

Advanced Software Engineering Techniques

Course 1 – October 7, 2025

Adrian Iftene
adiftene@info.uaic.ro

ASET – overview

- ▶ SWEBOK: place and role of software engineering, knowledge areas (KAs), related disciplines
- ▶ Development and maintenance of the systems: model driven agile development, patterns of enterprise application architecture, test-driven development, refactoring: code architecture
- ▶ Object oriented design classes: SOA, object-oriented design principles, Serverless
- ▶ Modeling, business modeling: BPMN, domain specific languages (DSL), frameworks: Eclipse Modeling Framework (EMF), Open Architecture Ware (OAW)

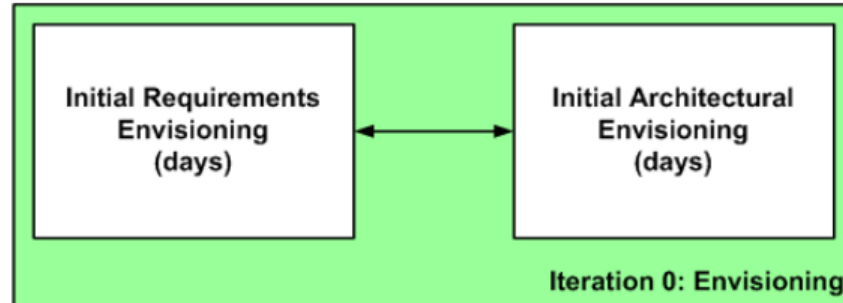
Model driven development

- ▶ Model-driven development (MDD) – software methodology focused on creating models close to a specific field than informatics concepts
- ▶ Model-driven architecture (MDA) is the best known initiative of MDD and was launched by the group OMG (Object Management Group) in 2001

Agile model driven development

▶ AMDD is agile version of MDD

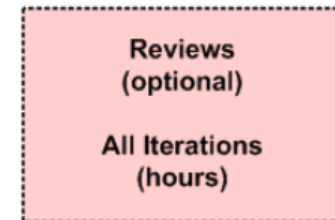
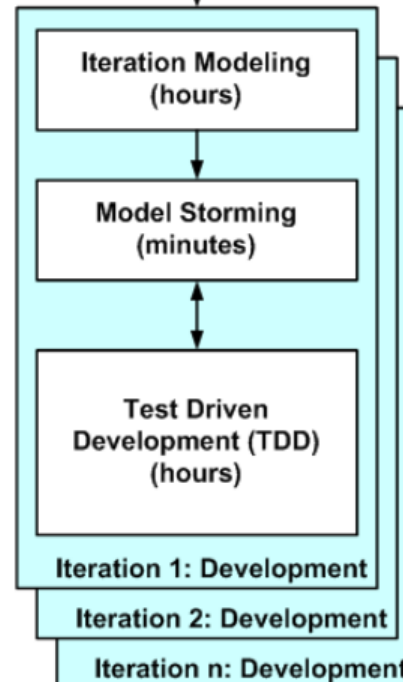
- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision



- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications



Test driven development

▶ Test-Driven Development – TDD

TDD steps:

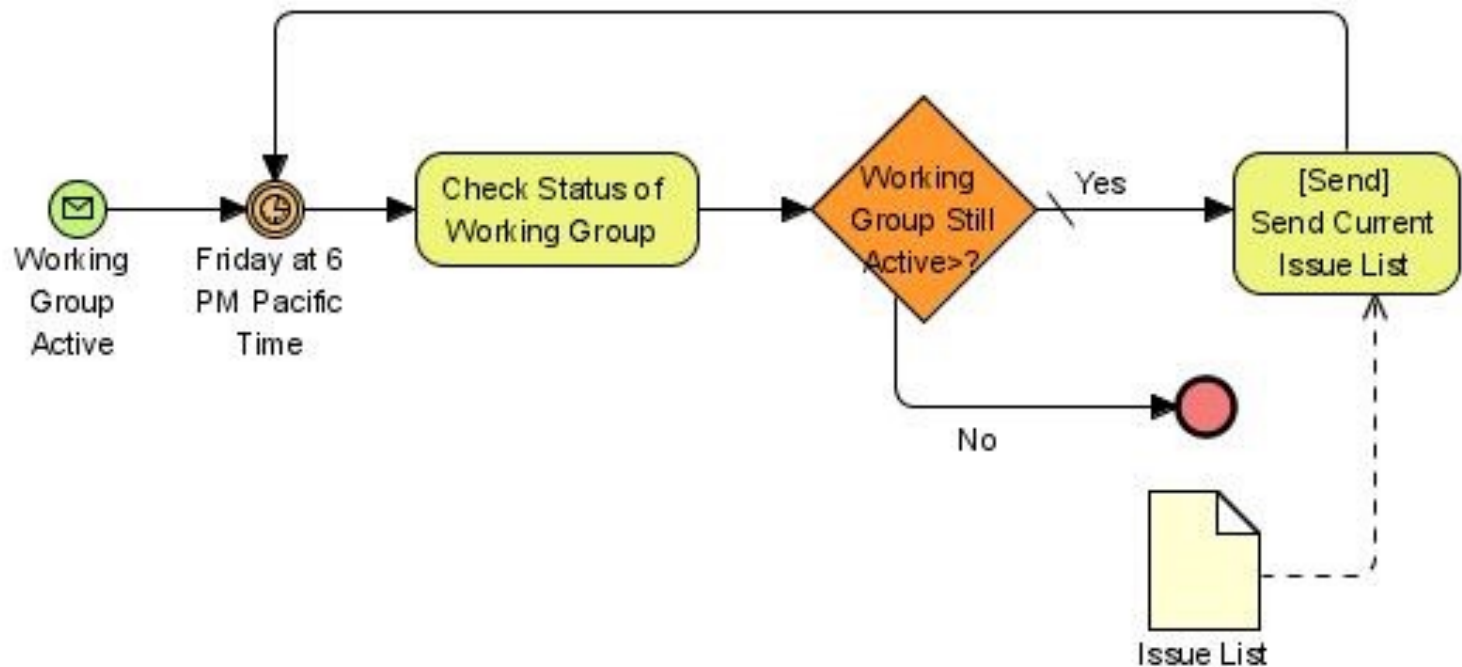
1. Add a test.
2. Execut tests; the new test will fail.
3. Add functional code such that pass all test.
4. Run tests again.
 - If the test fails, go to 3.
 - If the tests pass successfully, we can continue with other functionality
5. Refactoring code (functional and testing)

Modeling

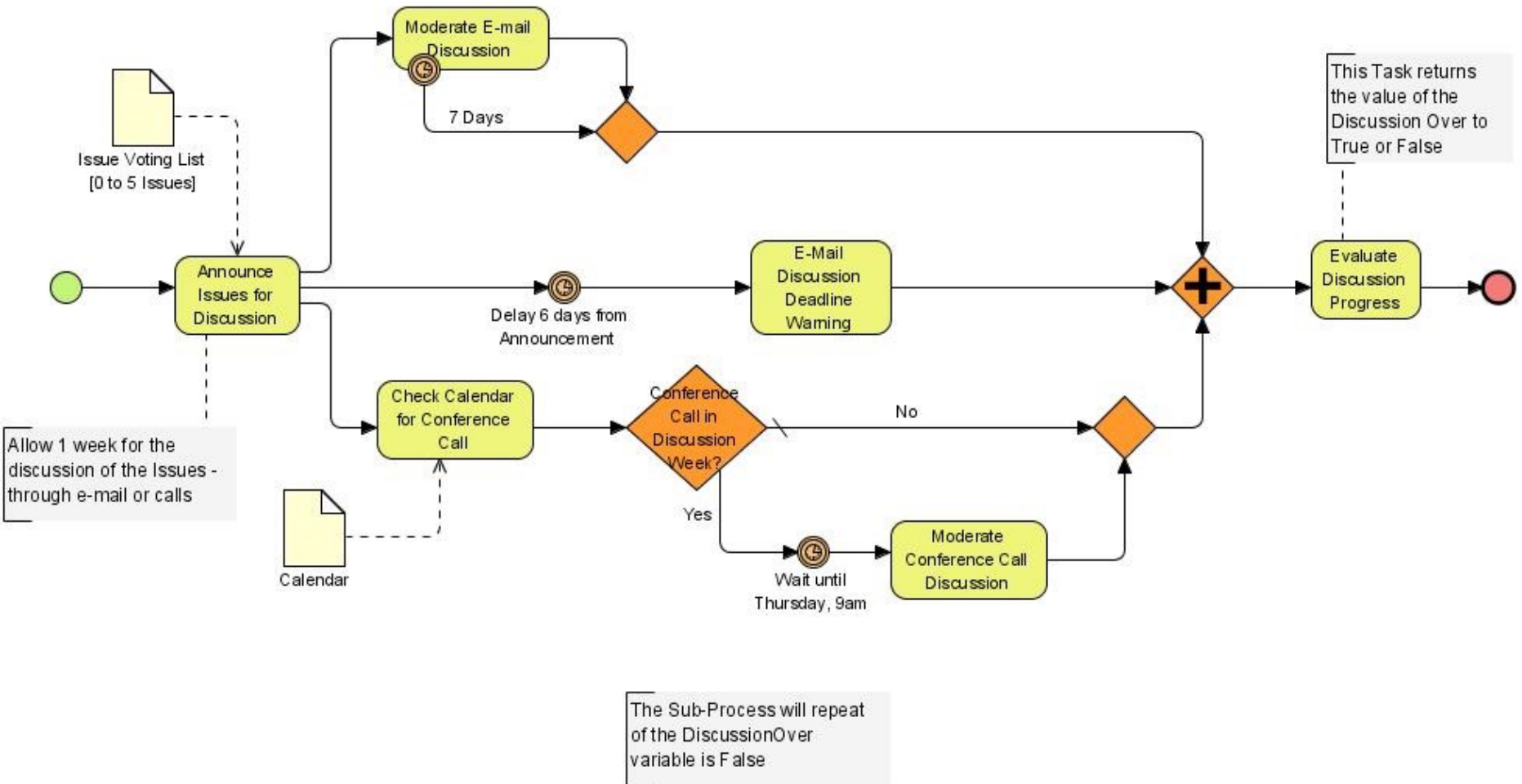
- ▶ IBM Rational Rose Modeler
- ▶ BPMN
- ▶ Domain specific languages (DSL)
- ▶ Working frameworks:
 - Eclipse Modeling Framework
 - Open Architecture Ware (OAW)

BPMN

- ▶ Business Process Modelling Notation (BPMN) is a graphical representation for specifying business processes in a workflow

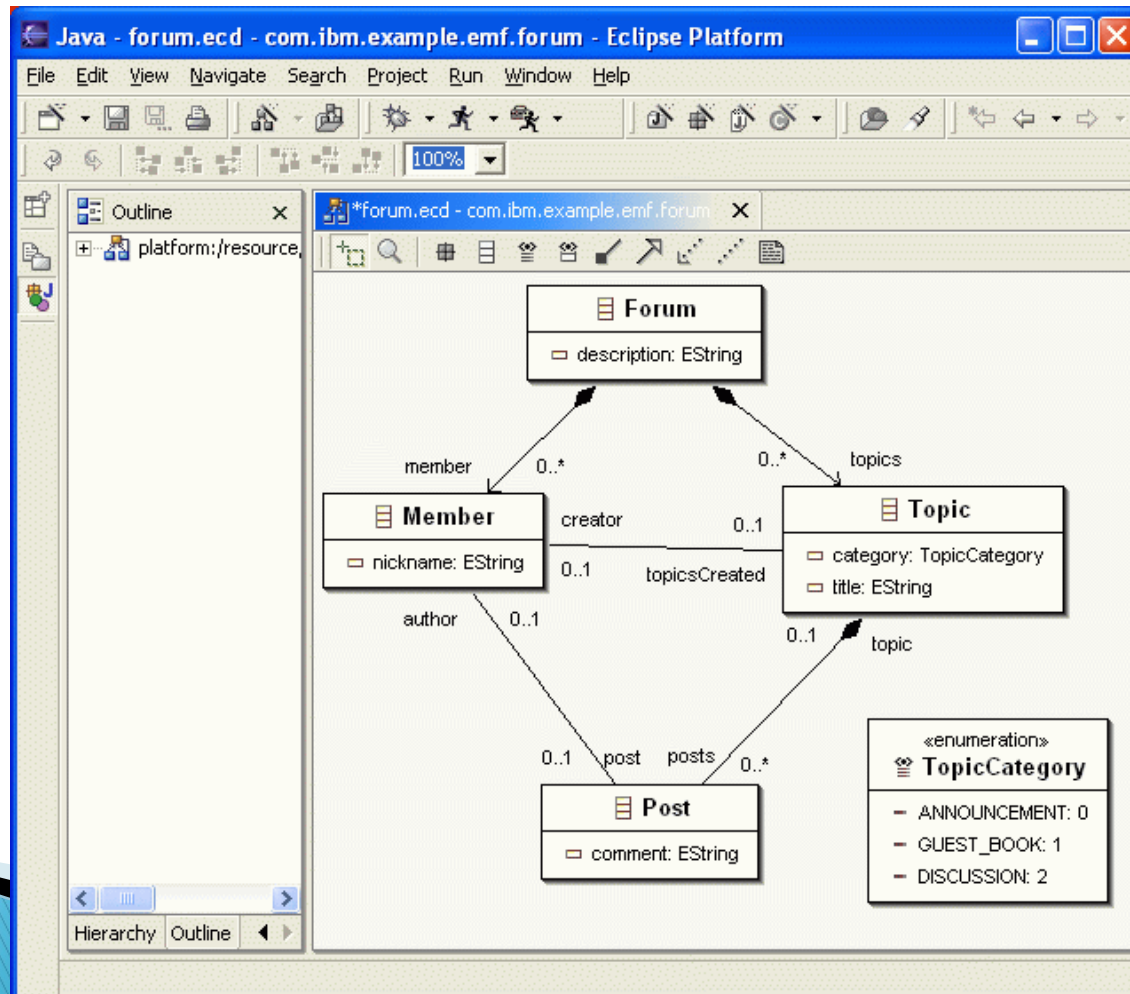


BPMN – Example



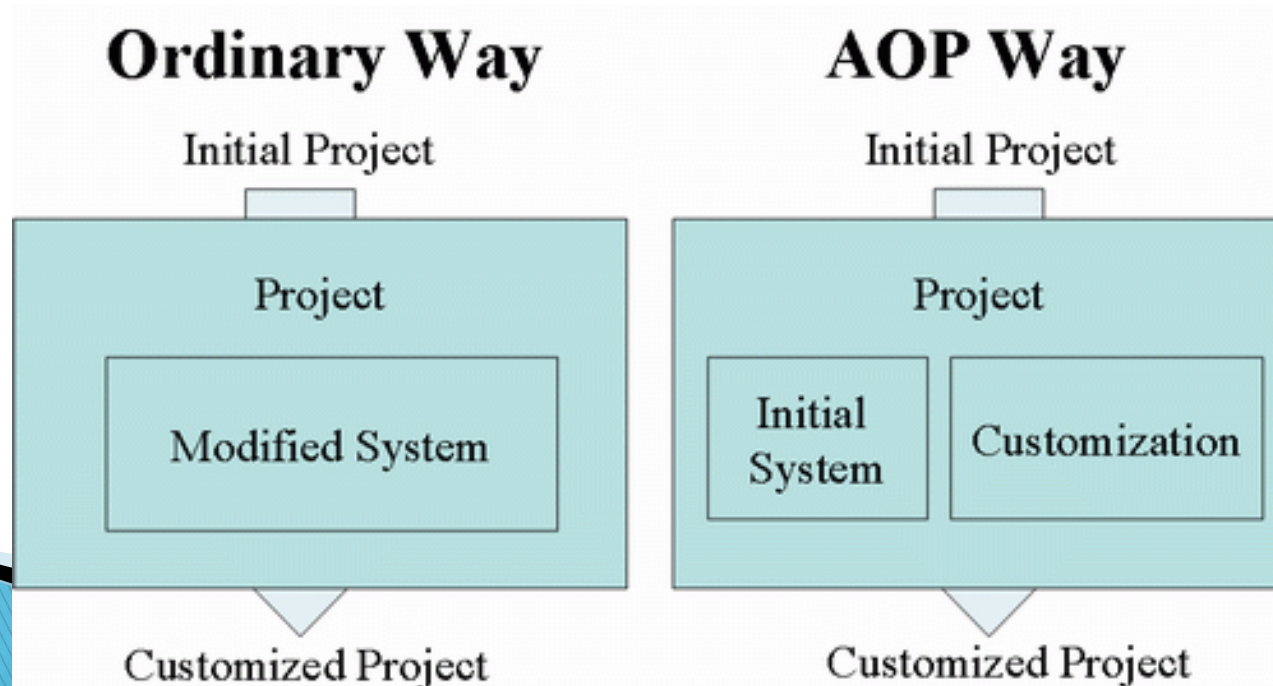
Eclipse Modeling Framework

- ▶ EMF is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model



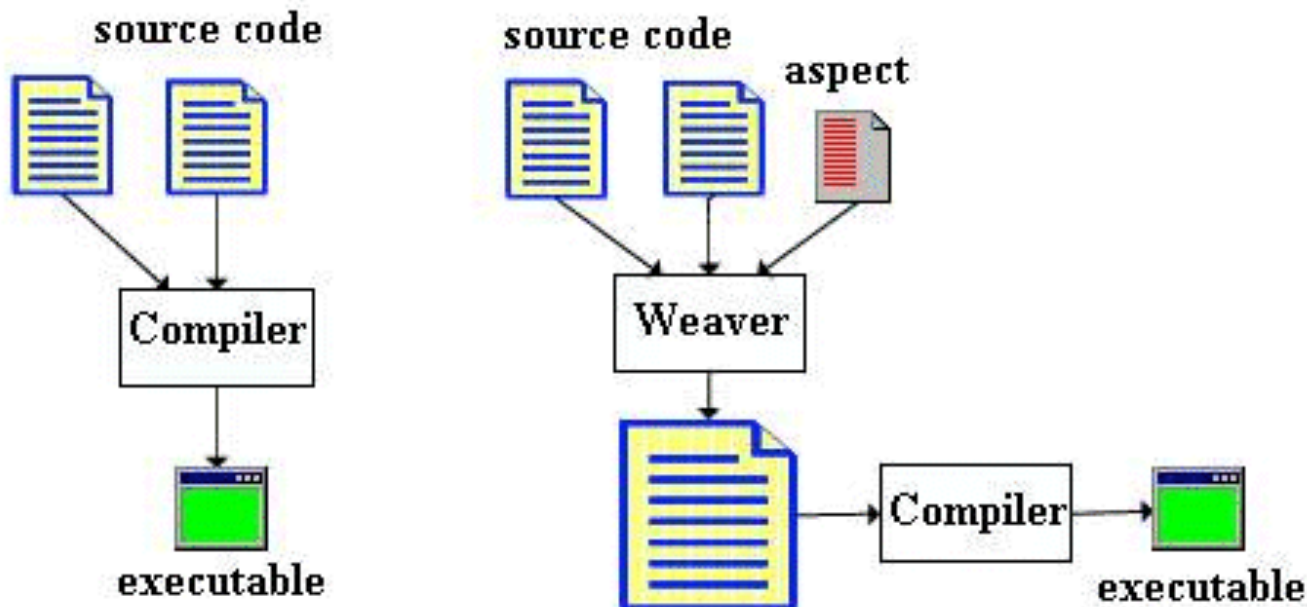
Aspect-oriented programming

- ▶ AOP is a programming paradigm which isolates secondary or supporting functions from the main program's business logic
- ▶ AOP increases modularity by allowing the separation of cross-cutting concerns
- ▶ AOP includes programming techniques and tools that support the modularization of concerns at the level of the source code



AOP – Basic Terminology

- ▶ **Cross-cutting concerns** – aspects of a program which affect other concerns
- ▶ **Advice** – additional code
- ▶ **Pointcut** – point where additional code is executed
- ▶ **Aspect** – the combination of the pointcut and the advice



AOP Languages

- ▶ Examples:
 - AspectJ,
 - CaesarJ,
 - CLOS,
 - Compose,
 - JAsCo,
 - ObjectTeams

AOP – AspectJ – Hello World!

The screenshot shows an IDE window with the following components:

- Package Explorer:** Shows a project named 'AspectJHelloWorld' with a source folder 'src' containing 'HelloWorld.java' and 'HelloWorldBeforeAfter.aj'. It also shows 'JRE System Library [JavaSE-1.6]' and 'AspectJ Runtime Library'.
- Code Editor:** Displays the content of 'HelloWorldBeforeAfter.aj'. The code is as follows:

```
public aspect HelloWorldBeforeAfter {  
    pointcut mainMethod() : execution(public static void main(String[]));  
    before() : mainMethod() {  
        System.out.println("Before Hello World!");  
    }  
    after() returning : mainMethod() {  
        System.out.println("After Hello World!");  
    }  
}
```

The 'public aspect' declaration is highlighted with a blue box, the 'pointcut' line with a red box, and the 'before' and 'after' advice blocks with a green box.
- Console:** Shows the execution output: '<terminated> HelloWorld (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 23, 2010 5:32:17 PM)' followed by 'Before Hello World!', 'Hello world!', and 'After Hello World!'.

- ▶ aspect
- ▶ pointcut
- ▶ advice

AOP – AspectJ – Example 2

- ▶ **Problem:** we want to know when something changes the student (*name* or *grade*)
- ▶ **Solution:** we add a pointcut for all “set” methods

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `AspectJHelloWorld`, `install`, `Jadex`, `Student`, `src`, `student`, `Student.java`, `StudentAspect.aj`, and `StudentAspectWild.aj`.
- Editor:** Displays the code for `StudentAspectWild.aj` in the `student` package. The code is:

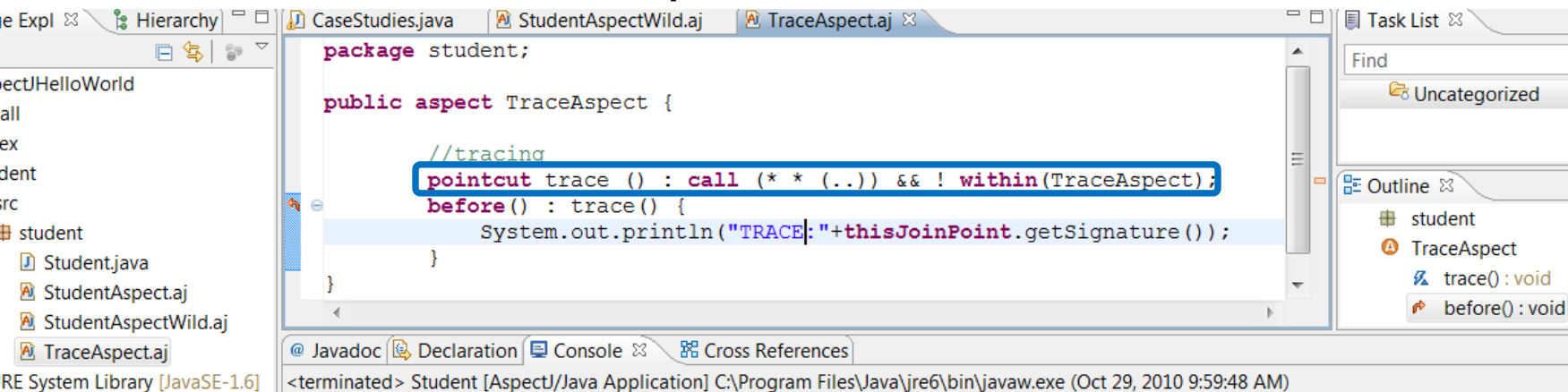
```
package student;

public aspect StudentAspectWild {
    pointcut changeStudent() : call(public * set*(..));
    before() : changeStudent() {
        System.out.println("STUDENT set:"+thisJoinPoint.getSignature());
    }
}
```
- Task List:** Shows a task list with a search bar and a category labeled "Uncategorized".
- Outline:** Shows the outline of the `student` package, including `StudentAspectWild` with methods `changeStudent(): void` and `before(): void`.
- Console:** Shows the output of the program, which is:

```
<terminated> Student [AspectJ/Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 29, 2010 9:19:20 AM)
STUDENT set:void student.Student.setName(String)
STUDENT set:void student.Student.setGrade(int)
in Student.setGrade:9
STUDENT set:void student.Student.setGrade(int)
in Student.setGrade:10
Ionel 10
```

AOP – AspectJ – Example 3

- ▶ **Problem:** we want to trace our program execution
- ▶ **Solution:** we add a pointcut for all methods



The screenshot shows an IDE with three tabs: CaseStudies.java, StudentAspectWild.aj, and TraceAspect.aj. The TraceAspect.aj tab is active, showing the following code:

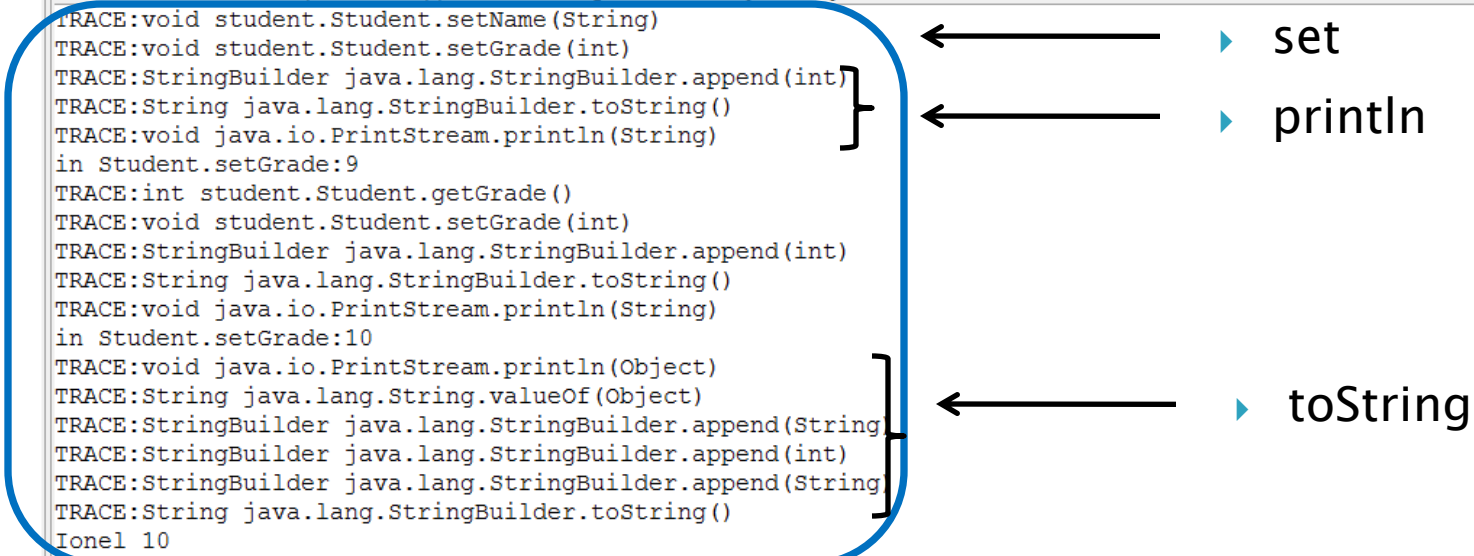
```
package student;

public aspect TraceAspect {

    //tracing
    pointcut trace () : call (* * (..) && ! within(TraceAspect));
    before() : trace() {
        System.out.println("TRACE: "+thisJoinPoint.getSignature());
    }
}
```

The console output shows the following trace messages:

```
<terminated> Student [AspectJ/Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 29, 2010 9:59:48 AM)
TRACE:void student.Student.setName (String)
TRACE:void student.Student.setGrade (int)
TRACE:StringBuilder java.lang.StringBuilder.append (int)
TRACE:String java.lang.StringBuilder.toString ()
TRACE:void java.io.PrintStream.println (String)
in Student.setGrade:9
TRACE:int student.Student.getGrade ()
TRACE:void student.Student.setGrade (int)
TRACE:StringBuilder java.lang.StringBuilder.append (int)
TRACE:String java.lang.StringBuilder.toString ()
TRACE:void java.io.PrintStream.println (String)
in Student.setGrade:10
TRACE:void java.io.PrintStream.println (Object)
TRACE:String java.lang.String.valueOf (Object)
TRACE:StringBuilder java.lang.StringBuilder.append (String)
TRACE:StringBuilder java.lang.StringBuilder.append (int)
TRACE:StringBuilder java.lang.StringBuilder.append (String)
TRACE:String java.lang.StringBuilder.toString ()
Ionel 10
```



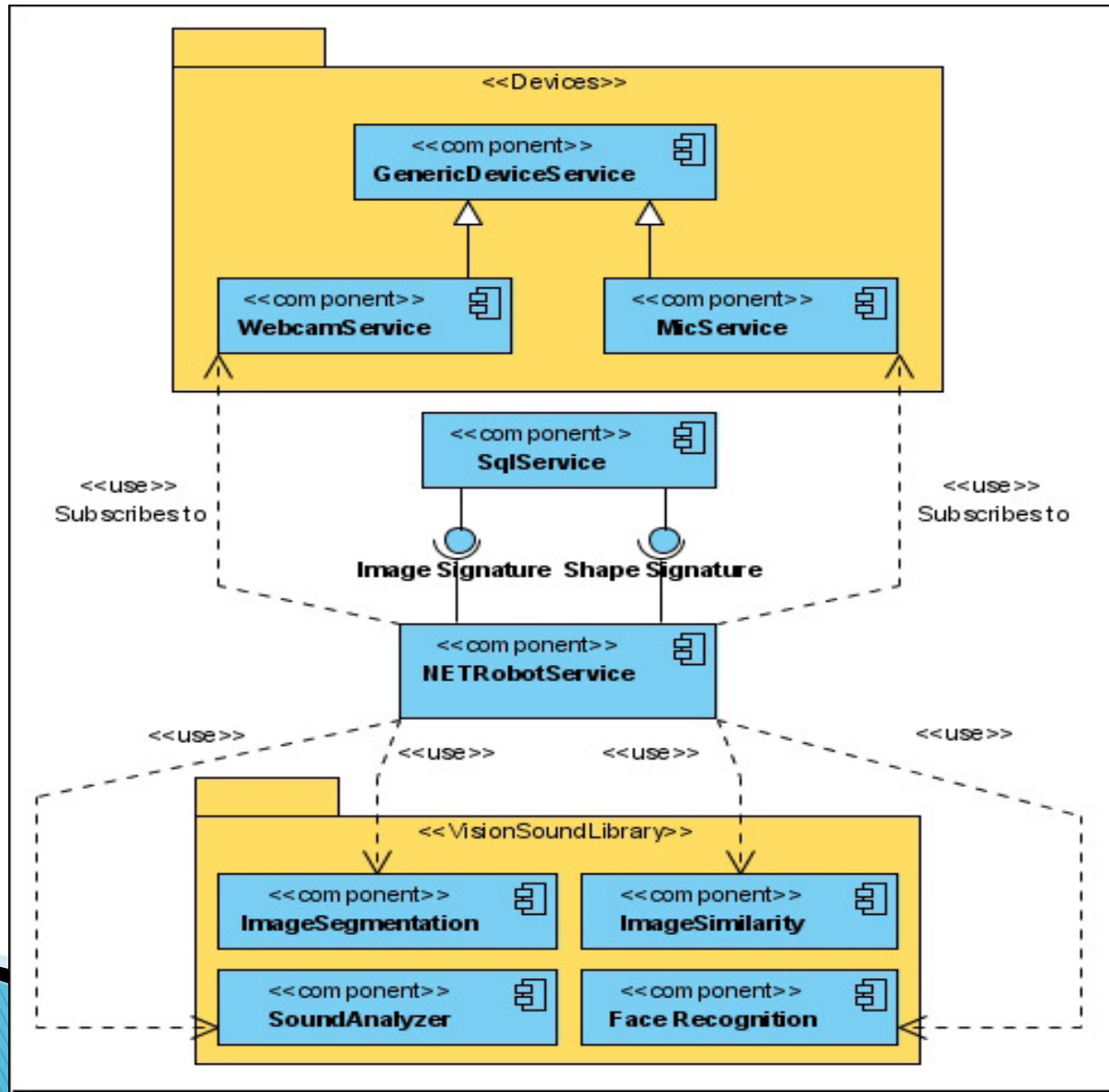
SOA

- ▶ SOA (Service Oriented Architecture) involves **distributing application functionality** into smaller units, **distinct** – called services – which can be distributed over a network and can be used together to create complex applications
- ▶ Services are **independent functional units** that solve specific problems and can be combined to solve complex problems.
- ▶ They can also be reused in different applications

SOA – Example

- ▶ Examples of services:
 - complete an application online to create an account
view a bank statement
 - make an online ticket orders
 - For a robot: services for vision, hearing, moved

SOA - .NetROBOT - Tudor D.



Object-oriented design principles

- ▶ Architecture and dependencies: *When we say that we have a degraded project?*
- ▶ Design principles of classes: *responsibility, dependencies, separation*
- ▶ Architecture design principles:
 - Reuse, versioning, closing
 - Coupling, dependence
- ▶ Object-oriented design patterns:
 - Abstract server, Adapter, Observer, Bridge, Abstract Factory

Architectural degradation

- ▶ Rigid – hard to change
- ▶ Fragile – changes have undesirable effects
- ▶ Immobility – separation into components is difficult
- ▶ Viscous – things not running to properly
- ▶ Additional complexity
- ▶ Additional repetition
- ▶ Opacity – hard to understand

Refactoring

- ▶ Successive changes lead to sub-optimal code structure
 - increase the complexity
 - decreases clarity
- ▶ Refactoring is a change in the internal structure of a software product in order to make it easier to understand and modify without changing its observable behavior
- ▶ Results:
 - decreased coupling
 - increasing cohesion

Refactoring – When?

- ▶ The following situations are signals for refactoring:
 - duplicate code
 - long methods
 - large classes
 - Long lists of parameters
 - Instructions switch by type objects – is recommended polymorphism
 - Speculative Generality – Hierarchy of classes that subclasses have the same behavior
 - Intense communication between objects (strong coupling)
 - Chaining of messages

Github

- ▶ Github – code
- ▶ GoogleDrive – documents

Links

- ▶ SOA: <http://www-01.ibm.com/software/solutions/soa/> ,
<http://ro.wikipedia.org/wiki/SOA>
- ▶ SOA for the real world: <http://www.javaworld.com/javaworld/jw-11-2006/jw-1129-soa.html?page=1>
- ▶ Abstract Server
<http://today.java.net/pub/a/today/2004/06/8/patterns.html>
- ▶ Agile Model Driven Development (AMDD)
<http://www.agilemodeling.com/essays/amdd.htm>
- ▶ Florin Leon – IP Curs 11
http://eureka.cs.tuiasi.ro/~fleon/Curs_IP/IP11_Implementarea.pdf
- ▶ OAW <http://www.openarchitectureware.org/>

Bibliography

- ▶ Robert Cecil Martin: *Design Principles and Design Patterns*. www.objectmentor.com.
- ▶ Robert Cecil Martin: *Agile Development. Principles, Patterns, and Practices*, Prentice–Hall, 2003