - **Modular programming** started in 1960s where each functionality or part of the system is implemented as a separate ***"independent?"*** module with an **interface**.
  - The aim of **modularization** is to slow down of what's called **software entropy**, before the system becomes unmanageable due to its complexity.
    - Software entropy = the risk that changing existing software will result in unexpected problems, unmet objectives, or total failure.

Structured Design

- **From Structured Programming and Modular Programming**
  - “David Lorge Parnas (born February 10, 1941) is a Canadian early pioneer of software engineering, who developed the concept of information hiding in modular programming, which is an important element of object-oriented programming today. He is also noted for his advocacy of precise documentation.”



Structured Design

- **From Structured Programming and Modular Programming**
  - What's a module : depending on at which level of abstraction you look from :
    - Sub-system
    - Plugin
    - Component
    - Package
    - Service
    - Class
    - Or even a function

Structured Design

- **From Structured Programming and Modular Programming**
  - What's a module : depending on at which level of abstraction you look from :
    - Sub-system
    - Plugin
    - Component
    - Package
    - Service
    - Class
    - Or even a function

Any entity at any level as long as it has a boundary which isolates or hides its inside from the outside via interface(s)

Interface here is not the interface as in Java/C++..

Structured Design

- **Interface : Linking Modules**
  - Linking modules is done via via an interface :

    “Any such **referenced element** defines an interface, a portion of the module boundary **across which data or control flow**. […] you may think of it as a socket into which the plug, represented by the connection from the referencing module, is inserted. Every interface in a module represents one more thing which is/must be known, understood, and properly connected by other modules in the system.” Taken from the Structured Design Book.

Structured Design

- **Interface : Linking Modules**
  - Linking modules is done via via an interface :

    “Any such referenced element defines an interface, a portion of the module boundary across which data or
    which the plug, represented by
    inserted. Every interface in a mo
    be known, understood, and pro
    Taken from the Structured Desi

A public method of a class can be considered one of its interface.

Or a public static/instance variable

Or a network socket

Or API Endpoint …

Structured Design

- **From Structured Design to Software Architecture**
  - Structured Design : high level of how divide a system into components /modules and connect them together (1974)
  - This is what's known today as software architecture

Structured design

by W. P. Stevens, G. J. Myers, and L. L. Constantine

Structured design is a set of proposed general program design considerations and techniques for making coding, debugging, and modification easier, faster, and less expensive by reducing complexity.[1] The major ideas are the result of nearly ten years of research by Mr. Constantine.[2] His results are presented here, but the authors do not intend to present the theory and derivation of the results in this paper. These ideas have been called *composite design* by Mr. Myers.[3-5] The authors believe these program *design* techniques are compatible with, and enhance, the *documentation* techniques of HIPO[6] and the *coding* techniques of structured programming.[7]

These cost-saving techniques always need to be balanced with other constraints on the system. But the ability to produce simple, changeable programs will become increasingly important as the cost of the programmer's time continues to rise.

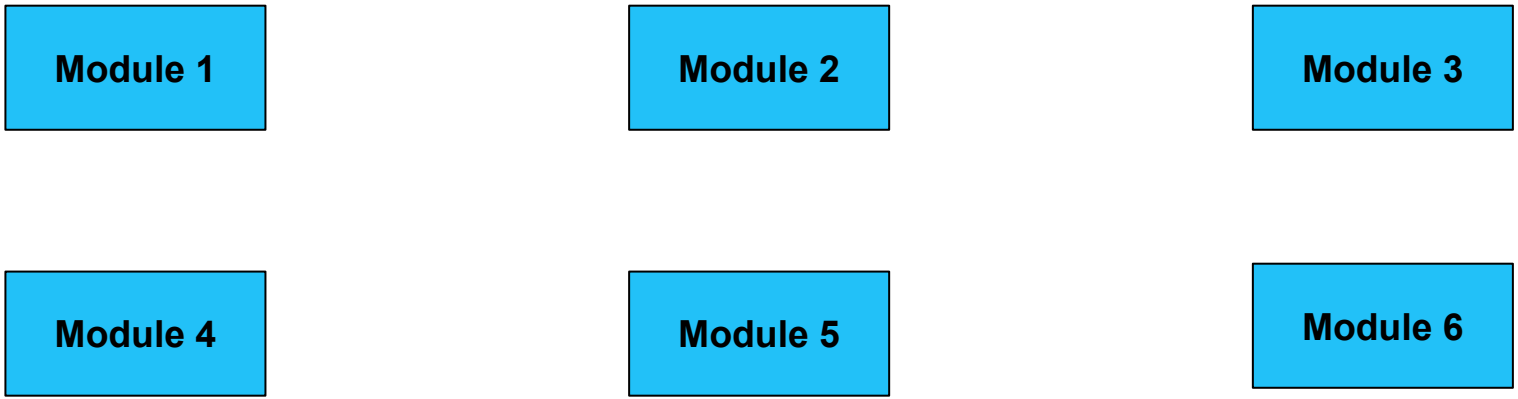
General considerations of structured design

Simplicity is the primary measurement recommended for evaluating alternative designs relative to reduced debugging and modification time. Simplicity can be enhanced by dividing the system into separate pieces in such a way that pieces can be considered, implemented, fixed, and changed with minimal consideration or effect on the other pieces of the system. Observability (the ability to easily perceive how and why actions occur) is another use-

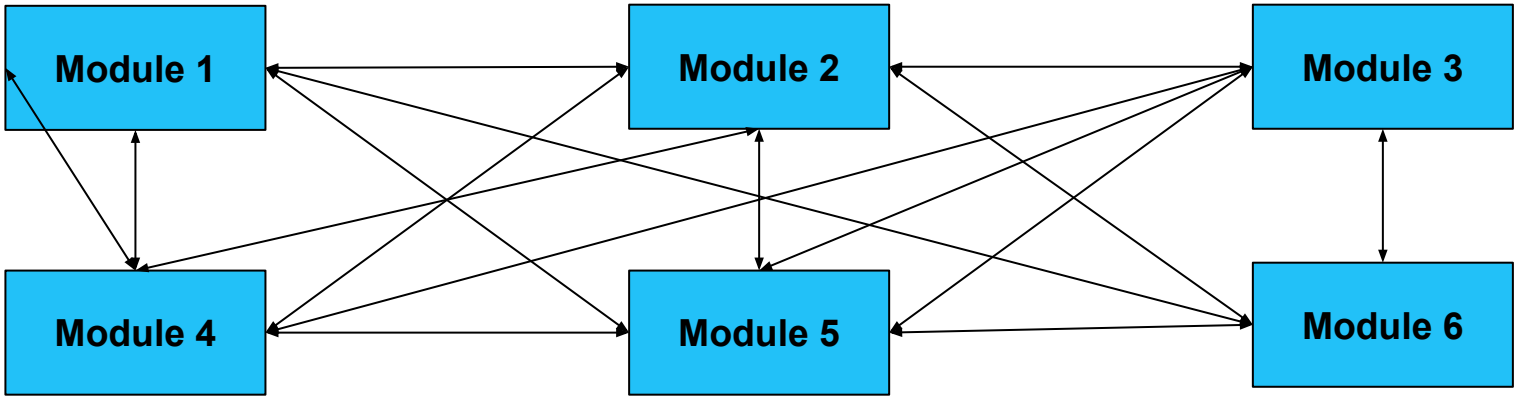
Structured Design

Will modularization be the panacea to producing optimal software ?

Structured Design

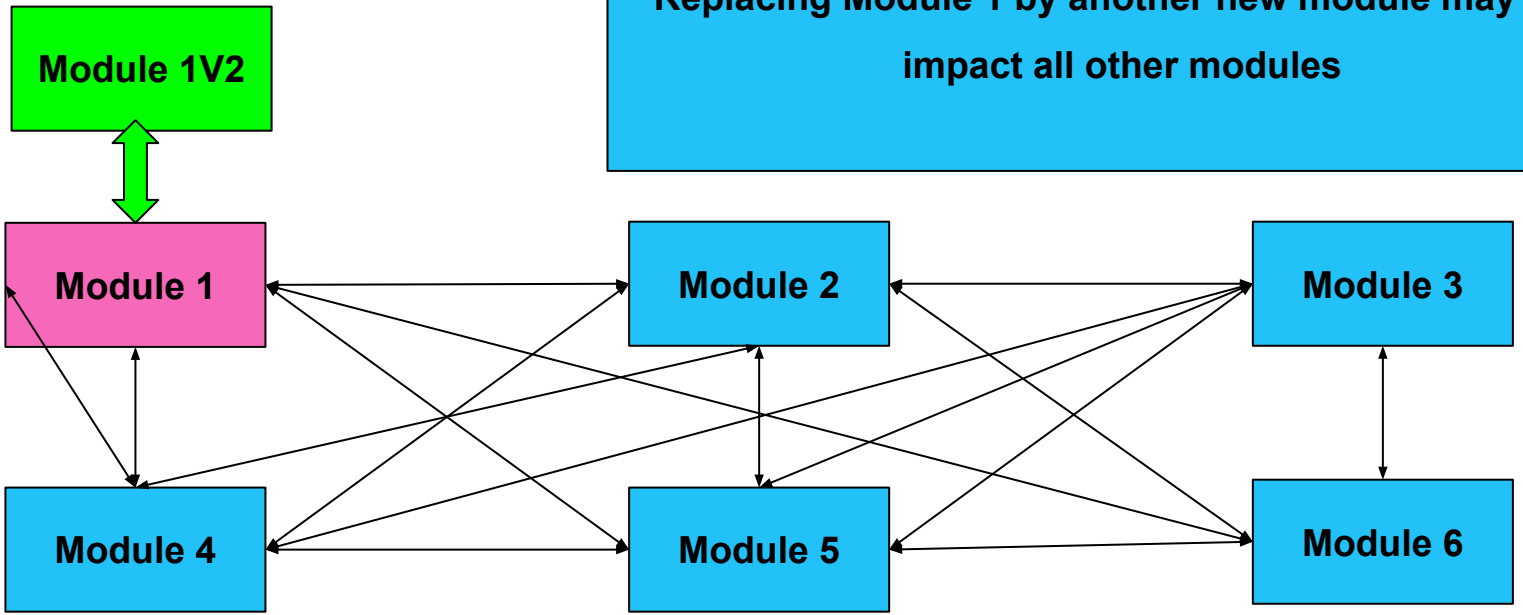


Structured Design



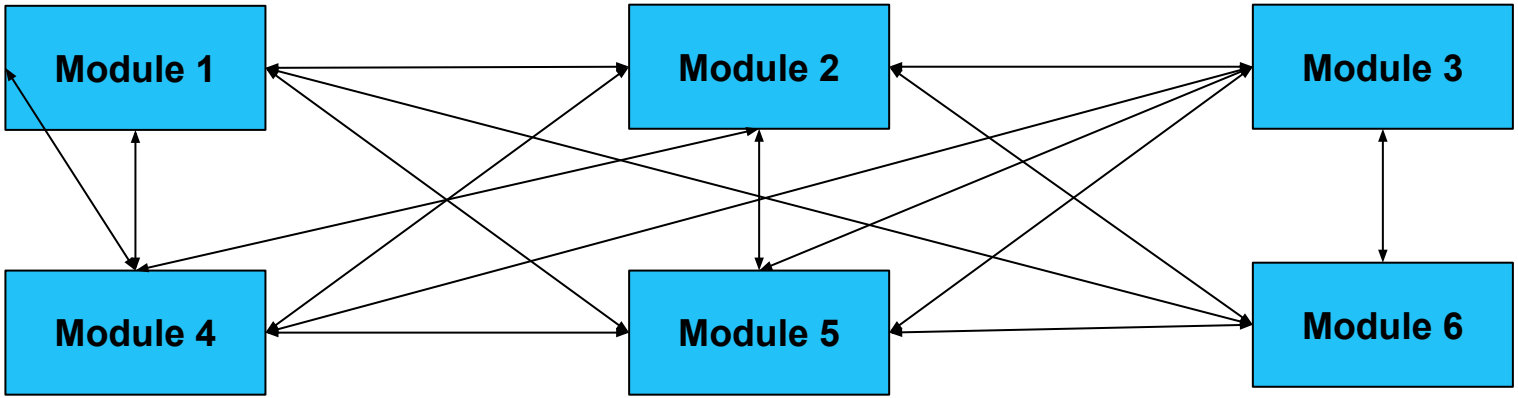
Structured De

Replacing Module 1 by another new module may still impact all other modules



Structured Design

Modules are HIGHLY CONNECTED via many interfaces ?



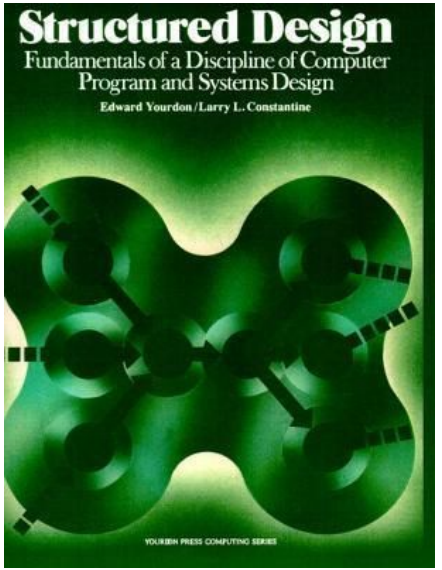
Structured Design

- **50 Years : Developing Software now ?**
  - When it comes to Web Applications, many applications are perceived as just doing the HTML design/coding and link it with some JavaScript files or PHP to do backend business logic.

  - But :
    - When it comes to maintainability or scalability, the software will fail.
    - Example : Implementing the undo features for the to-do App.

Structured Design : Cohesion and Coupling

- **Most important terms :**
  - As modularization is not enough to solve and reduce the complexity of software, but rather how modules are made and connected to each other.
  - The book “Structured Design, 1975” introduced the two notions:
    - Cohesion
    - Coupling
  - They become the most important concepts and metrics for perceiving the quality of good software



Coupling and Cohesion

- **Customer and Order Classes:**
  - Two modules : Customer + Order

```
 5  class Customer{
 6
 7      String firstname;
 8      String lastname
 9
10      Vector<Order> orderHistory;
11  }
```

```
14  class Order{
15
16      Date orderDate;
17      Customer customer;
18  }
```

Coupling and Cohesion

- **Customer and Order Classes:**
  - Two modules : Customer + Order

```
5   class Customer{
6
7       String firstname;
8       String lastname
9
10      Vector<Order> orderHistory;
11  }
```

```
14  class Order{
15
16      Date orderDate;
17      Customer customer;
18  }
```

Can the Customer Module re-used in other systems ? (Without the Order Class ?

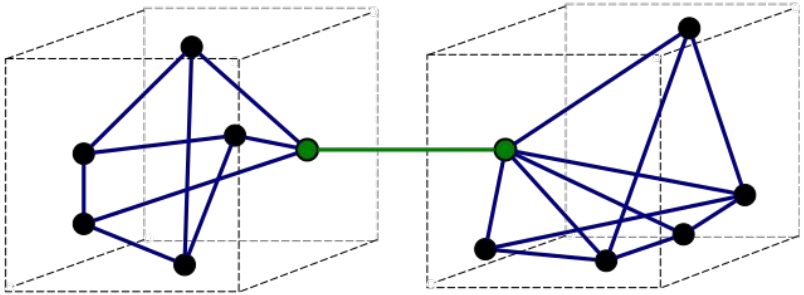
No : Because the Customer is coupled/tied to the Order class

Coupling and Cohesion

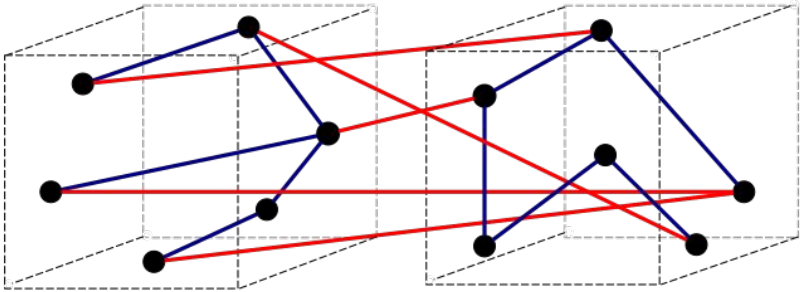
- **Coupling**
  - Definition :
    - Coupling is the measure of the degree of **interdependence** between the modules.
    - A good software will have **low** coupling.
    - Degree of interdependence can be measured by :
      - Number of interfaces
      - Types or complexity of dependency

Coupling and Cohesion

- Coupling



Good Software : Low/Loose Coupling



Bad Software : High Coupling

Coupling and Cohesion

- **Why low Coupling is good**
  - Improved Maintainability: Low coupling reduces the impact when changing one module **risking to change** other modules. Therefore, making it easier to modify or replace individual components without impacting the entire system.
  - Enhanced Modularity: Low coupling allows modules to be developed and tested in isolation. Therefore, improving the reusability of code and modules in other systems
  - Better Scalability and Extensibility: Low coupling facilitates the addition of new modules and the removal of existing ones.

Coupling and Cohesion

- **Types Coupling**
  - Data Coupling:
    - When data of one module is **passed** to another module, this is called data coupling.
  - Example :
    - A module gets the customer addressed (data) from another module by an API endpoint.

Coupling and Cohesion

- **Types Coupling**
  - Stamp Coupling:
    - Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc.
    - Called also : data-structured coupling
    - Example :
      - Need to get the name from another module.
      - Passing structure variable or object where the name is retrieved from the structure.

Coupling and Cohesion

- **Types Coupling**
  - **Control Coupling:**
    - Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.
  - Example :
    - We need to compute the salary for an employee in a given module, the control logic is dependent or tied to the data being retrieved from the absence module.

Coupling and Cohesion

- **Types Coupling**
  - **External Coupling:**
    - Occurs when modules rely on external hardware, APIs, or devices…
      - an external legacy application that sends the same set of data or contents to both modules
      - A hardware requirement
      - A common file/ folder in use by both modules
      - When both use same switch/router in the network for communication

Coupling and Cohesion

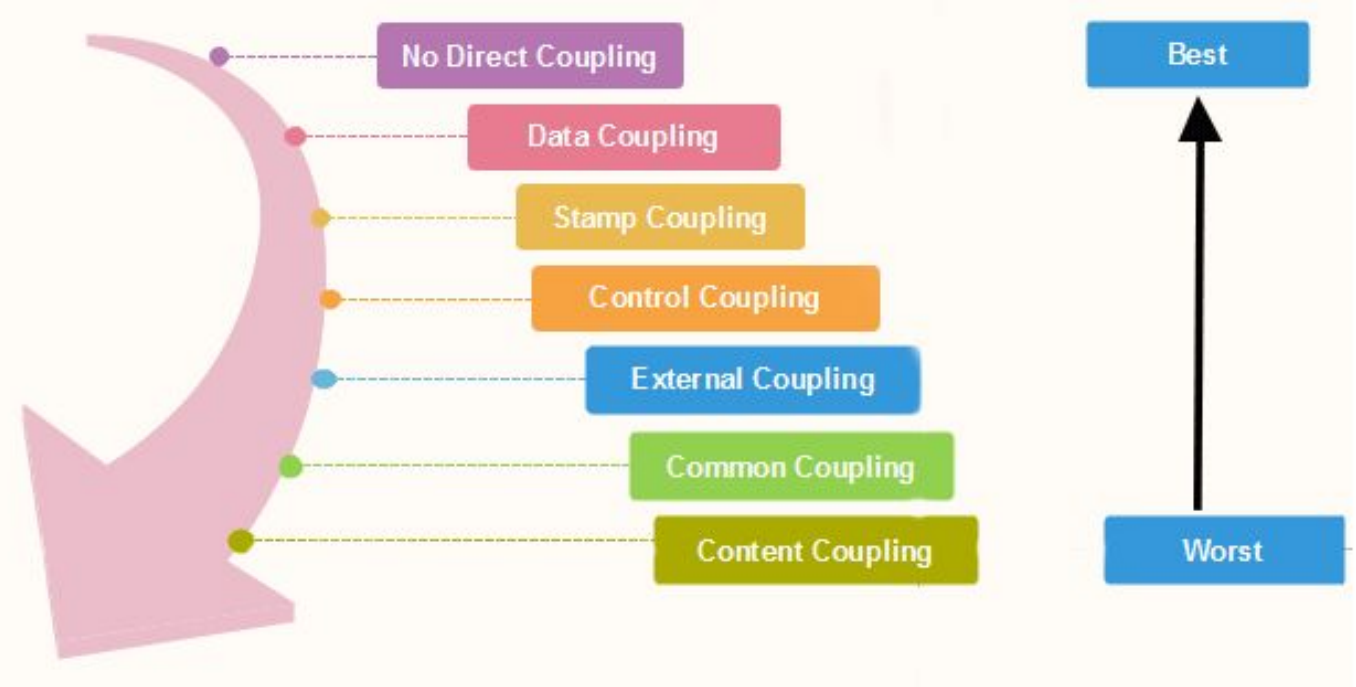
- **Types Coupling**
  - **Common Coupling:**
    - Common coupling is said to occur when two or several modules have access to the same global data
  - Example
    - A module writes data to a file which is used by another module.

Coupling and Cohesion

- **Types Coupling**
  - **Content Coupling** (**Most Undesirable**) :
    - Exists among two modules if they share code, e.g., a branch from one module into another module. (Invocation of a function)
  - Example :
    - A class invokes another class.

Coupling and Cohesion

- **Types Coupling**



Coupling and Cohesion

- **Cohesion**
  - Definition :
    - The degree to which the elements of a module **belong together**.
    - It measures the strength of relationships between pieces within a module based on a given aspect :
      - Functionalities
      - Objectives
      - ...

Coupling and Cohesion

- **Cohesion**
  - Remember the class for shapes, it has two methods
    - ComputeArea
    - OutputArea
  - Those two functionalities are
    - Not related
    - Not cohesive

```php
class AreaCalculator{
    protected $shapes;
    public function __construct($shapes = [])
    {
        $this->shapes = $shapes;
    }
    public function sum()
    {
        foreach ($this->shapes as $shape) {
            if (is_a($shape, 'Square')) {
                $area[] = pow($shape->length, 2);
            } elseif (is_a($shape, 'Circle')) {
                $area[] = pi() * pow($shape->radius, 2);
            }
        }

        return array_sum($area);
    }
    public function output()
    {
        return implode('', [
          '',
            'Sum of the areas of provided shapes: ',
            $this->sum(),
          '',
      ]);
    }
}
```

Coupling and Cohesion

- **Why High Cohesion is a good quality**
  - Improved readability and understandability: High cohesion results in clear, focused modules with a single, well-defined purpose, making it easier for developers to understand the code and make changes.
  - Better error isolation: High cohesion reduces the likelihood that a change in one part of a module will affect other parts, making it easier to
  - isolate and fix errors. Improved reliability: High cohesion leads to modules that are less prone to errors and that function more consistently,

Coupling and Cohesion

- **Types or Levels of Cohesion**
  - **Functional Cohesion** : different elements of a module, cooperate to achieve a single function
  - **Sequential Cohesion** : element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
  - **Communicational Cohesion** : when all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

Coupling and Cohesion

- **Types or Levels of Cohesion**
  - **Procedural Cohesion** : A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
  - **Temporal Cohesion** : When a module includes functions that are associated by the fact that all the methods must be executed in the same time
  - **Logical Cohesion** :When all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
  - **Coincidental Cohesion** : A module is said to have coincidental cohesion it contains tasks or instructions that may contain no meaningful relationship between them.

Coupling and Cohesion

- **Coupling vs. Cohesion vs. Good Design**
  - Advocates of structured design emphasis clearly that neither coupling nor cohesion are absolute truth: in design, everything is a trade-off. The trade-off is always based on experience, the benefits of the solutions, the drawbacks possibly to be faced.
  - That's why :
    - Exploring and experimenting is so important.
    - Software development is always difficult.

Architectural Techniques

- **What can be done to meet the technical attributes**
  - Use of architectural design
  - Combine if necessary
  - Ensure the use design patterns
  - Use other architectural techniques
    - Caching (Remote, vs. Cloud. Vs. Client……)
    - Messages
    - …

Documenting Software Design and Architecture

- The Software Architecture Document contains information about :
  - Components and their mapping and structuring
  - Technological Stack
  - Functional and nonfunctional requirements
  - Architectural diagrams including UML diagrams.
  - Executive Summary
- The document should describe what should be developed and how including the list of modules and how they should be communicating
- Important : No development before the document.

Documenting Software Design and Architecture

- Structure of the document :
  - Background : short section to describe briefly the system from a business point of view discussing :
    - System objectives
    - What the system do from the end-user perspective
    - Expected business impact
    - This section should never contain technical details.

Documenting Software Design and Architecture

- Structure of the document :
  - Requirements :
    - This includes a brief (not detailed) listing of :
      - Functional requirements
      - Non-functional requirements including the technical capacities which are not mentioned explicitly in the SRS document.

Documenting Software Design and Architecture

- Structure of the document :
  - Executive Summary:
    - The audience of this section is the management team.
      - Management are non-technical and are usually business-oriented people.
    - The goal of this section is to provide a high-level view of the architecture to convince the audience that the proposed solution is viable.
    - Executive summary : should not contain background information or the requirements again.

Documenting Software Design and Architecture

- Structure of the document :
  - Executive Summary:
    - The audience of this section is the management team.
      - Management are non-technical and are usually business-oriented people.
    - The goal of this section is to provide a high-level view of the architecture explaining the decision and choices taken to convince the audience that the proposed solution is viable.

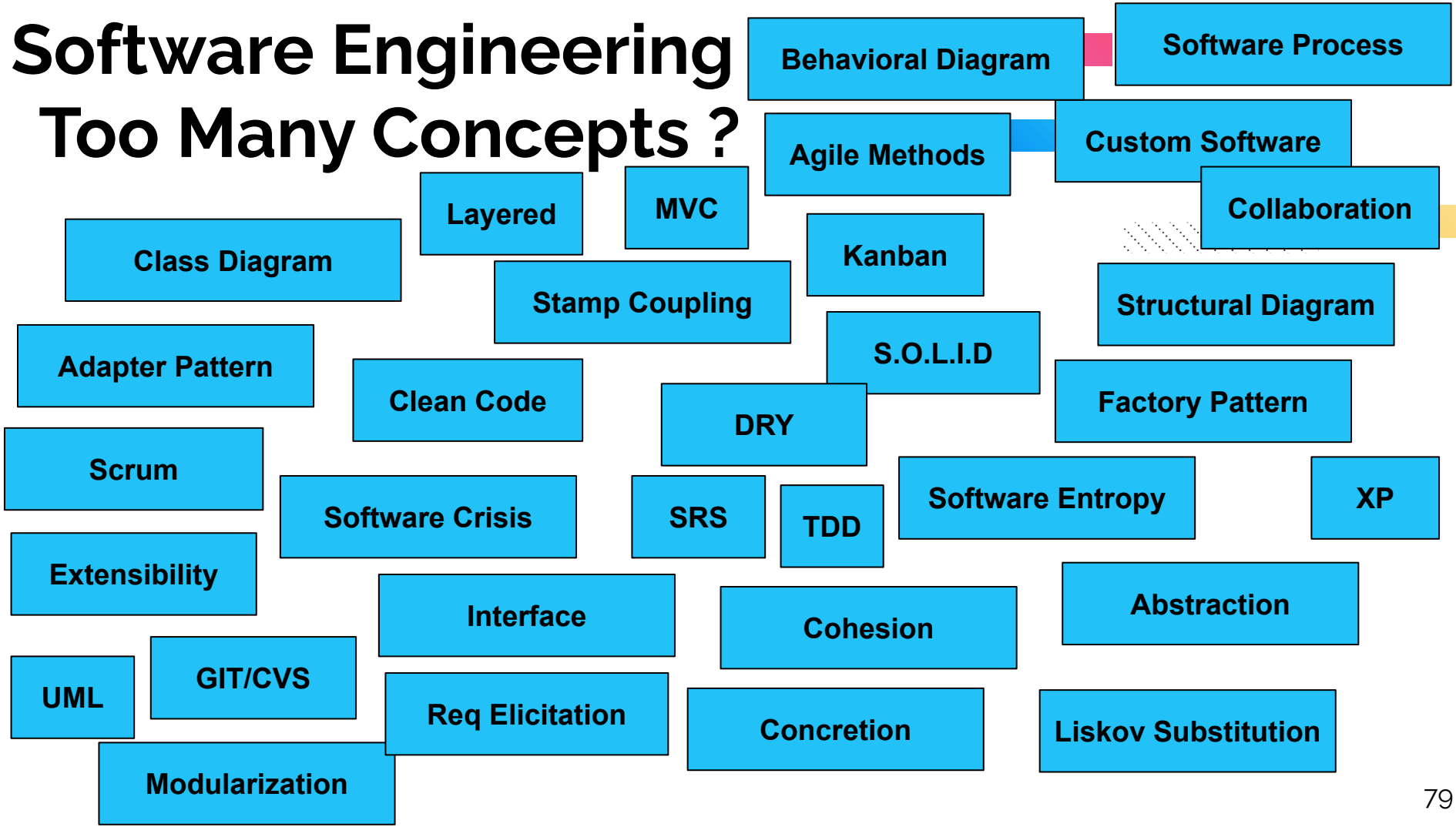
Documenting Software Design and Architecture

- Structure of the document :
  - Architecture Overview:
    - It would provide a high-level of the architecture
    - Shall be started with a general introduction describing the type of the application, the choice of architectural styles of the reasons for it.
    - It would include High-Level Diagram showing all architectural components composing the system.
    - A brief description of each component shall be provided
    - Technological Stack : Better to describe in the next section

Documenting Software Design and Architecture

- Structure of the document :
  - Components/Services/Subsystems/Classes
    - Each components shall be extremely detailed. This includes:
      - Component roles
      - Components Elements
      - UML Diagrams if necessary.
      - Component APIs/Interfaces with arguments and response structure codes (include even function names..)
      - Data Format
      - Technological Stack or constraints
      - Development guidelines

Software Engineering Too Many Concepts ?



Software Engineering: 11 weeks of lectures….

- A lot of concepts
- A lot of methods
- A lot of techniques
- A lot of terminologies
- A lot of Blahware…
- **Not to over-complicate the development of software development**
- But to :
  - **Simplify the Development and Implementation**
  - **Make sure that the system can be extensible and maintainable**

Ticket Queueing System: Specification

- Raw Specification:

A company (Algerie Poste) wanted to develop a new software for managing the queue of their waiting clients inside the waiting room where a new client is given a ticket with a number. There are many agents dealing with clients. When an agent becomes free, he/she will call the next client where the number is shown in a central display screen.

Ticket Queueing System: Functional Requirements

- List of requirements :
  - Issue Ticket
  - Call Next Customer
  - Skip To Next Customer
  - Display Waiting Info
  - Manage and reset the counter

Ticket Queueing System: Functional Requirements

- List of requirements :
  - Implicit requirements
    - Customize the printed ticket :
      - Add/Edit company logo
      - Add/Edit company name/address
      - Add/Edit Greeting messages…
    - Special priority queues
      - For people with special needs
    - ?

Ticket Queueing System: Functional Requirements

- List of requirements :
  - Implicit requirements
    - Custom
      - Ad
      - Ad
      - Ad
    - Special priority queues
      - For people with special needs
    - ?

What's the objective for a company to deploy such system ?

What the company would like to see when they deploy the system ?

Ticket Queueing System: Functional Requirements

- List of requirements :
  - Implicit requ
    - Custom
      - Ad
      - Ad
      - Ad
    - Special
      - For people with special needs
    - ?

- Statistics about the daily wait time + number of customers being served.

- Times/hours when the agency is busy vs when the agency has no customers.

- Analyse the performance of employees.

Ticket Queueing System: Functional Requirements

- List of requirements :

- Statistics about the daily wait time + number of customers being served.

- Times/hours when the agency is busy vs when the agency has no customers.

- Analyse the performance of employees.

These objectives can greatly impact the architecture and implementation :

Employee Performance : Authentication may be needed for employee + track all activities for an employee

- For people with special needs
  - ?

Ticket Queueing System: Functional Requirements

- List of requirements :

- Statistics about the c
of customers

- Times/hours when
when the agency

- Analyse the perfor

What if the objective : Reduce fraudulent and abusive activities of people getting multiple tickets for the sake to sell them for a profit ?

s can greatly impact the
and implementation :

ance : Authentication may
oyee + track all activities for
employee

  - For people with special needs
    - ?

Ticket Queueing System: Functional Requirements

- List of requirements :

What if the objective : Reduce fraudulent and abusive activities of people getting multiple tickets for the sake to sell them for a profit ?

This will even impact the hardware architecture, legal aspect, software architecture

As a potential solution : biometric solution can be used to record the fingerprint of people …

  - For people with special needs
    - ?

Ticket Queueing System: Functional Requirements

- To remember about the requirements
  - The customer does not know what they want,
  - Ask always for their objectives
  - Try to see existing products, analyse their functionalities
  - Gather and understand as much knowledge about the application domain and the company business processes.
  - Ask who will be using the system, where the system will run, possibly the materials available, infrastructure…

Ticket Queueing System: Technical Capacities

- Extensibility ?
- Scalability ?
- Portability ?
- Manageability ?
- Fault tolerance ?
- Performance ?
- Usability ?
- …