# Aspect Oriented and Component Adaptation in SPL

**Team 22**

Agrawal, Ria
A20356603
ragrawa4@hawk.iit.edu

Anand, Aishwarya
A20331867
aanand12@hawk.iit.edu

Dwivedi, Geetanjali
A20372250
gdwivedi@hawk.iit.edu

Sharma, Navjot
A20376692
nsharma11@hawk.iit.edu

Vadakke Veettil, Ashik Shukoor
A20375848
avadakkeveettil@hawk.iit.edu

## Introduction

Software Product Lines is a technique that explores systematic reuse of software artifacts in large scale to implement applications that share a common domain and have some customized features. SPL is emerging as a viable and important development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers. Software product line engineering can also enable rapid market entry and flexible response, and provide a capability for mass customization. It refers to software engineering methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production. Component adaptations and aspect orientation programming are the two terms that play vital role in SPL. Aspect-oriented programming is a new programming technique that takes another step towards increasing the kinds of design concerns that can be captured cleanly within source code. AOP modularizes software to isolates supporting functions from the business logic. It allows concerns to be expressed separately and automatically unified into working system. In past decades, object-oriented software development has achieved immense success but it has not achieved extensive reusability and maintainability because individual classes are too detailed, specific, and bound to application domains. As a result, component based software development emerged in late 1990s as a reuse-based approach to enhance the flexibility and maintainability of software systems. Component-based software engineering intends to construct applications by putting together reusable components. We can't use the components as it is. Components need to be modified in some way to match the application architecture or the other components. The process of changing the component for use in a particular application is often referred to as component adaptation. It helps in faster development of the product, also increases reusability. Let us discuss the above topics in more detail.

# Aspect-Oriented Programming

## Overview

- It is a programming paradigm that provides ability to modularize the representation of cross cutting concerns to maximize code reusability and solve the code-tangling problem.
- Synchronization, scheduling, logging, security and fault tolerance are some of the examples of cross-cutting functionalities.
- Implementation of these functionalities is scattered across multiple classes and they cut across the natural units of modularity, thus causing code-tangling.
- So AOP solves the above issues by decomposing a problem into functional and aspectual components. It uses weaving mechanism to combine them back.
- It has brought solutions to the many shortcoming of Object oriented programming. In object-oriented programming languages, crosscutting concerns are spread throughout the program in an undisciplined way. So, they can't be reused. It becomes difficult to refine or inherit them. As the dependencies between the cross-cutting concerns and other components are not specified, the interaction between the cross-cutting concerns and other top level components of the system will be hard to represent.
- The motivation of Aspect oriented programming is to eliminate code scattering and tangling. Aspect oriented technologies improve the modularization of software systems by two means: Using new constructions for the suitable encapsulation of cross cutting concerns into single modules and using mechanisms for composing cross cutting concerns with base modules.

## Concepts and terminologies in AOP:

- **Concern:** It is a set of information that has an effect on the code of a computer program. The details of a database interaction can be considered as a general concern whereas performing a complex calculation is a specific concern.
- **Cross cutting concerns:** Cross cutting concerns cut across the traditional boundaries of abstraction (like classes and modules) and affect other concerns. They cannot be completely isolated from the rest of the system neither during design nor during implementation.
- **Separation of Concerns (SoC):** For reducing the complexity and improving the reusability, the computer program is separated into different sections where each section addresses a distinct concern. This helps in defining the concerns in a single location and using them throughout the program. A program that uses Separation of concerns is called a modular program.
- **Aspect Weaving:** The process of inserting advices into a running program.
- **Scattering:** If the code of a concern is spread over different modules, then the implementation of the concern is said to be scattered.
- **Tangling:** If the code of a concern is mixed with the code of other concerns, then the implementation of the concern is said to be tangled. Thus, there can be significant dependencies between systems and difficulty in reading the code.

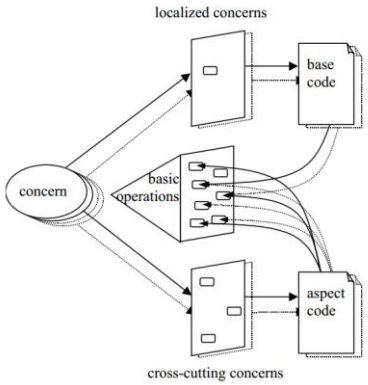
## Programming Constructs:

- **Join Point:** It is point of execution in a program where the aspects and the functional code are intermixed together. These points can be at object construction, method entry etc.
- **Point Cut**: It is a set of join points. They are specified using class names, method names or a regular expression which matches the class name or method name.
- **Advice**: It defines an action to be taken at a joint point. before(), after(), around(), after returning(), after throwing() are some of the advices.

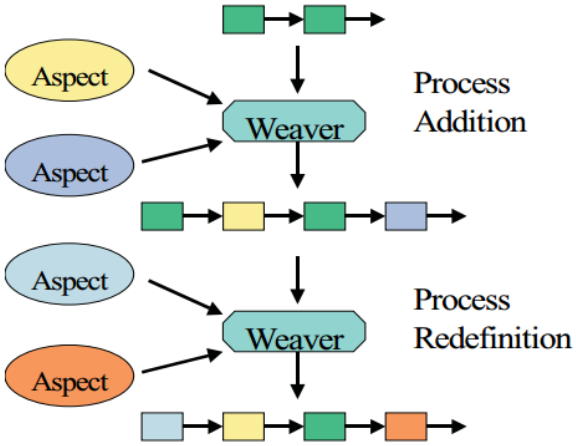
- **Aspect**: A crosscutting type that contains the definitions of advice, pointcuts, and method declarations. It does not contain the program's core functions.

## Implementation:

1) The designer of an aspect-oriented program would identify the functional and cross-cutting (aspects) components and separate them.




2) When the code is being executed, weave aspect and functional features.
   Weaving can be done in two ways:
   - **Implicit weaving:** A complex behavior implicitly results by directly forwarding events from a State chart to the another.
   - **Explicit weaving:** A complex behavior is explicitly depicted by a specific state machine whose states are compound states drawn from the Cartesian product of the sets of states of the component machines, and state transitions are sequences of state transitions taken from the component machines.




## Use-Case:

- **Transaction Management**: In enterprise application, around() advice can be used around request processing method. If it fails, that particular transaction can be rolled back. To restrict the access to a method for a set of users, before() advice can be used to verify the credentials. Similarly, we can limit the access for a particular user.

- **Exception Wrapping**: We can wrap an exception into a common exception object and throw it to the top level layers. After throwing() advice can be used to do this task.

❖ AOP is one of the fundamental technologies used in **Spring Framework**.

### Advantages :

We can summarize the advantage of Aspect Oriented Programming as- Quantification (or lack of scattering), where the code for each concern will be located at a single place which eases the maintenance of the system and Obliviousness (or lack of tangling), which makes it easier to read.

### Disadvantages :

- **Maintenance and Debugging:** In AOP, we have a code running but we don't have any clude is it getting called. If the advices cause some changes then it becomes tough to understand where it is getting changed. It makes debugging difficult.
- **Code bloat:** Small source can lead to much larger object code as code is "weaved" throughout the code base.

## Software Product Line (SPL):

- A set of products which would address the needs of a specific market and that are generated from reusable assets are referred to as Product lines. They have been widely adopted in software industry for multiple purposes. Software - intensive systems that are likely to share a common maintained set of characteristics that specify the need of a particular market and that are developed from a archived set of core assets in a prescribed way are regarded as Software Product Line.
- The main logical existence of software product line is the enhanced development of its core, reusability in terms of its assets and the degree to which the actual application defers to any other normal applications. During the development of the application, platform software is selected and it is configured in order to meet the requirements of that application.
- A typical software product line includes three main essential activities:
    - Core asset development
    - Product development
    - The management
- Here product development is built and implemented using the core asset development under the management which can be either organizational or technical. The development of the product from the core asset can occur in any order - either new product is built using the core assets or new core assets are developed from the existing product.
- Moreover, a feedback loop mechanism between the product development and the core asset development would add more value to the efficiency of the system. The core assets would be updated, as and when new products are being developed. The use of the core asset is noted and being fed back to the core asset development team.

# ACTIVITIES IN SOFTWARE PRODUCT LINE

## Core Asset Development Activity:

This activity is mainly used for finding and analyzing the production capability of the products. Here, no one way relation between the output and the context exist however, activities of producing the core asset could completely change the context.

Core asset development takes place with the context of the existing constraints and resources. The context has influence on how the core asset development is carried out and the nature of the output produced. There are four important contextual factors:

- Production Constraints
- Constraints in the product
- Pre-existing assets
- Production Strategy

The three outputs of the core asset development which are required to develop products are

- Core asset base
- Product line scope
- Production plan

## Product Development Activity:

The inputs for this activity product development are:

- Description of the product that outlines the product line scope.
- Feasible scope to include the product under the product line.
- Production plan, which contains details as how the core asset must be used
- The core asset from which the product is developed.

The product builders would provide their feedbacks related to the any issues / drawbacks of the core asset which could arise during product development.

## Management Activity:

- The different activities under the management activities are
    - Assigning the various resources in the organization
    - Coordinating and supervising the resources.

- Both the technical and organizational levels of the management must be committed together the software product line level. Technical management decides the production method and also makes sure that those who develop the core assets and products are engaged in their activities; also they are following the process that are defined for the product line and collect sufficient data to track their progress.

- Organizational and technical management contributes to the core asset by making available the management artifacts that were used in developing products in the product line for reuse.
- Organizational management identifies the constraints of the production and determines the production strategy. It should also create an organizational structure that makes sure that the organization units receive the right resources like trained personnel.

## PROS AND CONS OF SPL:

Pros:

- Enhances the profitability to both the organization as well as customers.
- Assists in maintaining the customer satisfaction (since the product is only successful if the customers are satisfied).
- Better quality is assured that keeps the productivity and customer satisfaction high as well as maintains the organization to be an active player in the market.
- It makes use of the human resources smartly and efficiently.
- It is beneficial to the stakeholder of the company. Software product line approach would lead to quicker development of the product that drastically increases the productivity rate. Software developers would have much more job satisfaction than expected.

Cons:

- Since the products would depend on the previous versions, there are chances that there would be an increase in the complexity of the product – such as creating dependency where the reusable components made use of for generating the newer versions of the products.
- When products are merged, the scope would also proportionally increase. However, this increase in scope would further lead to spike in complexities, thereby reducing the effectiveness of the system.
- This approach possesses the risk of investment since the organization cannot get immediate results.
- Moreover, it is a time taking process that requires long term management.
- Scalability issues - Special units must be created if employees of an organization are increased, reorganization is necessary.

# Component Adaptation

## What is Component Adaptation?

In past decades, object-oriented software development has achieved immense success but it has not achieved extensive reusability and maintainability because individual classes are too detailed, specific, and bound to application domains. As a result, Component-oriented programming emerged. It aims on creating reusable components that can be then used for component based application development. A component in software development process is basically a module consisting of both code and data and gives an interface that can be called by other components. Once we have reusable components, application development process becomes an easy task with just three steps of selection, **adaptation** and composition of these components rather than implementing the application from scratch.

The naive view assumes that to implement any new application a set of components can be put together by connecting inputs to outputs and it is ready to use. However, researches in Software reuse has shown that 'as-is' reuse of a component is very rare, and the components are generally needed to be changed in some way to match the application architecture and other related components. This process of changing the component for use in a particular application is knows as **component adaptation.**

## Requirements for component adaption techniques:

• **Transparency**: The component adaptation should be transparent which means both the user of the adapted component and the component itself are not aware of the adaption happening between them.

• **Black-box**: Before the component is released to be reused, the engineer should always develop a mental model to describe the functionality of a component and that model should be small and simple because the adaption techniques only require knowing the interface of the component and not the internal structure of the component.

• **Composable**: The component adaptation technique should be easily composable with the component it is applied to such that there is no need to redefine the component. Moreover, the adapted component should also be composable with other components as it without the adaptation.

• **Configurability**: Adaptation techniques can be useful and reusable if it provides sufficient configurability of the specific part i.e. which selectors should be replaced with what name in the adaptation type changing operation names.

• **Reusability**: It should provide reusability of the adaption type and a particular instance of the adaptation type, Both the generic and specific part, since these two aspects are intertwined and generic part cannot be separated from specific part and vice versa.

## Conventional Component Adaptation techniques:

Component adaption can be achieved in several ways; there are two conventional techniques that are as follows:

1. **White-Box-** white box techniques require the software engineers to have a thorough knowledge of the internals, and then adapting a reused component either by changing its internal specification or by overriding the excluding parts of internal specification.
   Example of white box-
   - **Inheritance:** Inheritance provided by e.g. Smalltalk-80 and C++, makes the state and behavior of the reused component available to the reusing component. Depending on the language model, all internal aspects or only part of the aspects become available to the reusing component. All the methods and variables defined in the parent superclass become available to the subclass(child). Hence, it provides the benefit that the code exists in one location only.
     However, the disadvantage is that the software engineer must have detailed knowledge about the internal functionality of the superclass, when overriding superclass methods and when defining new behavior using behavior defined in the superclass.

   - **Copy-Paste:** Copy-Paste technique is used when an existing component provides some similarity with the component needed by the software engineer, in that case the engineer just need to copy the code of that part of the component that is suitable to be reused in the component under development. After copying the code, the software engineer will often

make changes to it to make it fit the context of the new component and additional functionality will be defined or copied from other sources.
Although this technique provides some reuse, it has many disadvantages, one among those is that there are multiple copies of the reused code that are existing and the software engineer has to understand the reused code, thoroughly. This technique only fulfils transparency requirement, and other requirements such as black-box, composable, configurable and reusability are side lined, since there is no encapsulation boundary between component code and adaptation code, the two codes are merged.

2. **Black box-** The black box techniques on the other hand doesn't rely on the internal specification and only requires the software engineers to have knowledge about the interface only.
   Example of black box-
   - **Wrapping:** wrapping black box techniques, declares components under one encapsulated component umbrella called the wrapper, but it only has the functionality for passing the requests for small changes from clients to the wrapped component. Another thing is aggregation, there is no clear distinction between aggregation and wrapping. Aggregation on one hand is used to form new functionality out of already existing components that provide necessary functionality however wrapping on the other hand is used to adapt the behaviour/functionality of the component enclosed in the wrapper.
     The main disadvantage of wrapping is that its implementation is an overhead, because complete interface of the wrapped component needs to be handled by the wrapper, including those interface elements that need not be adapted. Moreover, this may lead to excessive amounts of adaptation code and serious performance reductions.

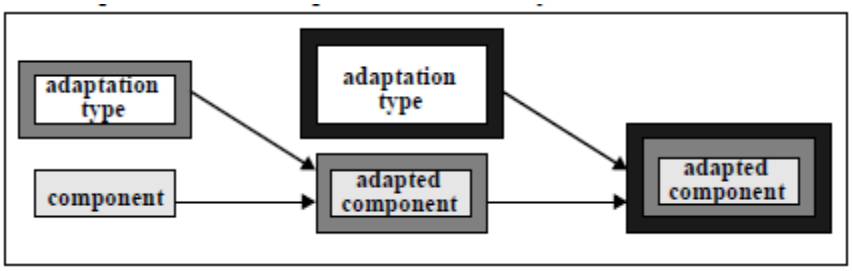
## Evaluating the Conventional Adaptation techniques:

On evaluating the conventional adaptation techniques, we can see only few requirements have been fulfilled by each of the techniques mentioned above, Hence we need an alternative approach which will fulfil all the requirements that are required for an effective component-based software engineering. One Solution is Superimposition.

| Requirement | Copy-Paste | Inheritance | Wrapping |
|---|---|---|---|
| transparent | + | + | - |
| black-box | -- | - | + |
| composable | - | - | + |
| configurable | - | - | - |
| reusable | - | - | +/- |

## Component adaptation through Superimposition:

Superimposition component adaptation techniques is based on the principle that a component and the functionality adapting the component are two separate entities on one hand and are very tightly integrated on the other hand. An object superimposition S of B over O is defined as the additional overriding behavior B over the behavior of a component object O. Different from traditional inheritance techniques, a single unit of superimposed behavior can change several aspects of the basic component's behavior.




### Use-Case:

Consider an e-commerce website that would involve separate component for inventory management, order management, user management, payment, fraud and so on. To develop a composite system that incorporates all those components and give a well-defined consolidated view of e-commerce is one of the main core of component adaptation.

### Advantages:

- It reduces software development costs.
- It speeds up software development.
- It reduces the maintenance burden associated with the support.
- Eases system upgradation.

### Disadvantages:

Due to the availability of components and the diversity of target applications, mismatches between pre-qualified existing components and a particular reuse context in applications are often inevitable and have been a major hurdle of component reusability and successful composition. Although component adaptation has acted as a key solution for eliminating these mismatches, existing practices are either only capable for adaptation at the interface level, or require too much intervention from software engineers.

## Conclusion

Aspect-oriented programming modularizes the code and decompose the complex problem. It solves major issues such as scattering and code tangling. AOP provides modularity, reusability and higher level of abstraction. It makes the code more readable and manageable. Component adaptation increases reuse thereby allowing faster development, more robust code and less redundant effort. It reduces the software development cost. Software product lines are emerging as a viable and important development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers. Software product line engineering can also enable rapid market entry and flexible response, and provide a capability for mass customization. A software product line epitomizes strategic, planned reuse. Using Aspect-oriented techniques and Component Adaptation improve the usability of the Software Product Line. Hence to make software development better, easier and cheaper we should leverage the features provided by these approaches.

## References :


- https://en.wikipedia.org/wiki/Aspect-oriented_programming
- Jan Bosch. *Superimposition: A Component Adaptation Technique*. Citeseerx
- Binary Component Adaptation - Ralph Keller and Urs Hrlzle 1
- A formal approach to component adaptation Andrea Bracciali a, Antonio Brogi a,*, Carlos Canal b
- Raphael; Cecília M. F.; Leonardo P.; Felipe N.; Leonardo Montecchi; *Evolving a Software Products Line for E-commerce Systems: A Case Study*; ACM
- Katharina Mehner; Mark-Oliver Reiser; Matthias Weber; Applying Aspect-Orientation Techniques in Automotive Software Product-Line Engineering
- http://www.jot.fm/issues/issue_2004_03/column6.pdf
- http://www.sei.cmu.edu/productlines/research/