Advanced Modularization, Aspects, and Their Application in Software Product Lines

Aspects around us

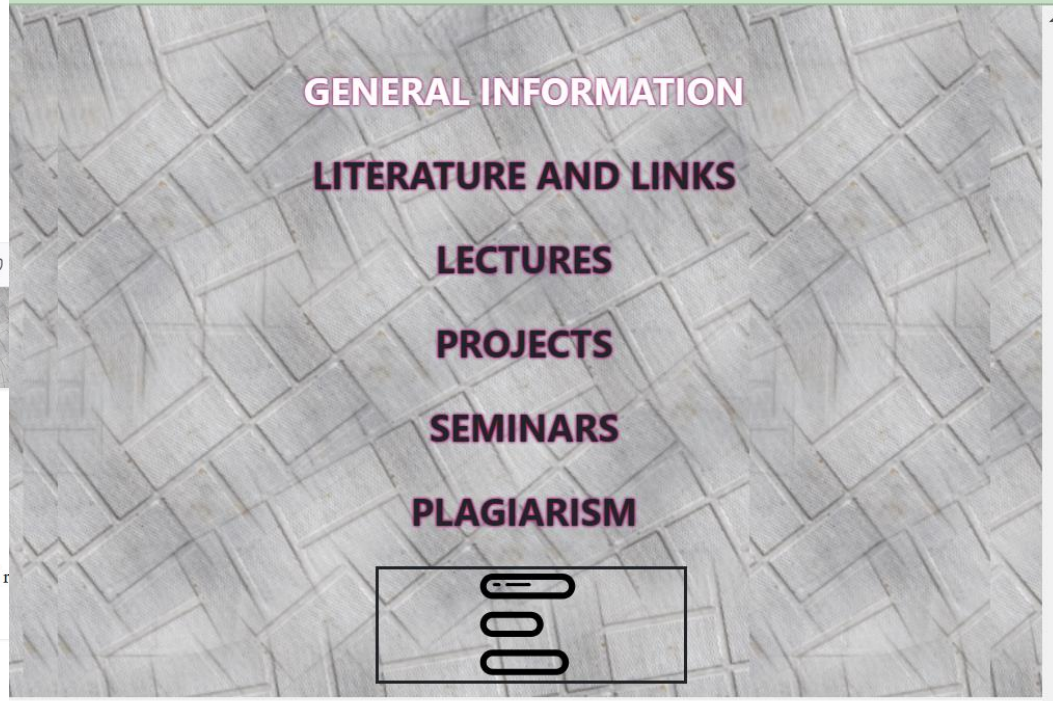
Ing. Jakub Perdek About the author

https://jakubperdek-26e24f.gitlab.io/

I am a doctorand focused on reuse in the area of software product lines at the Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava.
I have membership in AdvanSD research group.

- Research Fields
  - **Software product lines (dissertation):** *Annotation based, Aspect oriented, and model driven.*
  - **Information retrieval (master thesis):** *Similarity metrics, LDA, indexing, BIG Data (Hadoop, PIG, Hive), etc.*
  - **Machine and Deep learning (bachelor thesis):** *Their application.*
  - **Data structures and algorithms:** *Experimenting with algorithms applied to different technologies.*
  - **Computer Graphics (side):** *Its use in Web development.*

Web Page of the subject



Literature and Links

General

Aspect-Oriented Software Association: https://aosd.net/

Programming conference: https://programming-conference.org/

ooo

AspectJ

Home page of AspectJ: https://eclipse.dev/aspectj/

AspectJ Development Tools (AJDT) plugin for Eclipse: https://www.eclipse.org/ajdt/

Aspect Oriented Programming: Radical Research in Modularity: Kiczales' Gregor Lecture at Google

Capabilities, Basics, Advanced topics and Application of AspectJ: The AspectJ in Action *Laddad, Ramnivas, 2003. AspectJ in programming. Greenwich, CT: Manning. ISBN 978-1-930110-93-9.*

ooo

Lectures

The scheduled lectures prepared for this subject with available materials and r
are:

Lecture 1:
Advanced Modularization, Aspects, and their Application in Software Product Lines

Lecture 2:
Advanced Topics in Aspect-oriented Languages: AspectJ Programming Language

A List of Submissions and deadlines

You have to submit during semester following items:

- **[up to 21.10.2024] Project objective: 200-300 words can be more**
- **[up to 03.11.2024] Mini project: Aspect-oriented software development using Theme, patterns, and AspectJ**
- **[up to 18.12.2024] Report in research article template (not mandatory but recommended) as PDF containing at least 8 pages.**
- **[up to 18.12.2024] Artifacts created during project (everything what is created to support final outcomes as code, diagrams, figures, graphs, etc.)**

https://jakubperdek-26e24f.gitlab.io/aosd.html

Aspect-Oriented Software Developement

Taught by: Doc. Ing. Ján Lang, PhD., Ing. Jakub Perdek

Asistent: Ing. Jakub Perdek, Ing. Viktor Matovič

Study type: Master's

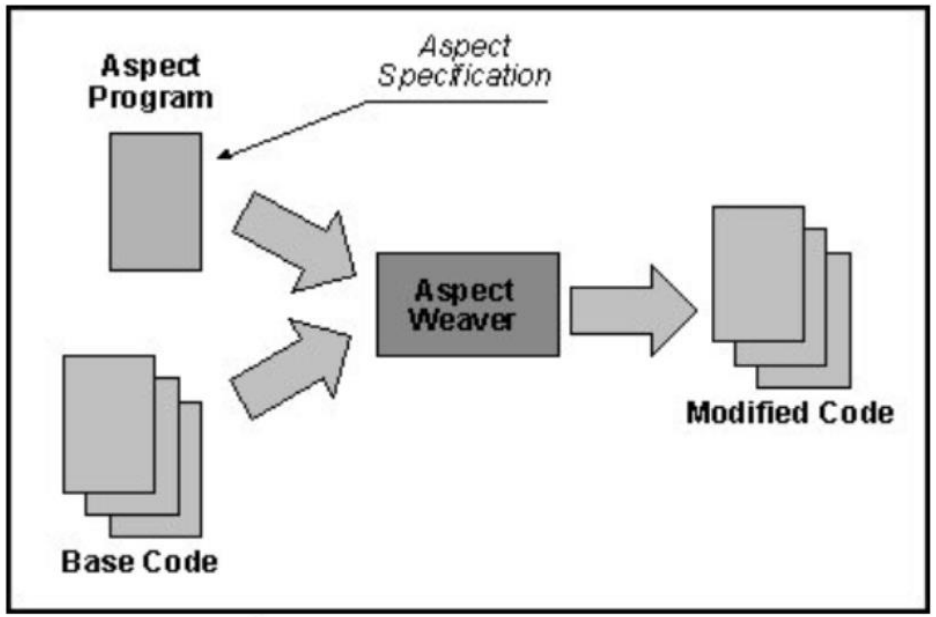
Study program: Intelligent Software Systems

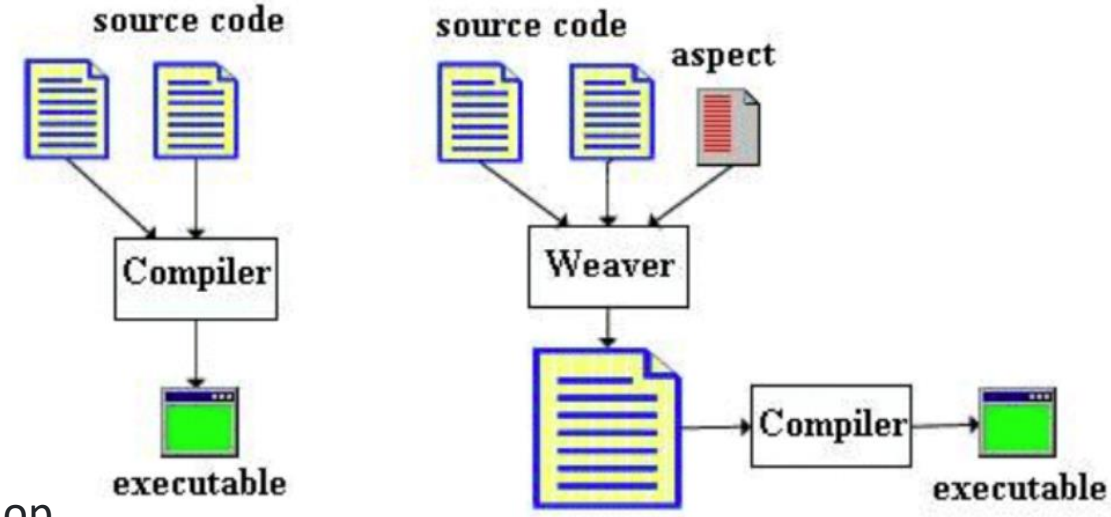
Term: Winter

Weekly hours (lectures–exercises): 2-2

Completion: Exam

Credits: 6



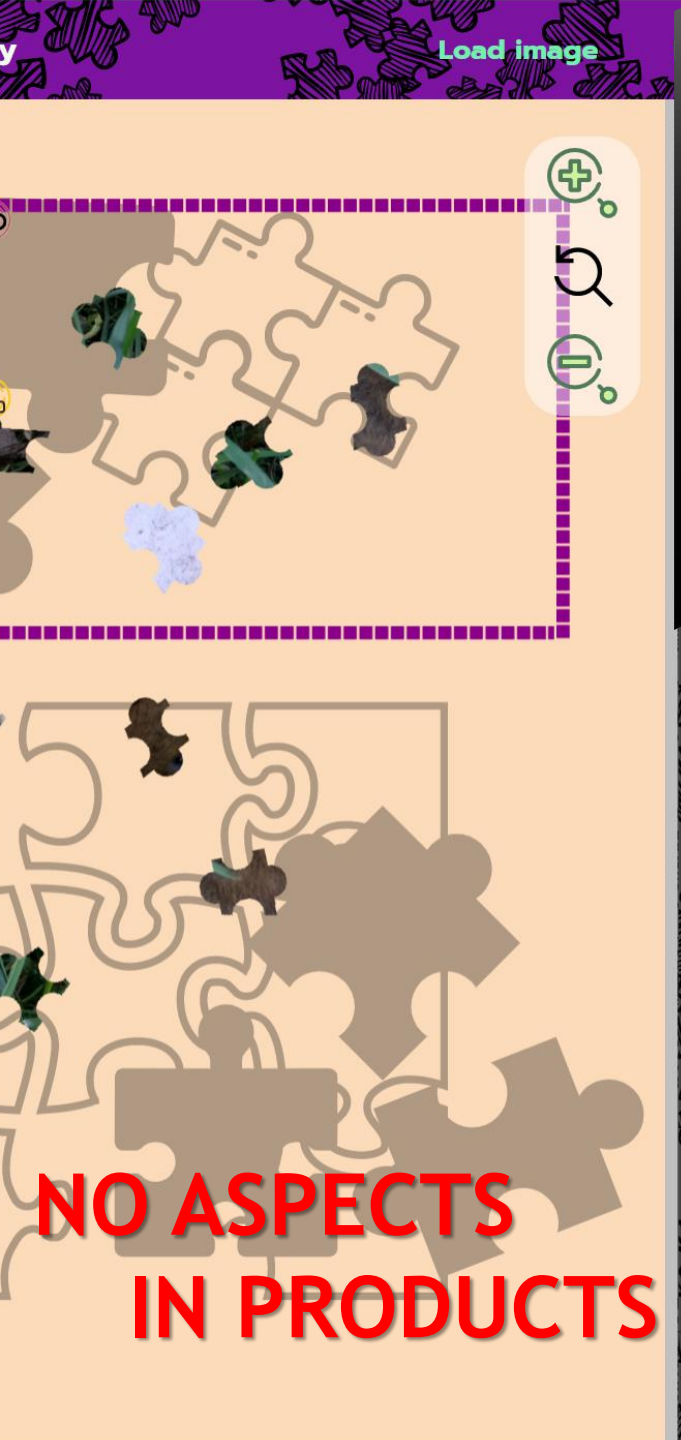


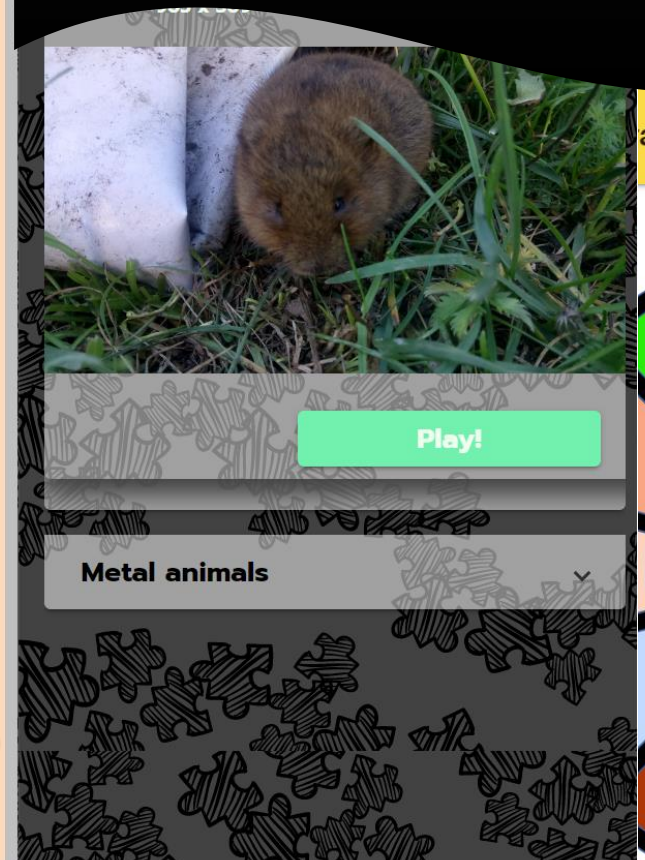
Source: https://sites.google.com/site/sites/system/errors/WebspaceNotFound?path=%2Fjavatouch%2Fintroductiontoaop

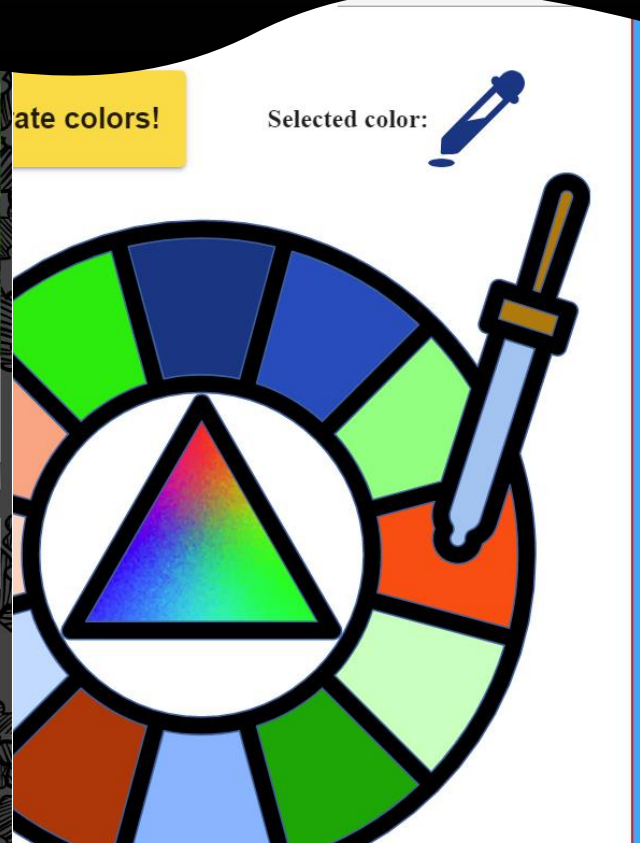
What it will about?

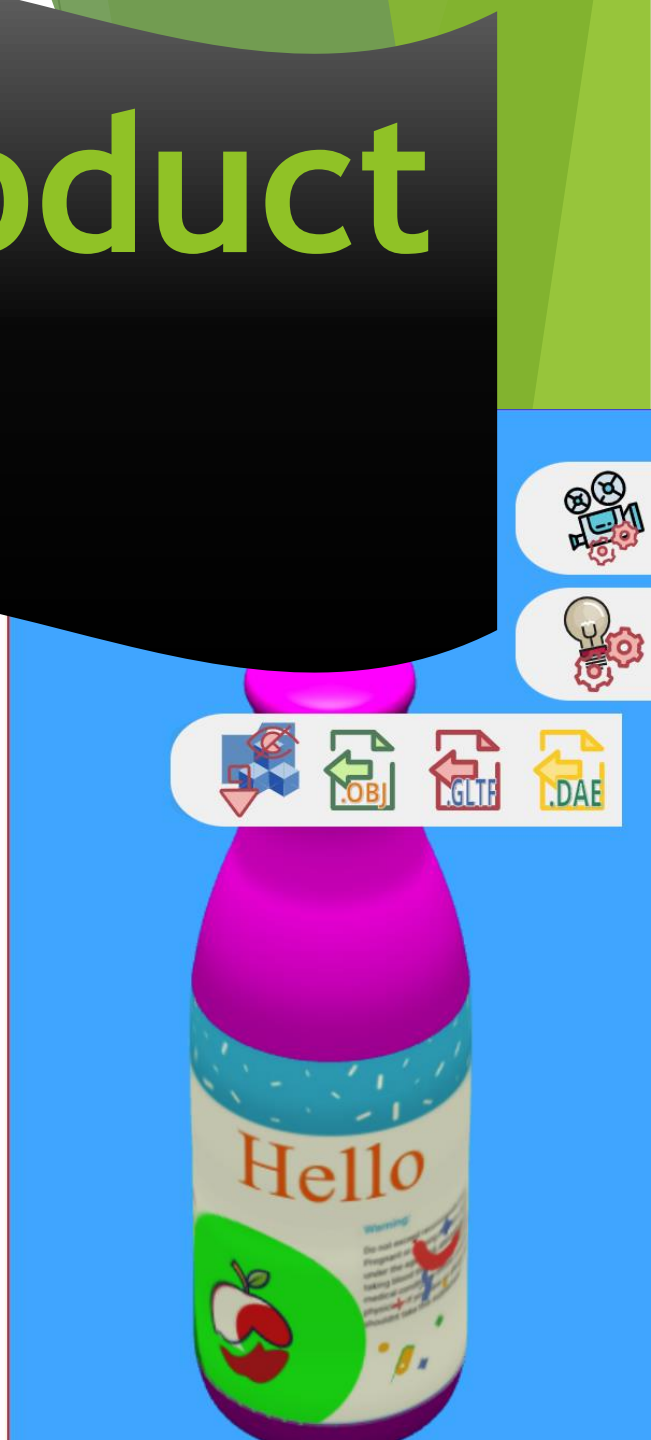
- Aspect-oriented programming
  - Its features in various languages (Java, JavaScript/TypeScript, C, C++, Python, DSL (domain-specific languages), …)
  - Implications to use-cases
  - Its use across the whole development cycle (analysis, design, implementation, testing, maintenance)
  - Benefits and drawbacks of aspect-oriented programming
  - Its use in advanced modularization + cases
  - Its use in software product lines as complex systems
- Advanced modularization
- Software product lines as complex systems
- Variability handling (in annotation based software product lines)
  - How to evolve variable features independently using aspects

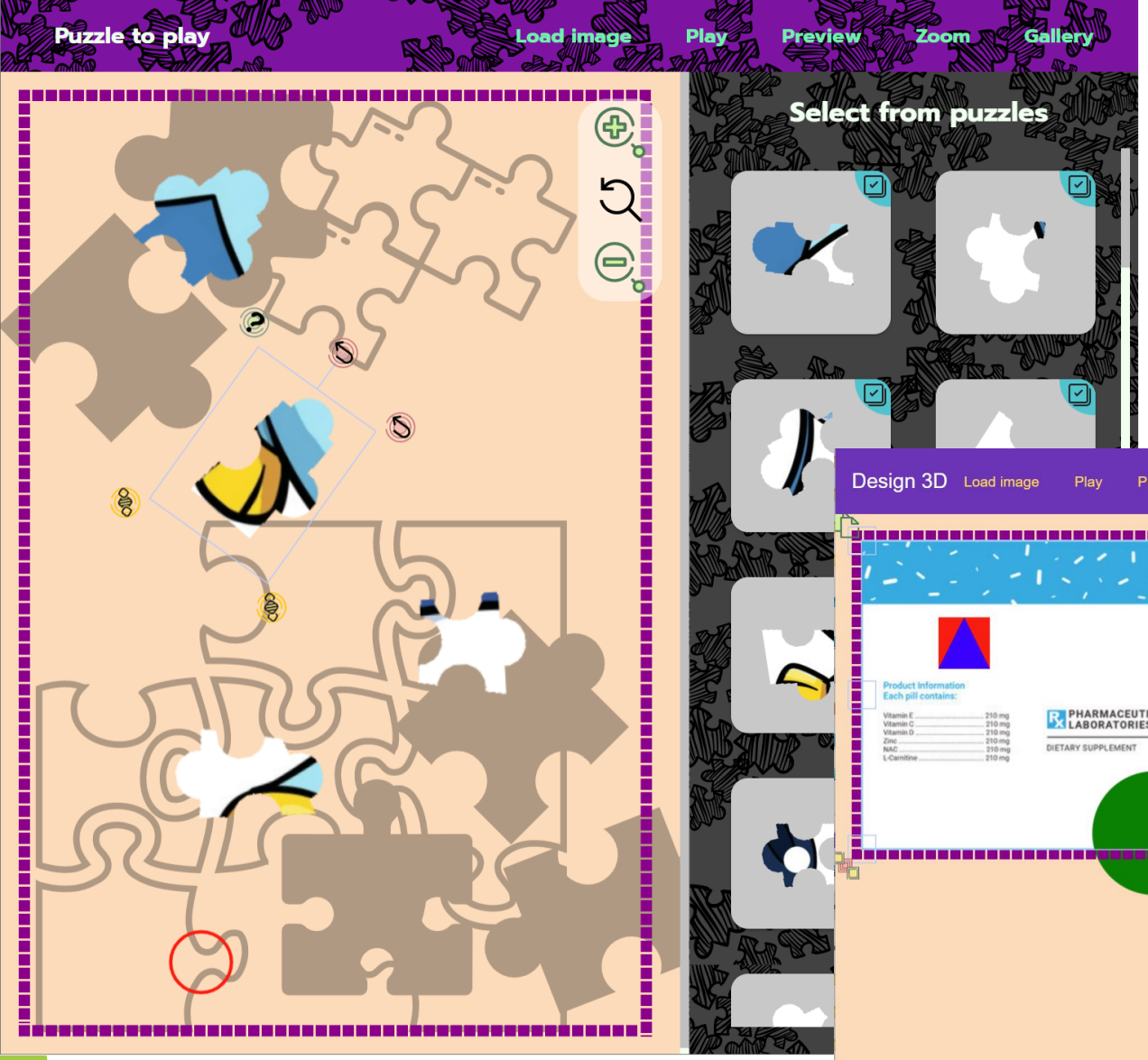
Software product lines



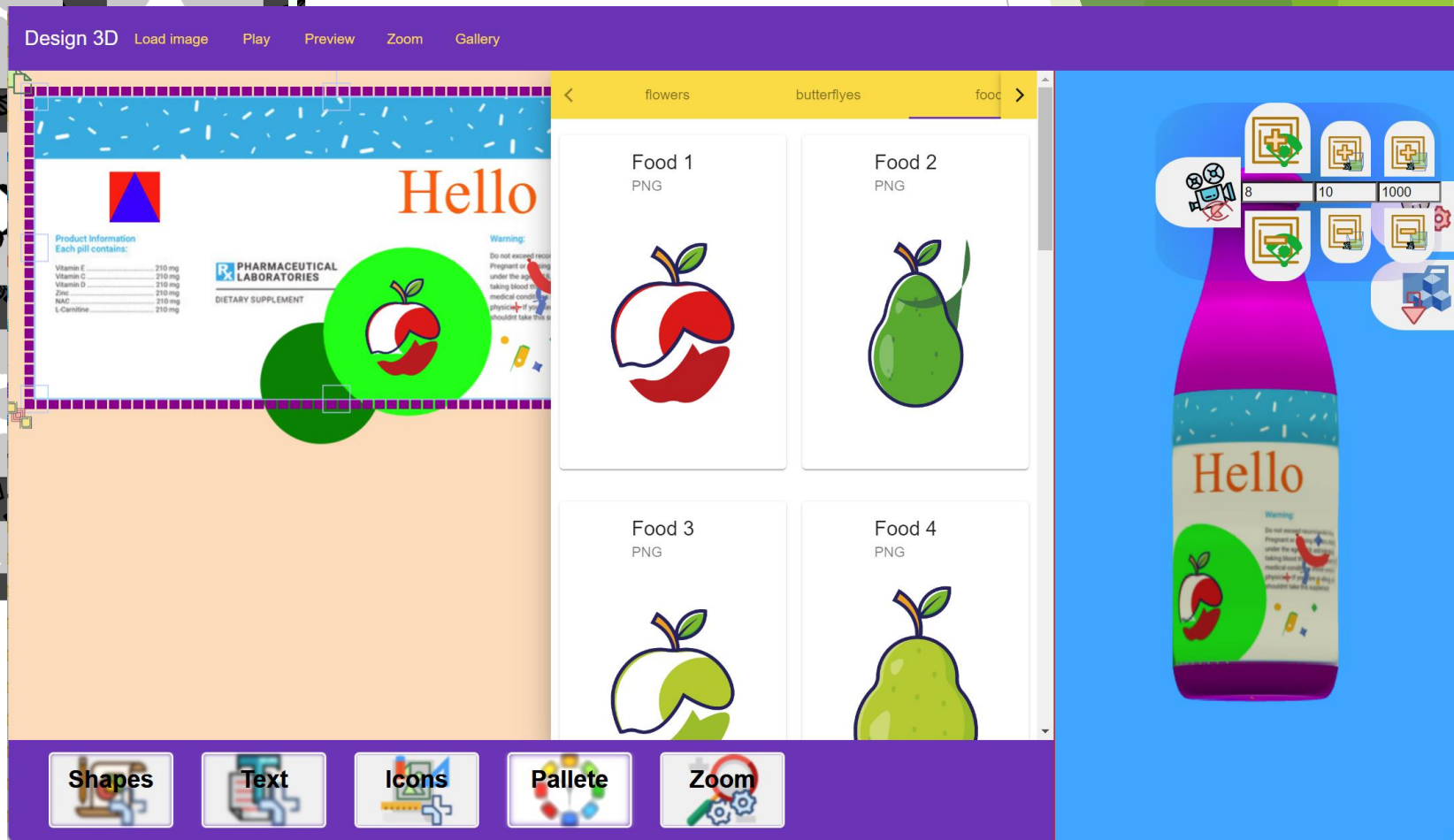








Commonality vs. Variability



Puzzle app. vs. Desing app.

Previous page of the course

- http://www2.fiit.stuba.sk/~vranic/aosd/index.html

Projects

A List of Submissions and deadlines

You have to submit during semester following items:

- **[up to 21.10.2024] Project objective: 200-300 words can be more**
- **[up to 03.11.2024] Mini project: Aspect-oriented software development using Theme, patterns, and AspectJ**
- **[up to 18.12.2024] Report in research article template (not mandatory but recommended) as PDF containing at least 8 pages.**
- **[up to 18.12.2024] Artifacts created during project (everything what is created to support final outcomes as code, diagrams, figures, graphs, etc.)**

Mini Project

Mini project: Aspect-oriented software development using Theme, patterns, and AspectJ

Specify, desing and implement small aspect-oriented project with Themes patterns and AspectJ primarily as part of your semestral project if it is feasible:

Prepare following Themes DOC/UML: 5 points

- 1. The basic view as an annalytic model Themes/DOC that captures themes and relationships. Two themes are sufficient. **1 point**
- 2. The transformation from basic view (previous point) into crosscutting view inn Themes/DOC notation. **1 point**
- 3. The design model in Themes/UML notation with specified relation between themes. You may use the indirect relationships known from the JPDD notation. **3 points**

Prepare following aspect-oriented code written in AspectJ: 5 points

- 1. The aspect with all necessary classes provided as Mock. The code has to be compillable. **2 points**
- 2. The application of autochtonous (native) aspect-oriented design pattern. **2 points**
- 3. Explanation of the applied pattern. The contradicting forces and the way how are resolved should be mentioned. **1 point**

Project Assessment

Project Assessment

The semestral project and mini project fully for **50 points** will be assessed according to following criteria:

- **Project objective – 5 points**
- **Part on aspect-oriented software development using Theme, patterns, and AspectJ – 10 points**
- **Project report and acompanying artifacts – 35 points**
  - Results (what have you actually done) – 15 points
  - Scope (the problem you've chosen shouldn't be trivial) – 4 points
  - Presentation (how well are the results presented in the report) – 4 points
  - Interpretation (how well you explained the meaning of what you've done) – 6 points
  - Comparison (how well you compared your results to what others have done in that area) – 6 points

Seminar Presentations - Demonstrate Progress of Your Work!

The length of your presentation is restricted to **12 minutes**. Ten minutes is usually enough, but a presentation that is too short can result in omitting essential parts. The content is determined by the actual week of the semester. The later you present, the more content, such as your implementation, diagrams, and results are expected.

The presentation itself and discussion can be in Slovak or English.

Please submit your presentation to AIS a day before you present, and notify your discussants, at least with the abstract of your work, to acquaint them with the topic that will be presented.

The context of the presentation should include:

- The overview of a current state in the focused area/about solved problem.
- How does your intended contribution/chosen topic relate to this subject (aspect-oriented programming and/or software product lines).
- Present your own contribution/ideas. (recommended in all project phases - highly praised)
- Show the supporting code (if any) and/or application for the introduced contribution.
- Visualize what is done. Use diagrams, tables, use cases, etc. (later phases).
- Provide evaluation of the results, capabilities, and outcomes. Do not forget to discuss them.
- Compare your work to what others have done if it is possible.
- Do not forget to mention what your methodological contribution lies in. Additionally, prepare research questions (two are enough).
- State how your semestral work will continue - future work. (You can check if what you will provide/provided is enough.)

Total: 15 points

Discussions After Presentations

Discuss weak and strong aspects of the presented work! Recommend improvements or further direction.

Each seminar presentation will have at least two discussants asking for more details, discussing weak and strong aspects of the presented work, and recommending improvements.

The content of a discussion is assessed according to the occurrence of the following cases:

- The adequate addressability of a discussed topic.
- Following and extending discussed ideas if possible.
- Taking into account alternative directions in solving/handling presented problems or getting more details about them.
- At least three high-quality questions, including adequate addressability and reactions during the discussion.
- Inclusion of aspect-oriented topics such as weaving, obliviousness and quantification, separation of crosscutting concerns, pointcuts, etc., into questions.
- Questions should not be too familiar or subjective.
- Points out possibilities for evaluating presented contributions.

Do not forget to participate also in the following general discussion. Comment and ask about presented topics.

Each discussion: 3,5 points

Number of discussions: 2 times

Total: 7 points

Assessment Summary

The maximum number of points for presentations and discussions is the following:

Presentation

Presentation - document: 6 points

Presentation - oral: 7 points

Discussion

Discussions: 7 points

Overall Assessment

- **Projects** - **50 points** (project objective + semestral project + mini project)
- **Seminars** - **20 points** (discussions and presentation)
- **Final exam** - **30 points**

100 points

Lecture 1: Advanced Modularization, Aspects, and their Application in Software Product Lines

Lecture 2: Advanced Topics in Aspect-oriented Languages: AspectJ Programming Language

Lecture 3: Aspect-Oriented Design Patterns and Their Use For Advanced Modularization

Lecture 4: Aspects in Analysis and Design: Theme and JPDD

Lecture 5: Techniques and Technologies in Variability Management: From Conditional Compilation, Frame Technology, Framed Aspects, Lightweigth Method towards Aspect Free Products

Lecture 6: Aspects and Use Cases

Lecture 7: Component and Composite Approach to Aspect-Oriented Programming: Application to Software Product Lines

Lecture 8: Automated Complexity-Optimized Lightweight Variability Handling with Expressed Feature Models in Code: Applications in Software Product Line Evolution

Lecture 9: Supporting Reuse With Aspects

Lecture 10: [Guest Lecture] Ing. Oliver Udvardi: Optimalizácia vývoja softvéru pomocou softvérových produktových línií

Lecture 11: Discussions and Final Presentations (optionally topics from Software Product Lines)

Lecture 12: Discussions and Final Presentations (optionally topics from Software Product Lines)

Plagiarism

- Taking each other's work, which is not yours as your own, is known as plagiarism. It's a forbidden practice. Cases, where such practices are detected will be sent for further investigation and resolution by a disciplinary commission according to valid rules. These rules apply in this subject and disciplinary process.

Resolving Plagiarism Cases In This Subject

- **Cheating on tests:**
  Disqualification from the test and assigning zero points.
- **Cheating on exams:**
  Disqualification from the exam and granting FX from the whole subject.
- **The parts of unreferenced work in the project which are not authors':**
  Rejection of submitted project and granting FX if elaboration of the project is mandatory from the whole subject. Plagiarism is not tolerated in any range.

Disciplinary Commision

Disciplinary violation will include the participation of the dean of faculty in its negotiations and resolution with a disciplinary commision of this faculty. The disciplinary commission then optionally in accordance with taken conclusions, gives recommendations to the dean to:

- give a guilty student a rebuke
- conditionally disqualify a guilty student from the faculty
- disqualify a guilty student from the faculty

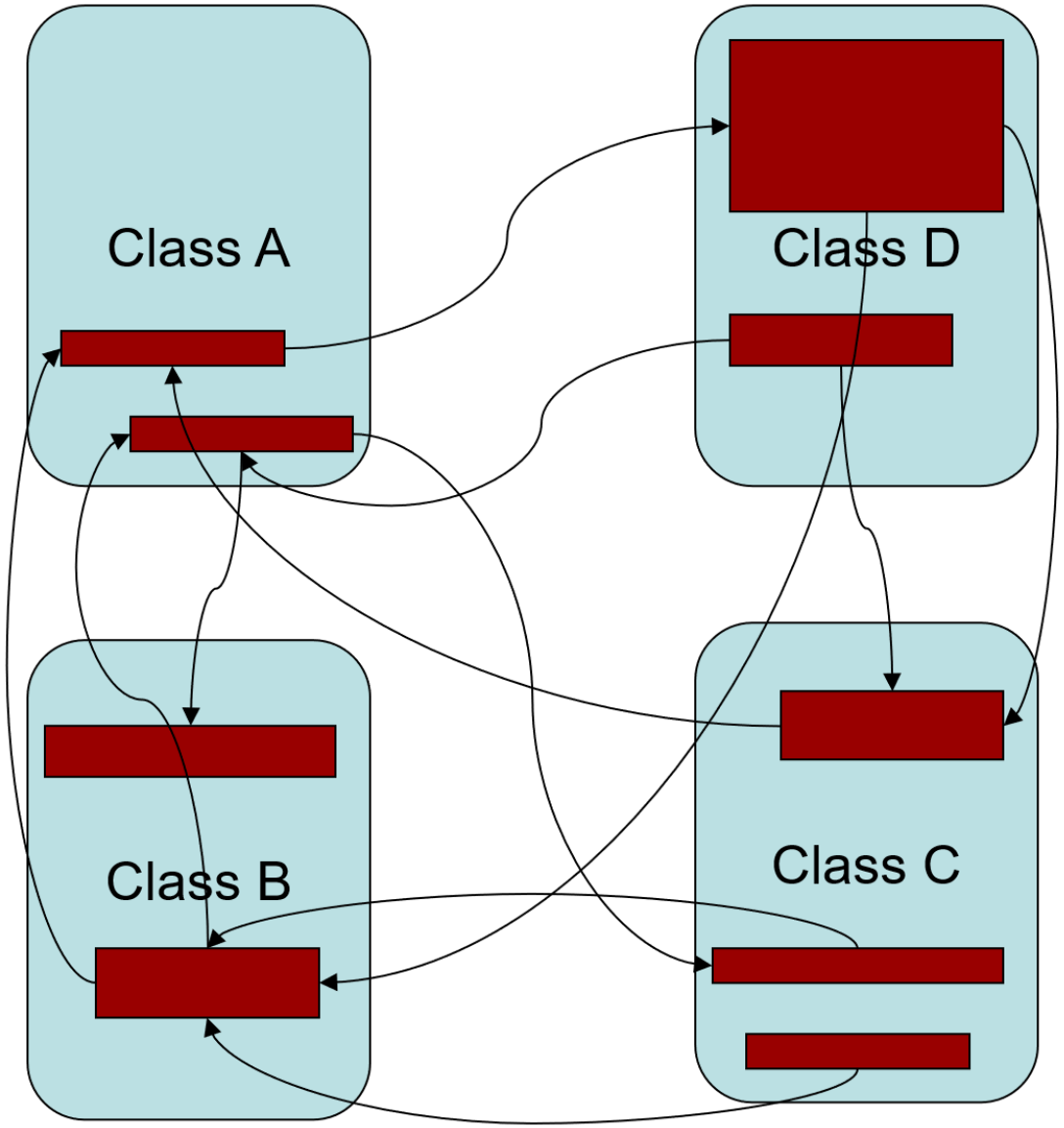
Plagiarism Prevention / Guidelines

- Guidelines such as UC Davis demonstrate actions and practices to get around plagiarism. The most important of them are:
  - Work continuosly and independently.
  - Check your work after some time (two days or a week) and improve formulations.
  - Do not use ChatGPT or any artificial intelligence for paraphrasing.
  - Take brief notes from other materials along with used sources.
  - Reference paraphrased work with reference in each sentence where it is used. Reference others' work correctly in a way that is recognizable from yours (references at the end of the block probably do not conform to this).
  - Share information about your work to such an extent that some details will not be provided. For example, talk about code but not show all its fragments or provide the whole project to others.
  - Do not apply minor/cosmetic changes to other's work (replacing words with their synonyms, changing the order of sentences, etc.) or give credit to it if the work is the same in substantial aspects as yours.
  - Do not cheat and respond to others on tests, except to the personnel from the subject

Advanced Modularization, Aspects, and Their Application in Software Product Lines

Aspects around us

[C++, Java, Smalltalk, Python,…]

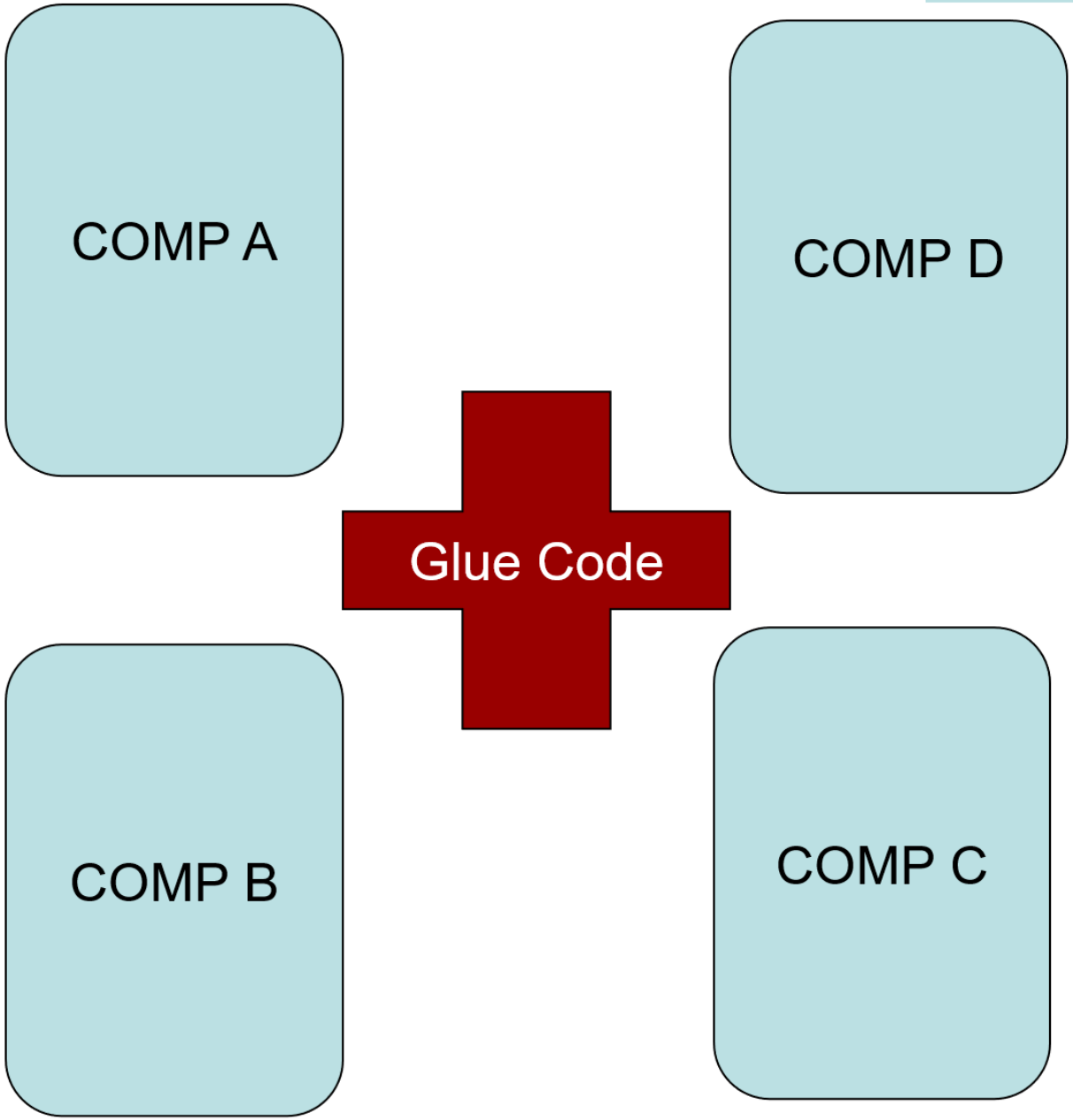


What is the main problem?

REUSABILITY

Source: http://ssel.vub.ac.be/jasco/lib/exe/fetchb6e2.php?cache=cache&media=documentation%3Ajasco-vub-session1.ppt

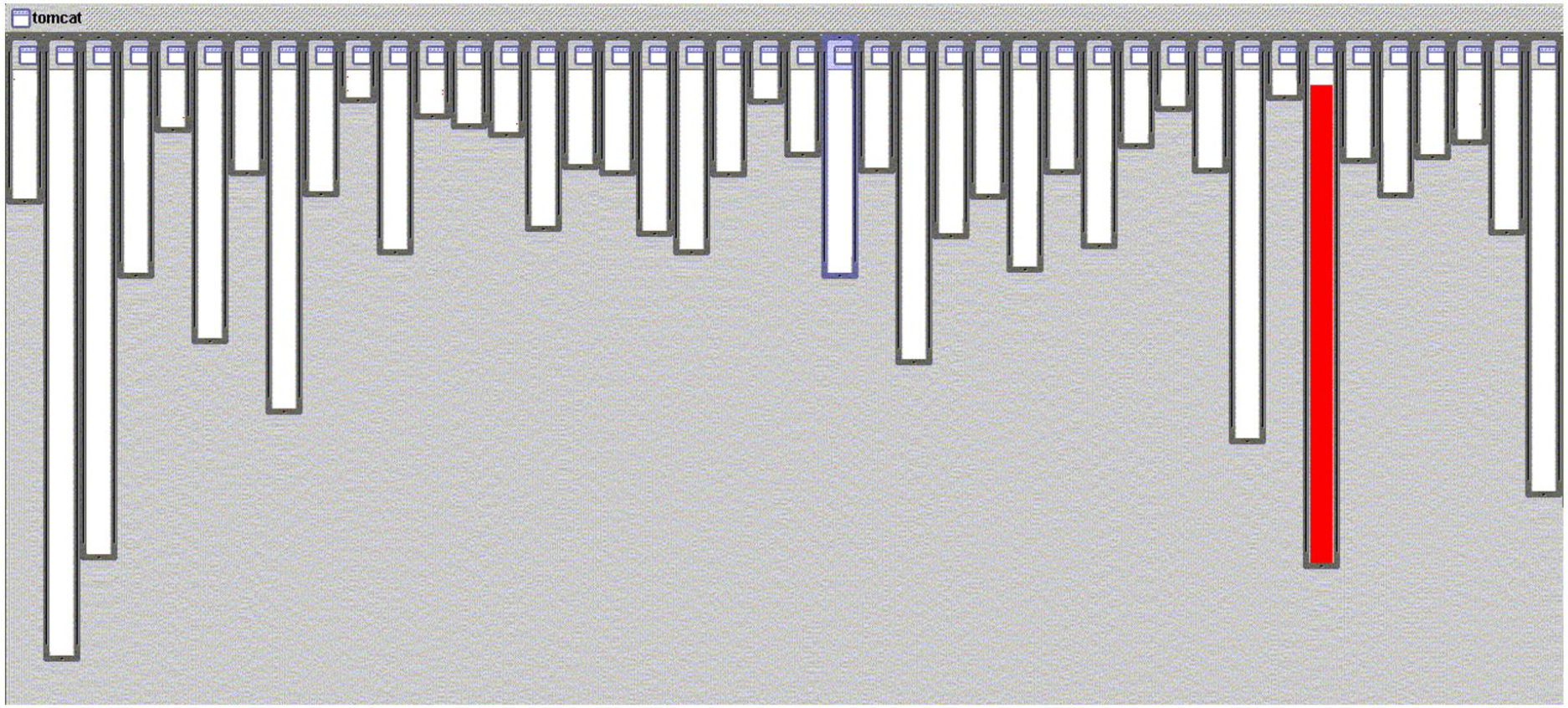
[Java Beans, EJB, .NET, CORBA,...]



Source: http://ssel.vub.ac.be/jasco/lib/exe/fetchb6e2.php?cache=cache&media=documentation%3Ajasco-vub-session1.ppt

Good Modularity

XML Parsing in Apache Tomcat

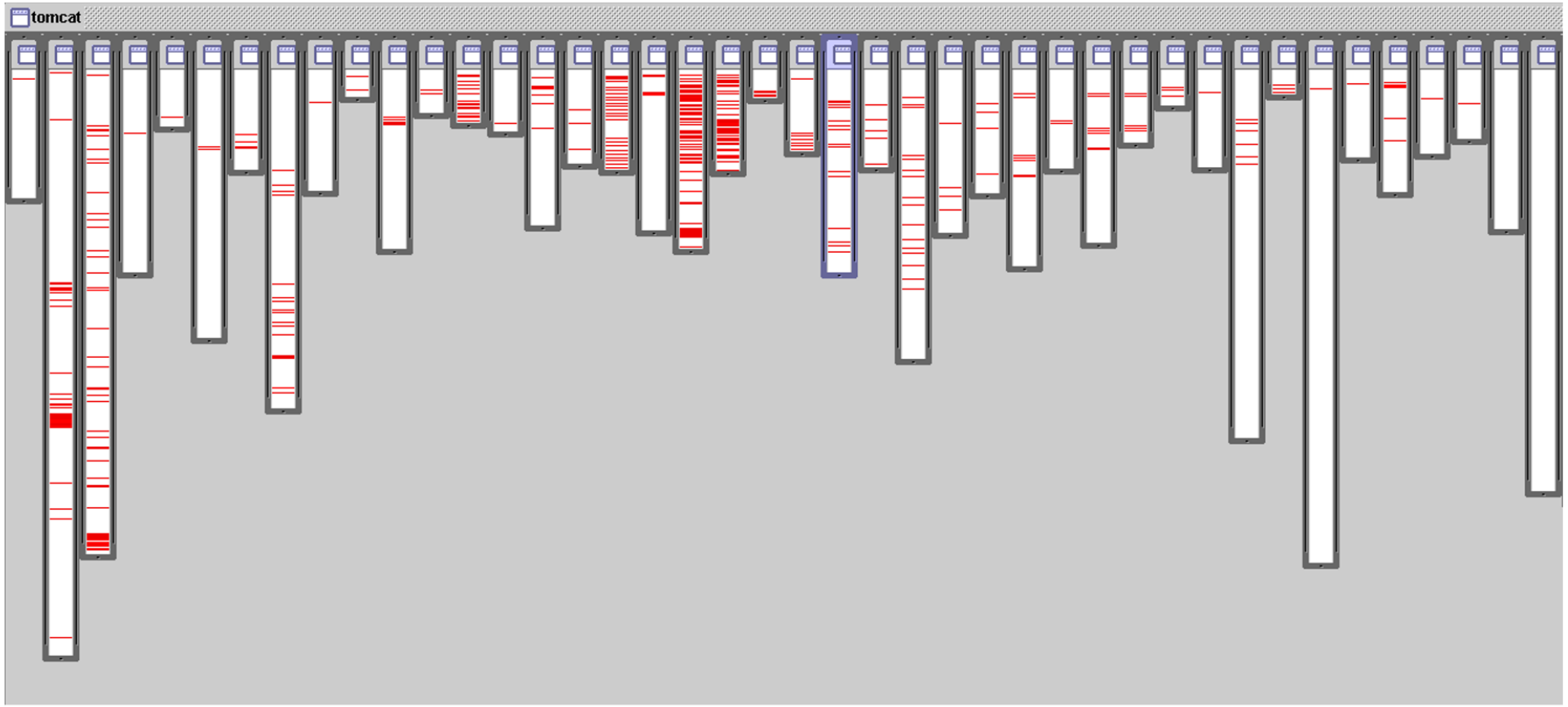


[Picture taken from the aspectj.org website]

Source: http://ssel.vub.ac.be/jasco/lib/exe/fetchb6e2.php?cache=cache&media=documentation%3Ajasco-vub-session1.ppt

Bad modularity

Logging in Apache Tomcat



BAD modularity:

[Picture taken from the aspectj.org website]

Source: http://ssel.vub.ac.be/jasco/lib/exe/fetchb6e2.php?cache=cache&media=documentation%3Ajasco-vub-session1.ppt

How to separate concerns?

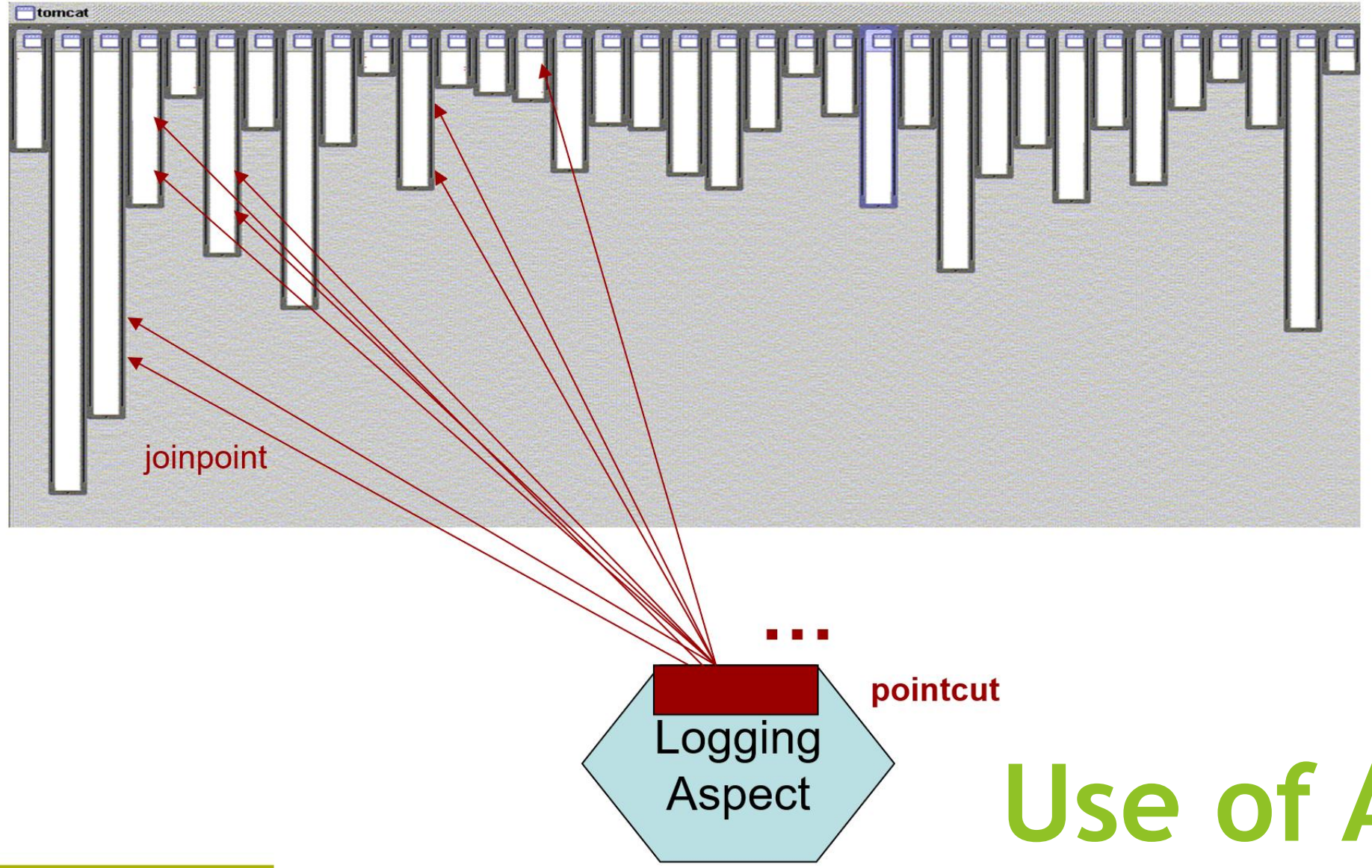
-using design patterns

-using component development, encapsulating functionality and providing it through interfaces

-using architectural patterns

[AspectJ, AspectC++, DJ, ConcernJ,Soul/AOP,…]

Logging in Apache Tomcat



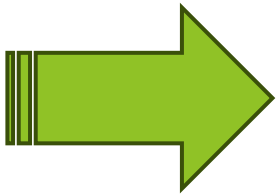
Use of AOP

Source: http://ssel.vub.ac.be/jasco/lib/exe/fetchb6e2.php?cache=cache&media=documentation%3Ajasco-vub-session1.ppt

How to separate crosscutting concerns?

New concern is not tangled with original code

Original code should remain untouched



Open-Closed principle

One of SOLID principles

For extensions

For changes

Injecting content to particular places after development.

At compilation-time

At load-time

At run-time

Aspects-oriented paradigm

Weaving functionality of crosscutting concerns that are preserved in modular way in standalone files called aspects into original code.

UC Place an Order

Basic Flow: Place an Order

Similar analogy with use cases?

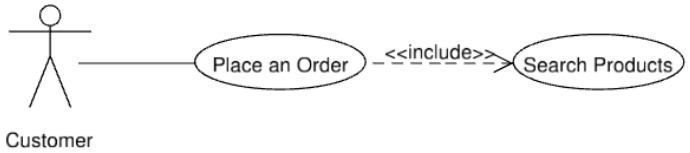
1. Customer selects to place an order.
2. UC *Search Products* is being activated.
3. Customer confirms the product selection and adjusts its quantity.
4. If the product is available, System includes it in the order.
5. Customer continues in ordering further products.
6. Customer chooses the payment method, enters the payment data, and confirms the order.
7. Customer can cancel ordering at any time.
8. The use case ends.

Example taken from: http://www2.fiit.stuba.sk/~vranic/

UC Place an Order

Basic Flow: Place an Order

1. Customer selects to place an order.
2. UC *Search Products* is being activated.
3. Customer confirms the product selection and adjusts its quantity.
4. If the product is available, System includes it in the order.
5. Customer continues in ordering further products.
6. Customer chooses the payment method, enters the payment data, and confirms the order.
7. Customer can cancel ordering at any time.
8. The use case ends.



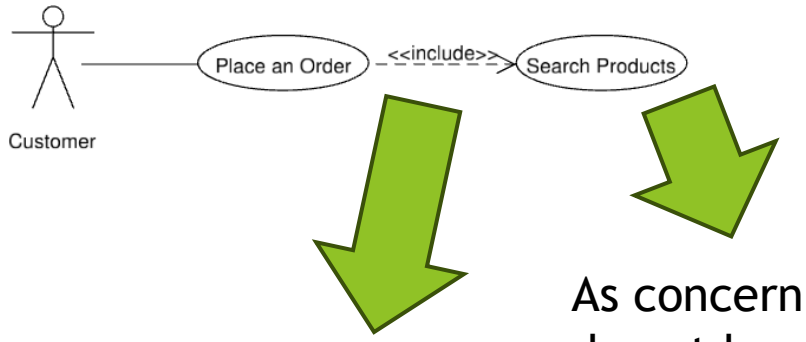

```
public class Ordering {
    ...
    public void order() {
        ...
        new ProductSearch().search(product);
        ...
    }
    ...
}
```

Example taken from: http://www2.fiit.stuba.sk/~vranic/

UC Place an Order

Basic Flow: Place an Order

1. Customer selects to place an order.
2. UC *Search Products* is being activated.
3. Customer confirms the product selection and adjusts its quantity.
4. If the product is available, System includes it in the order.
5. Customer continues in ordering further products.
6. Customer chooses the payment method, enters the payment data, and confirms the order.
7. Customer can cancel ordering at any time.
8. The use case ends.



Some kind of weaving

As concern that do not know about Place An Order concern

```
public class Ordering {
    ...
    public void order() {
        ...
        new ProductSearch().search(product);
        ...
    }
    ...
}
```

Example taken from: http://www2.fiit.stuba.sk/~vranic/

UC Place an Order

Basic Flow: Place an Order

1. Customer selects to place an order.
2. UC *Search Products* is being activated.
3. Customer confirms the product selection and adjusts its quantity.
4. If the product is available, System includes it in the order.
5. Customer continues in ordering further products.
6. Customer chooses the payment method, enters the payment data, and confirms the order.
7. Customer can cancel ordering at any time.
8. The use case ends.

Extension points:

- *Checking Product Availability*: Step 4

Example taken from: http://www2.fiit.stuba.sk/~vranic/

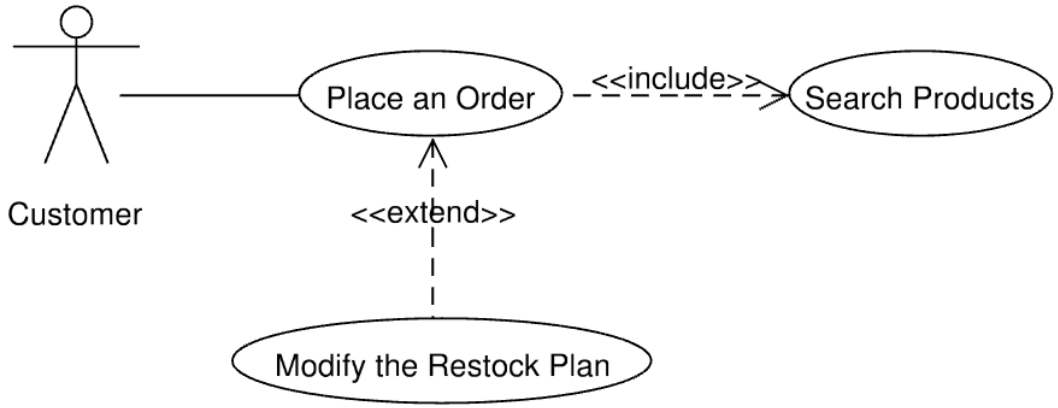
UC Place an Order

Basic Flow: Place an Order

1. Customer selects to place an order.
2. UC *Search Products* is being activated.
3. Customer confirms the product selection and adjusts its quantity.
4. If the product is available, System includes it in the order.
5. Customer continues in ordering further products.
6. Customer chooses the payment method, enters the payment data, and confirms the order.
7. Customer can cancel ordering at any time.
8. The use case ends.

Extension points:

- *Checking Product Availability*: Step 4



UC Modify the Restock Plan

Alternate Flow: Modify the Restock Plan

After the *Checking Product Availability* extension point of the Place an Order use case:

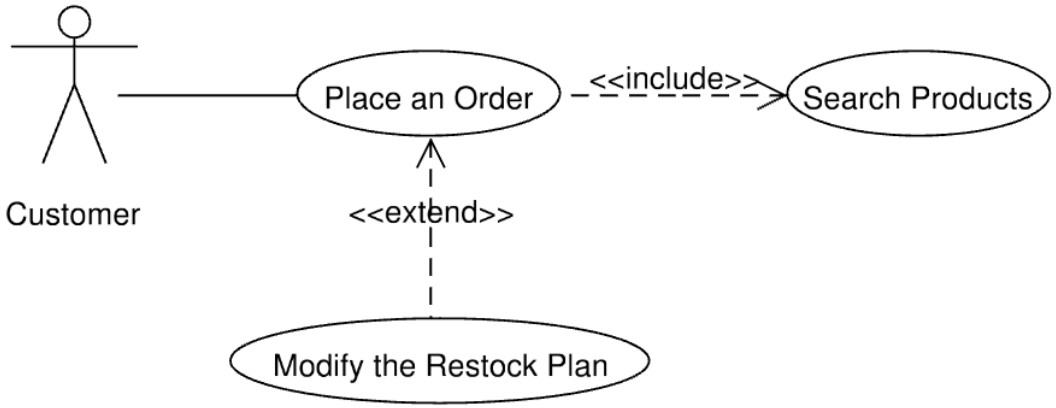
1. System checks the available quantity of the product being ordered.
2. If the quantity is below the limit, System adds the quantity under demand to the restock plan.
3. The flow continues with the step that follows the triggering extension point.

Example taken from: http://www2.fiit.stuba.sk/~vranic/

UC Place an Order

Basic Flow: Place an Order

1. Customer selects to place an order.
2. UC *Search Products* is being activated.
3. Customer confirms the product selection and adjusts its quantity.
4. If the product is available, System includes it in the order.
5. Customer continues in ordering further products.
6. Customer chooses the payment method, enters the payment data, and confirms the order.
7. Customer can cancel ordering at any time.
8. The use case ends.



Extension points:

- *Checking Product Availability*: Step 4

After Step 4:

UC Modify the Restock Plan

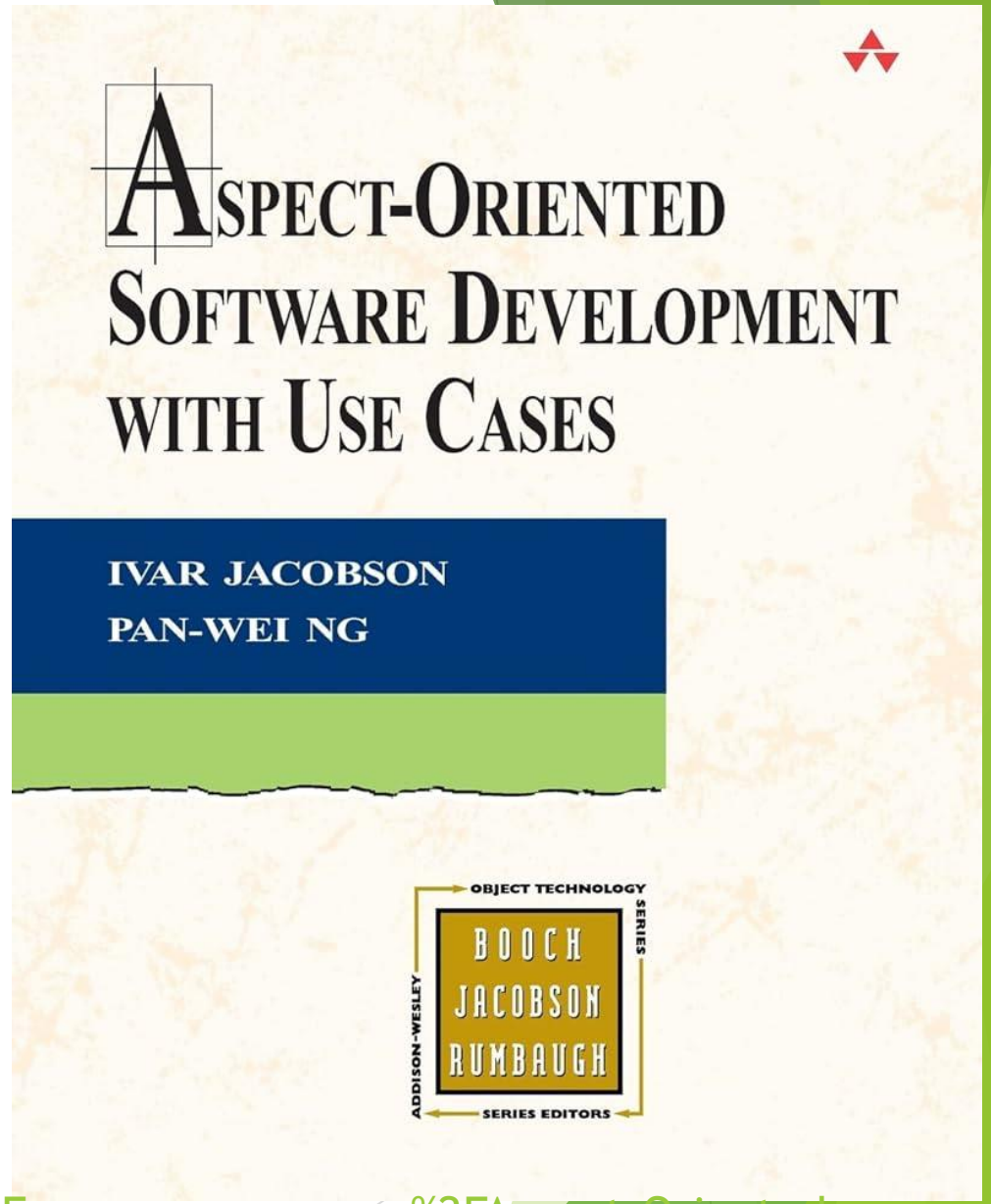
Alternate Flow: Modify the Restock Plan

After the *Checking Product Availability* extension point of the Place an Order use case:

1. System checks the available quantity of the product being ordered.
2. If the quantity is below the limit, System adds the quantity under demand to the restock plan.
3. The flow continues with the step that follows the triggering extension point.

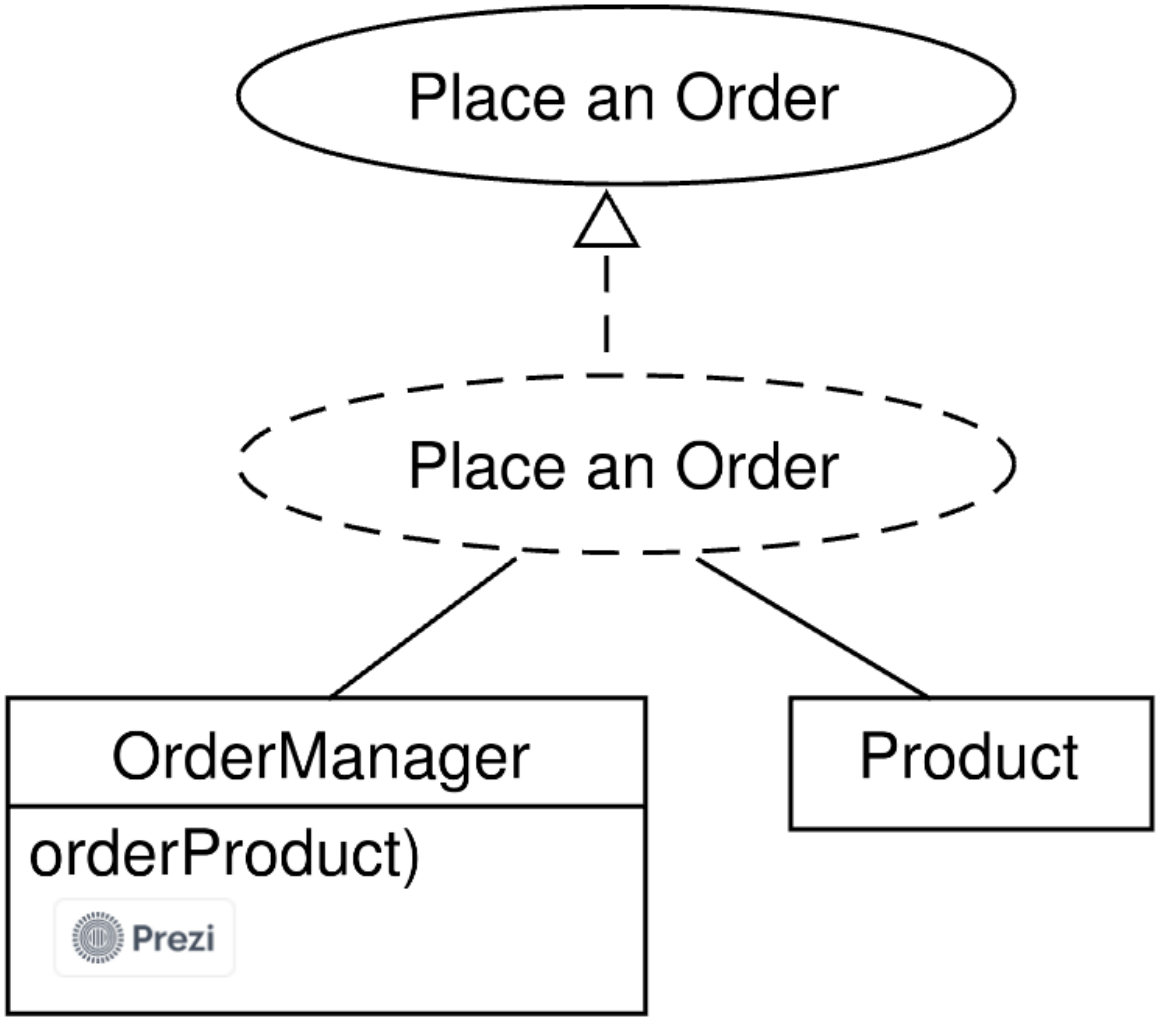
Example taken from: http://www2.fiit.stuba.sk/~vranic/

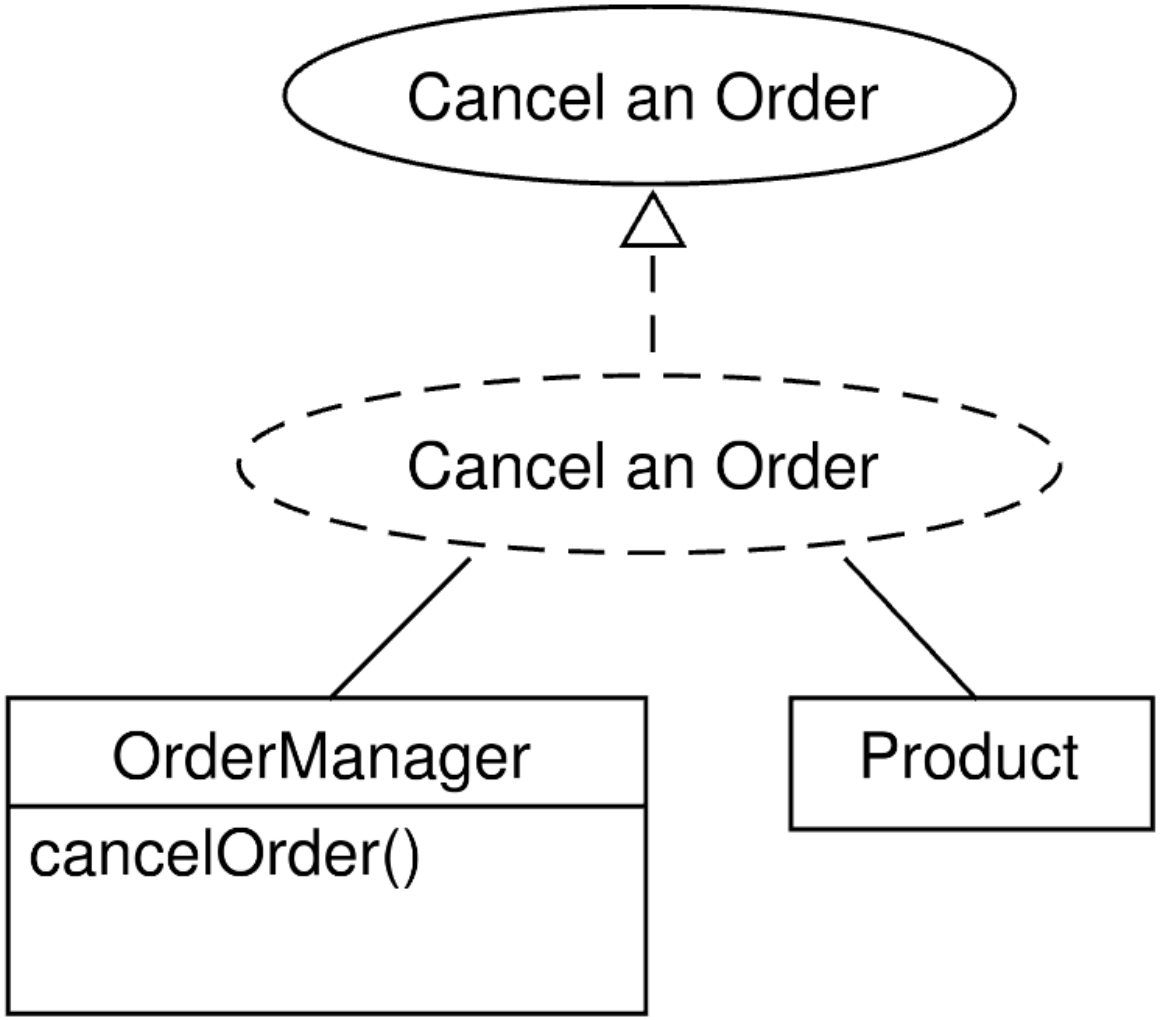
Approach to Use-Case Modelling Using Aspects

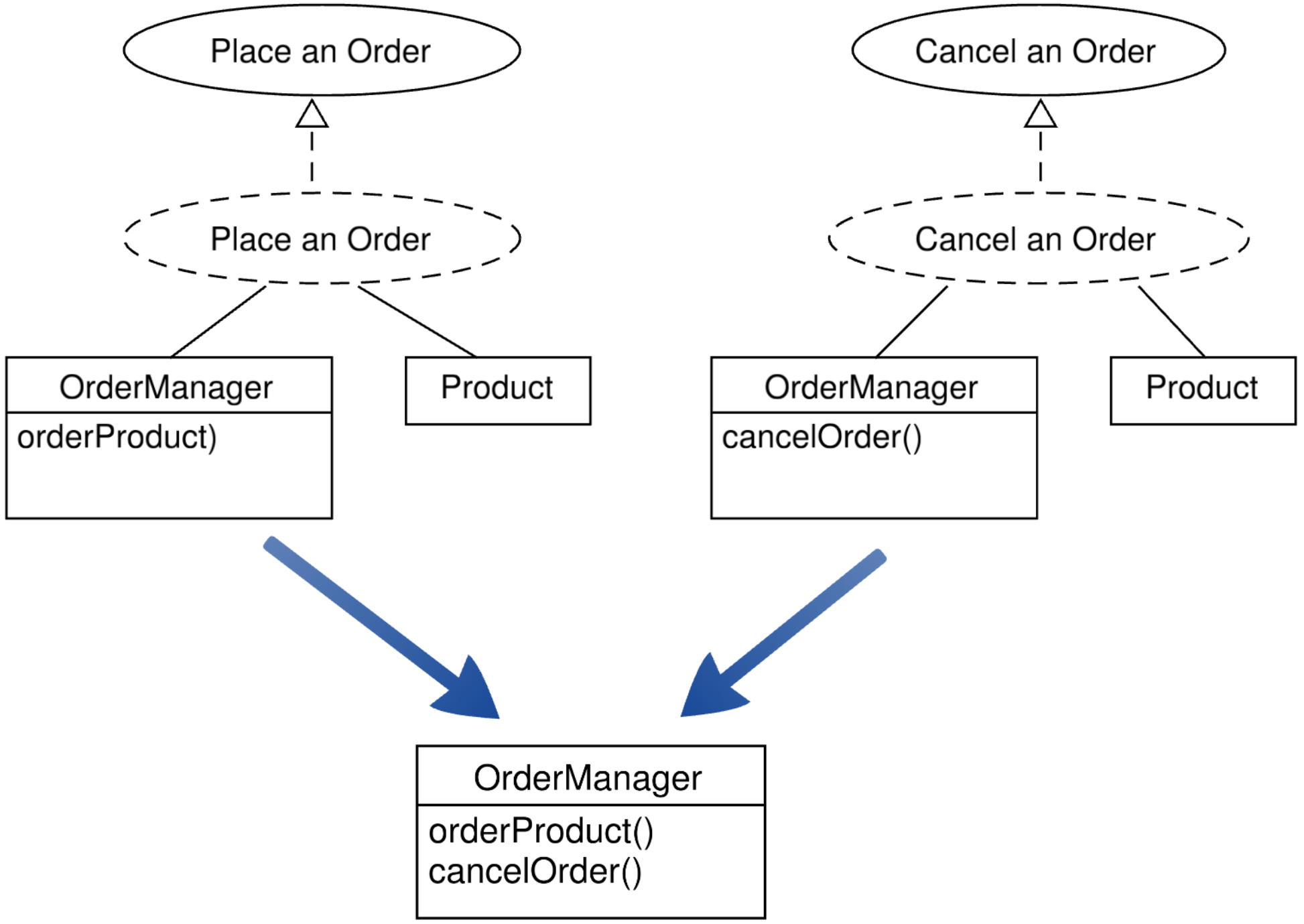


A refreshingly new approach toward improving use-case modeling by fortifying it with aspect orientation." -- Ramnivas Laddad, author of AspectJ in Action "Since the 1980s, use cases have been a way to bring users into software design, but translating use cases into software has been an art, at best, because user goods often don't respect code boundaries. Now that aspect-oriented programming (AOP) can express crosscutting concerns directly in code, the man who developed use cases has proposed step-by-step methods for recognizing crosscutting concerns in use cases and writing the code in separate modules. If these....

Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.amazon.com%2FAspect-Oriented-Software-Development-Use-Cases%2Fdp%2F0321268881&psig=AOvVaw0VBnKKncqUKNR37gDS5eT&ust=1726665756534000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCIjyj6uJyogDFQAAAAAdAAAAABAE


Place an Order
Place an Order
OrderManager
orderProduct)
Product

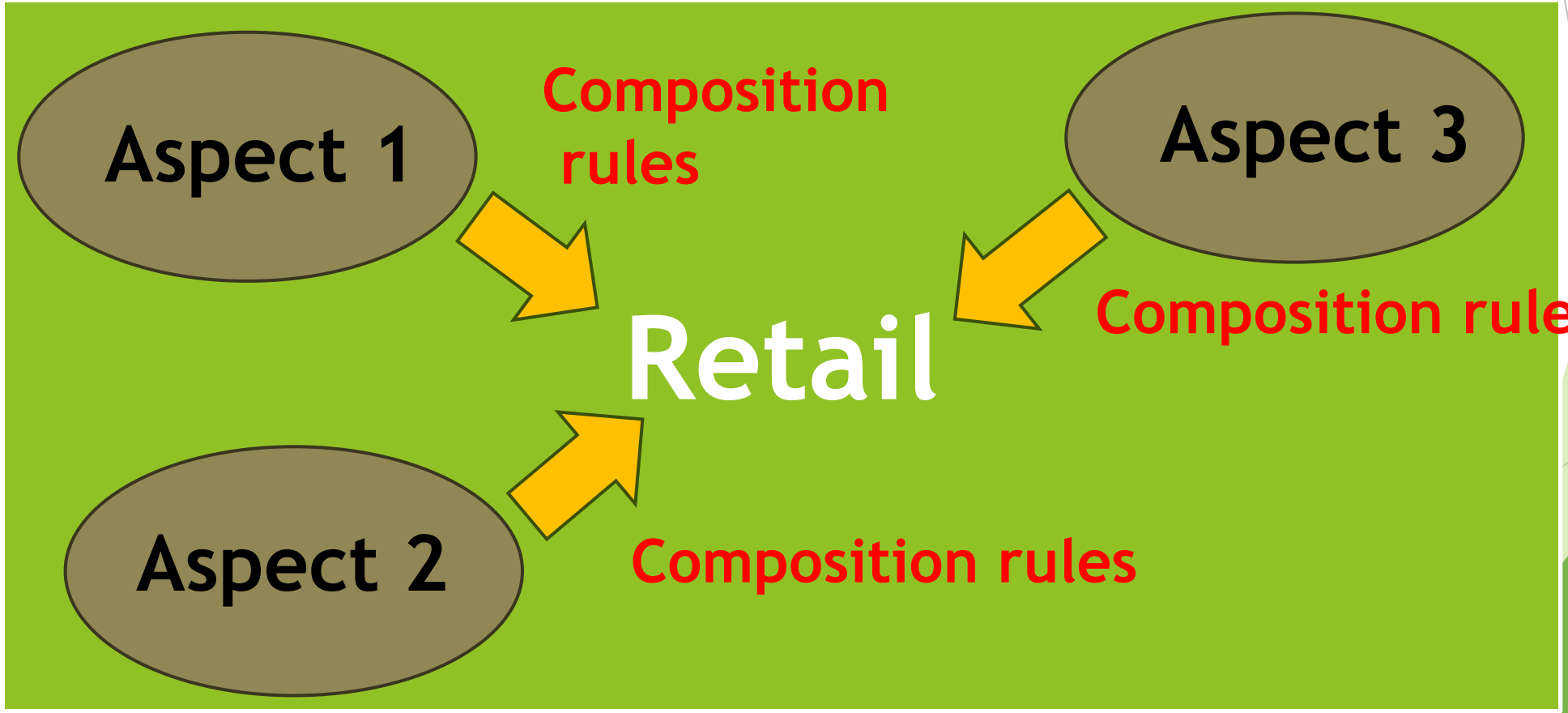

Cancel an Order
Cancel an Order
OrderManager
cancelOrder()
Product


Place an Order
Cancel an Order
Place an Order
Cancel an Order
OrderManager
orderProduct)
Product
OrderManager
cancelOrder()
Product
OrderManager
orderProduct()
cancelOrder()

Symmetric AOP modularization

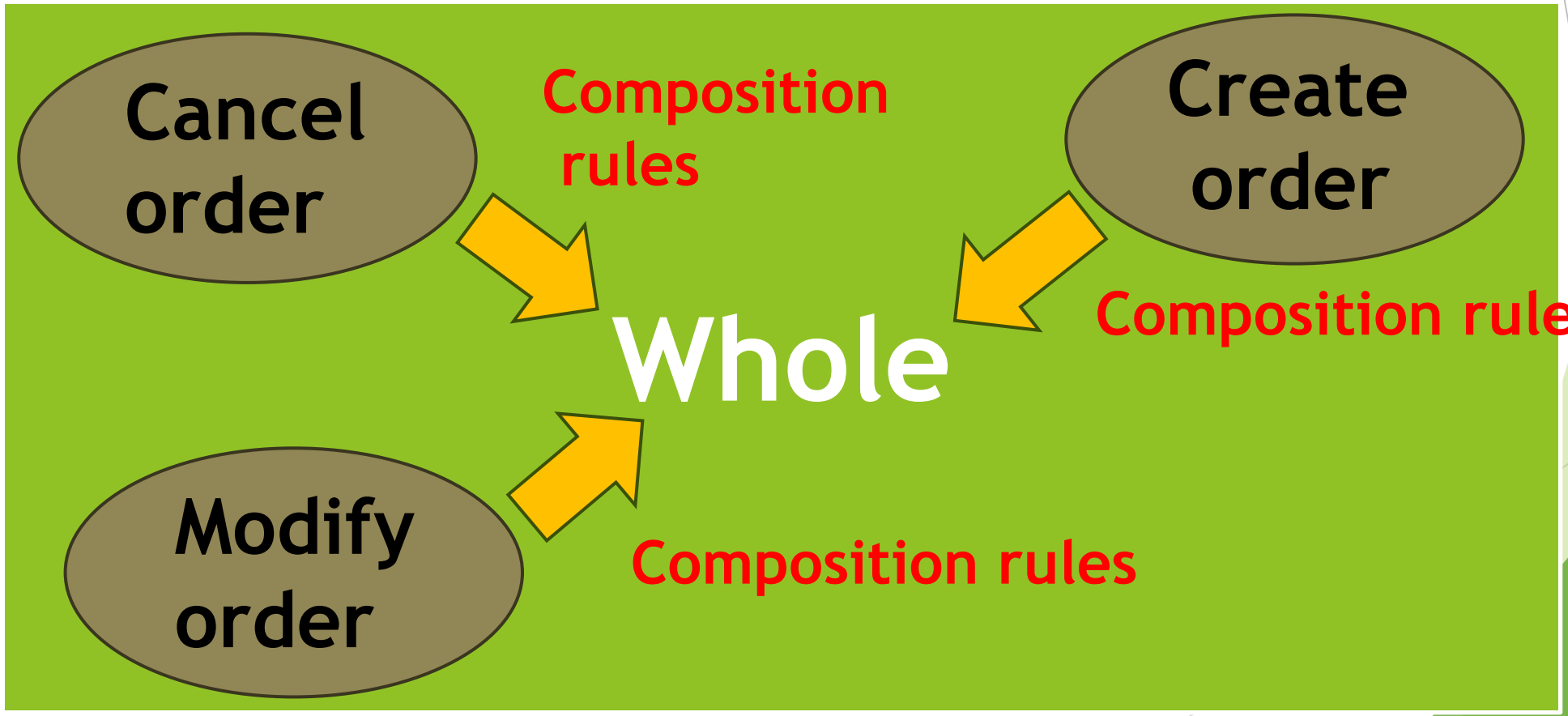
Whole is composed out of the aspects at the same level:

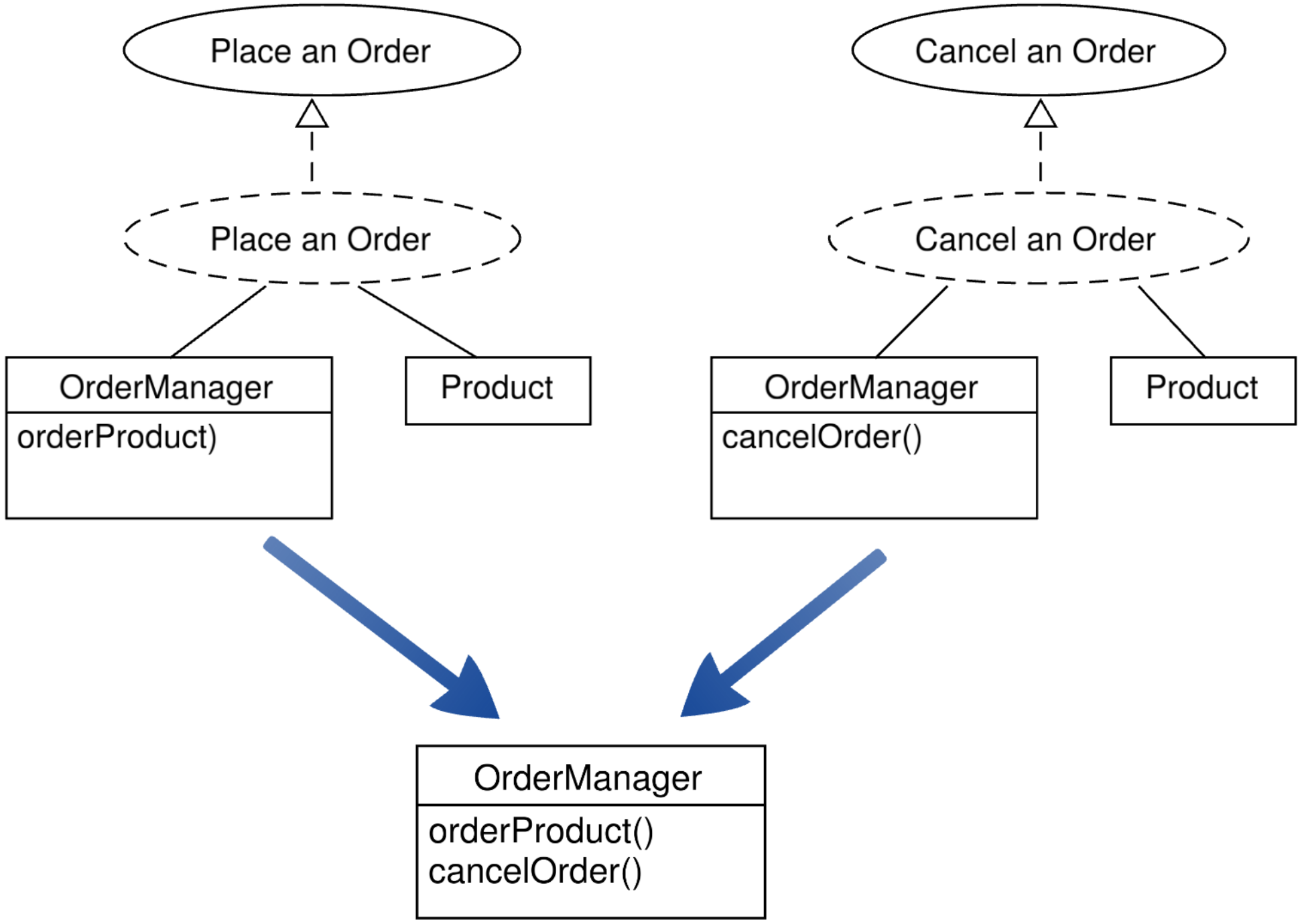
Each aspect is different view of whole



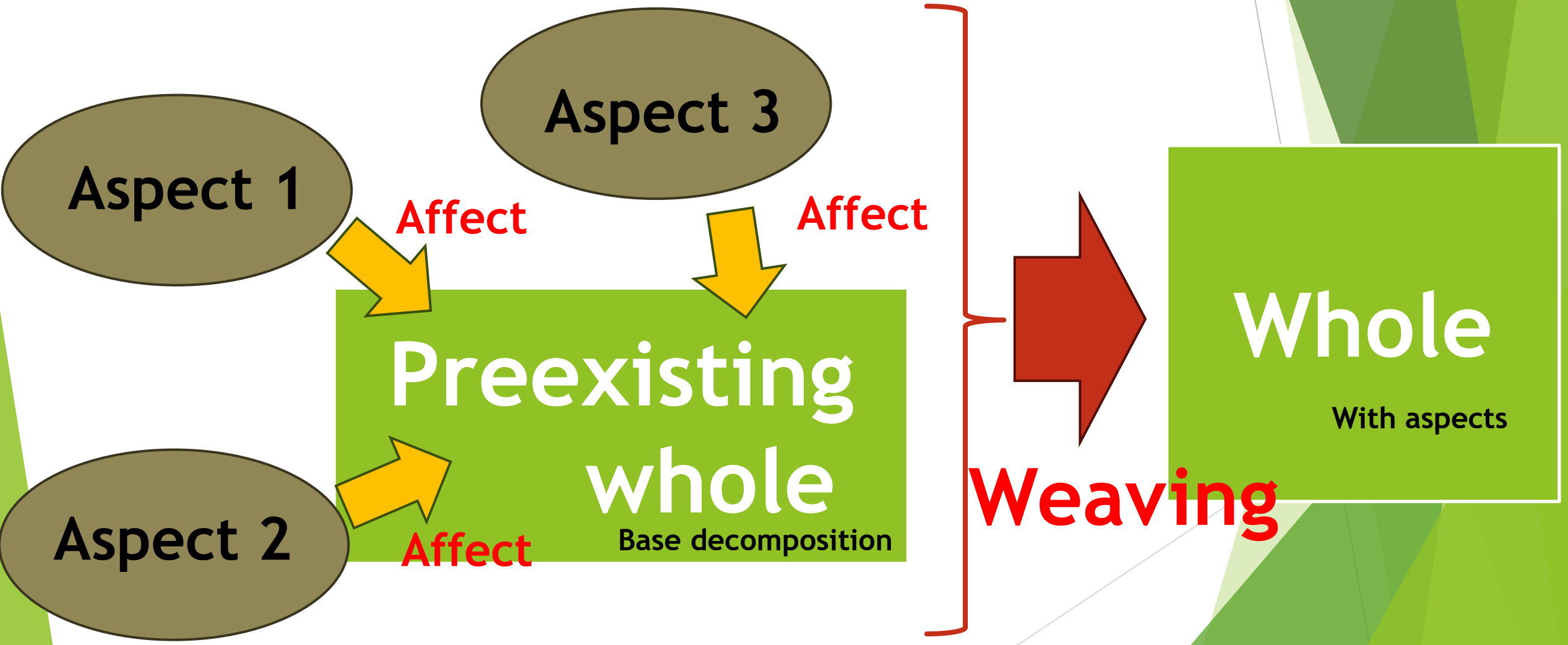
Symmetric AOP modularization

Retail/order is composed out of its concerns (create, cancel, modify,…) at the same level:




Place an Order
Cancel an Order
Place an Order
Cancel an Order
OrderManager
orderProduct)
Product
OrderManager
cancelOrder()
Product
OrderManager
orderProduct()
cancelOrder()

Asymmetric AOP modularization



Asymmetric AOP modularization

Base decomposition  Aspects

AspectJ

Symmetric AOP modularization

Whole is composed out of different views

HyperJ

Peer use cases

Symmetric AOP

Entities at the same level are composed into whole

vs.

Asymmetric AOP

Extend relationship

Basic entity is affected by special entity (aspect)

Dictionary of basic terms

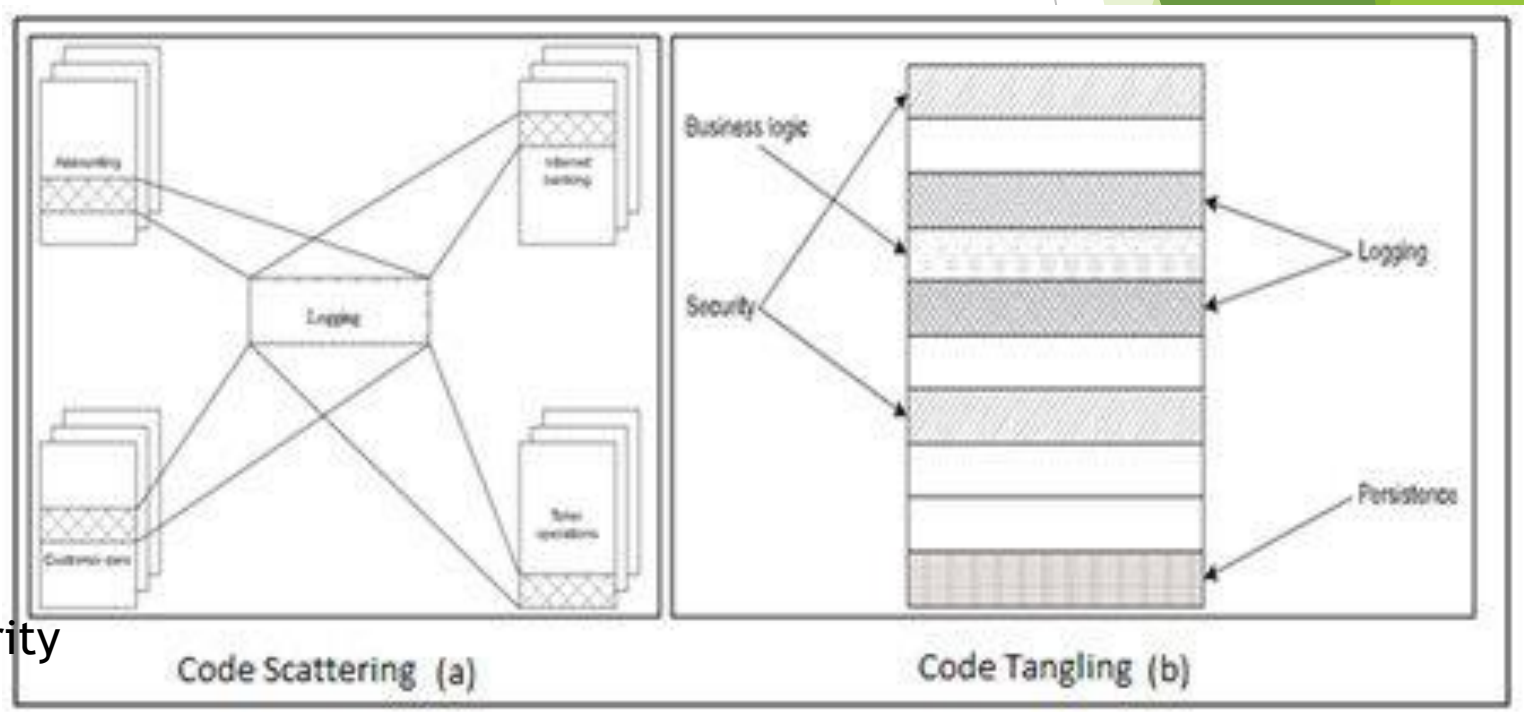
- **Concern**
  - Specific requirement or consideration that must be addressed in order to satisfy overall system goal (Laddad 2003).
- **Software system**
  - Realization of a set of concerns (Laddad 2003).
- **Aspect**
  - New unit of modularization. Aspects crosscuts other modules.
- **Crosscutting concerns**
  - System-wide concerns (such as logging, remote management, and path optimization, etc.) that span multiple modules
- **Aspect weaver**
  - A compiler-like entity which composes the final system – combines core and crosscutting modules.
- **Weaving**
  - The process where core and crosscutting concerns are combined into the final system.
- **Pointcuts**
  - Sets of join points

Dictionary of basic terms

- **Join Point**
  - Any point in the system that can be executed (Laddad 2003). It can be calling a method, creating a class, accessing a variable, but also executing the condition in the code. Different languages only support certain captures types, for example the developed language AspectJ works at the level of methods and fields (Lee and Kang 2004).
  - Determined with pointcuts
- **Variation Point**
  - A point that identifies the places where it occurs to variations (Jacobson et al. 1997). As long as it is modeled on their basis variability, then the main components of assets consist of these points, and also of from them, components of the target system will be created using variants (Webber and Gomaa 2004). All available variants are thus supported.
- **Advice**
  - The advice is the action and decision-making part which it allows for the expression of intersecting action at the joining point (Laddad 2003). Point connection is captured by the corresponding pointcut. It is implemented as construction in the form of a method called before (before), behind (after) or wrapping (around) the specified join point.
  - To express modification of original code
- **Pointcuts**
  - Selects sets of join points and collects context in their place

Dictionary of basic terms

- **Code Tangling**
  - Module handles multiple concerns simultaneously.
- **Code Scattering**
  - Single issue is implemented in multiple modules.
  - ***A) Duplicated code blocks***
    - Repeated almost identical code
  - ***B) Complementary code blocks***
    - Complementary parts of the same concern implemented by various modules



Taken from: Magableh, Aws & Alsobeh, Anas. (2018). Aspect-Oriented Software Security Development Life Cycle (AOSSDLC). 10.5121/csit.2018.81204.

Implementation of logging concern

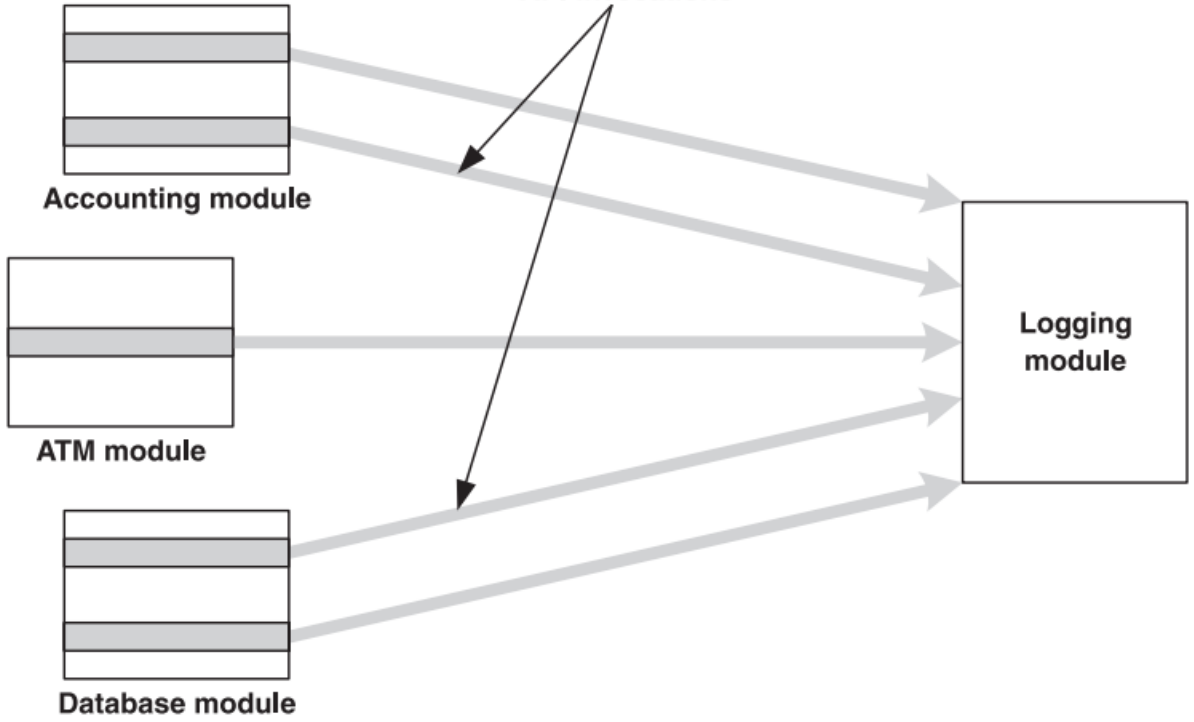


Figure 1.5 Implementation of a logging concern using conventional techniques: The logging module provides the API for logging. However, the client modules—Accounting, ATM, and Database—each still need to embed the code to invoke the logging API.

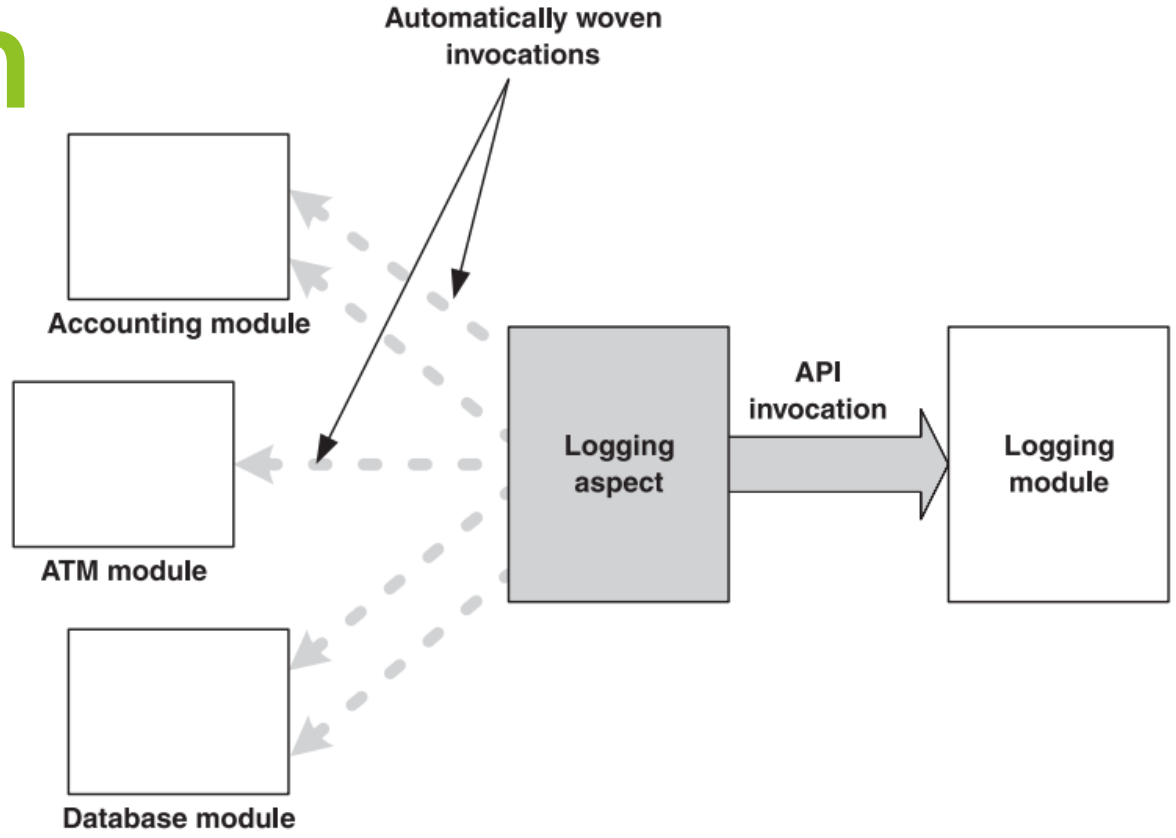


Figure 1.6 Implementation of a logging concern using AOP techniques: The logging aspect defines the interception points needing logging and invokes the logging API upon the execution of those points. The client modules no longer contain any logging-related code.

Taken from: LADDAD, Ramnivas, 2003. AspectJ in action: practical aspect-oriented programming. Greenwich, CT: Manning. ISBN 978-1-930110-93-9

Pointcuts - different kinds

- Annotation based pointcuts
- Method/constructor call
- Method/constructor execution
- Execution object pointcuts
- Control flow based pointcuts
- Argument pointcuts
- Initialization
- Field Access
- Exception Handling
- Advice execution
- Conditional pointcut
- Pointcut based on Lexical structure

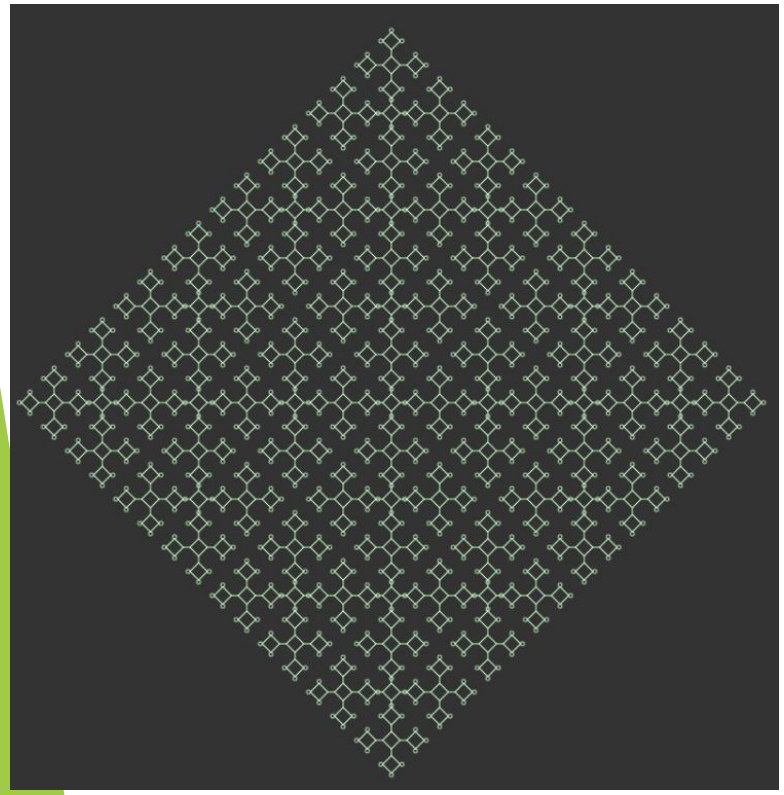
Example: Annotation based join points

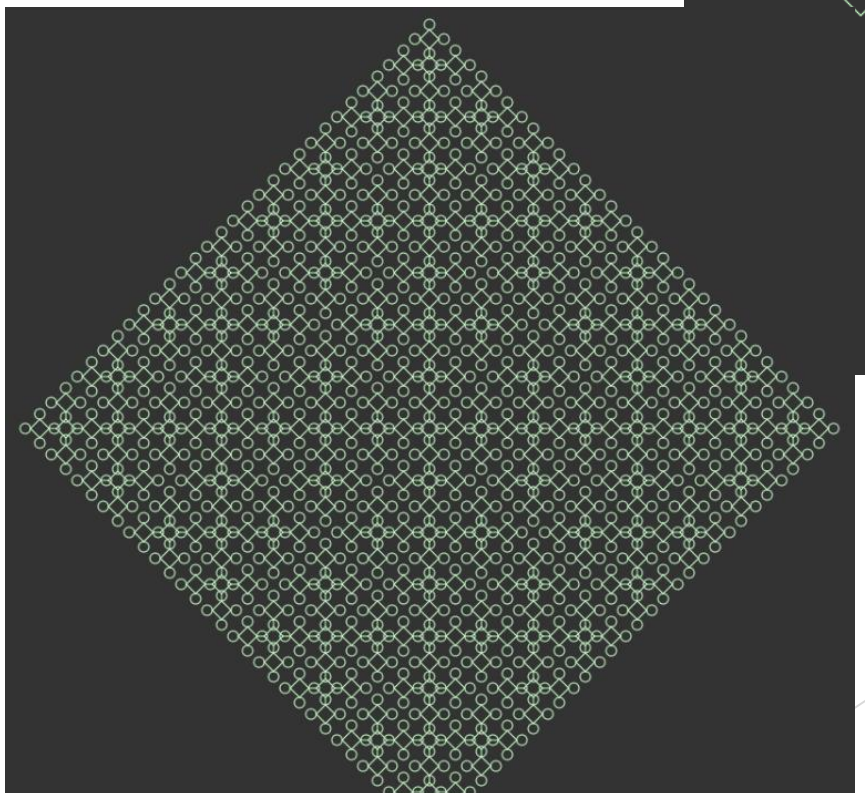
```
1  var newVariable = "Hallo";
2  function a(param1, param2) {
3      @wholeClass({ "outsideGallery": "true" })
4      class GG {
5          callMe() { /* ... callMe function logic ...
                */ }
6      }
7  }
8  /* ... other content ... */
```

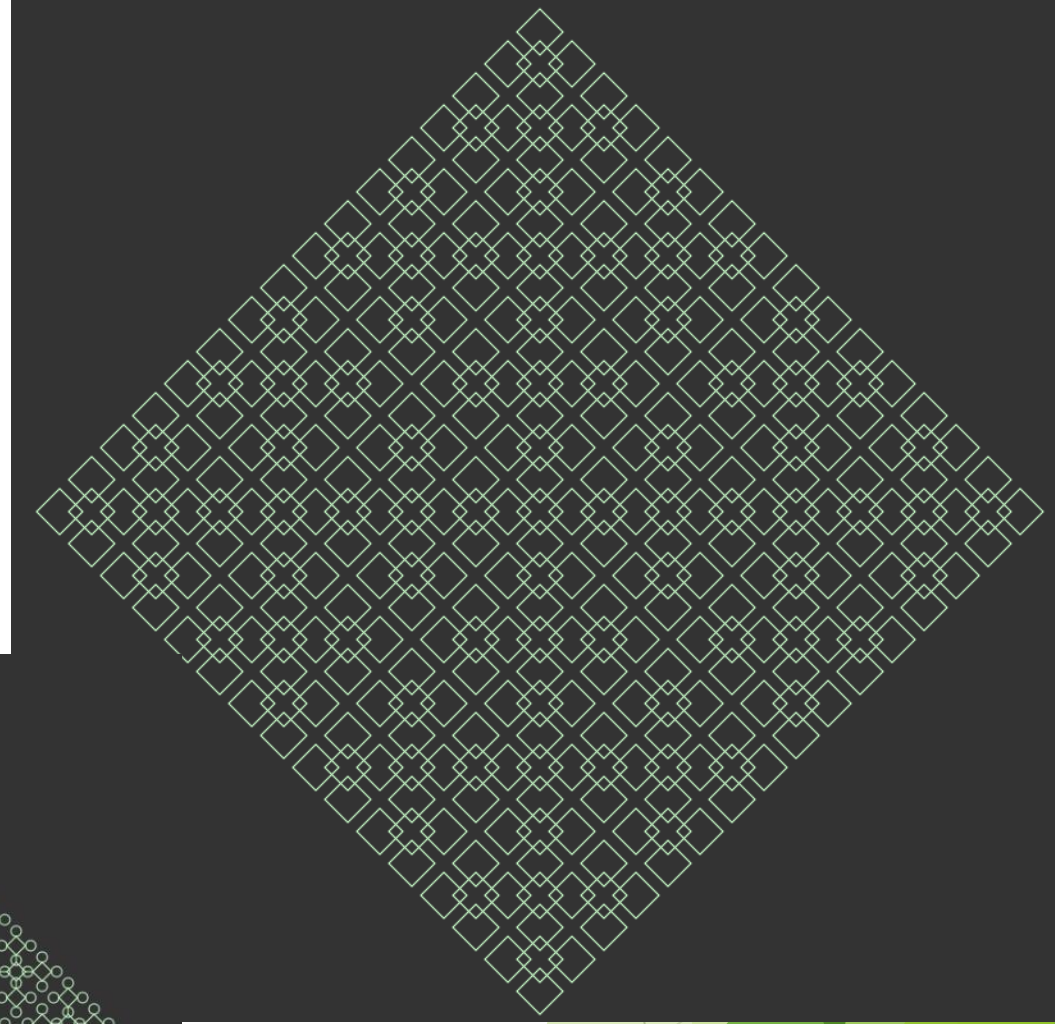
Automated pointcut extraction with semantic information

```
1  var markerVP1 = { "global": {}, "inner": { "p": 699 } }
2  @AnnotationVP1.variableVP() var newVariable = "Hallo";
3  var markerVP2 = { /*...*/ };
4  @AnnotationVP2.functionVP() function a(param1, param2) {
5      var markerVP3 = { /*...*/ };
6      @AnnotationVP3.classVP()
7      class GG {
8          @AnnotationVP10.classVariableVP()
9          markerVP6 = { /*...*/ };
10         @AnnotationVP4.classFunctionVP()
11         callMe() {
12             var markerVP4 = { /*...*/ };
13         }
14         @AnnotationVP11.classVariableVP()
15         markerVP7 = { /*...*/ };
16     }
17 }
18 var markerVP15 = { /*...*/ };
19 /* ... other content ... */
```

Application: Software product line for fractals

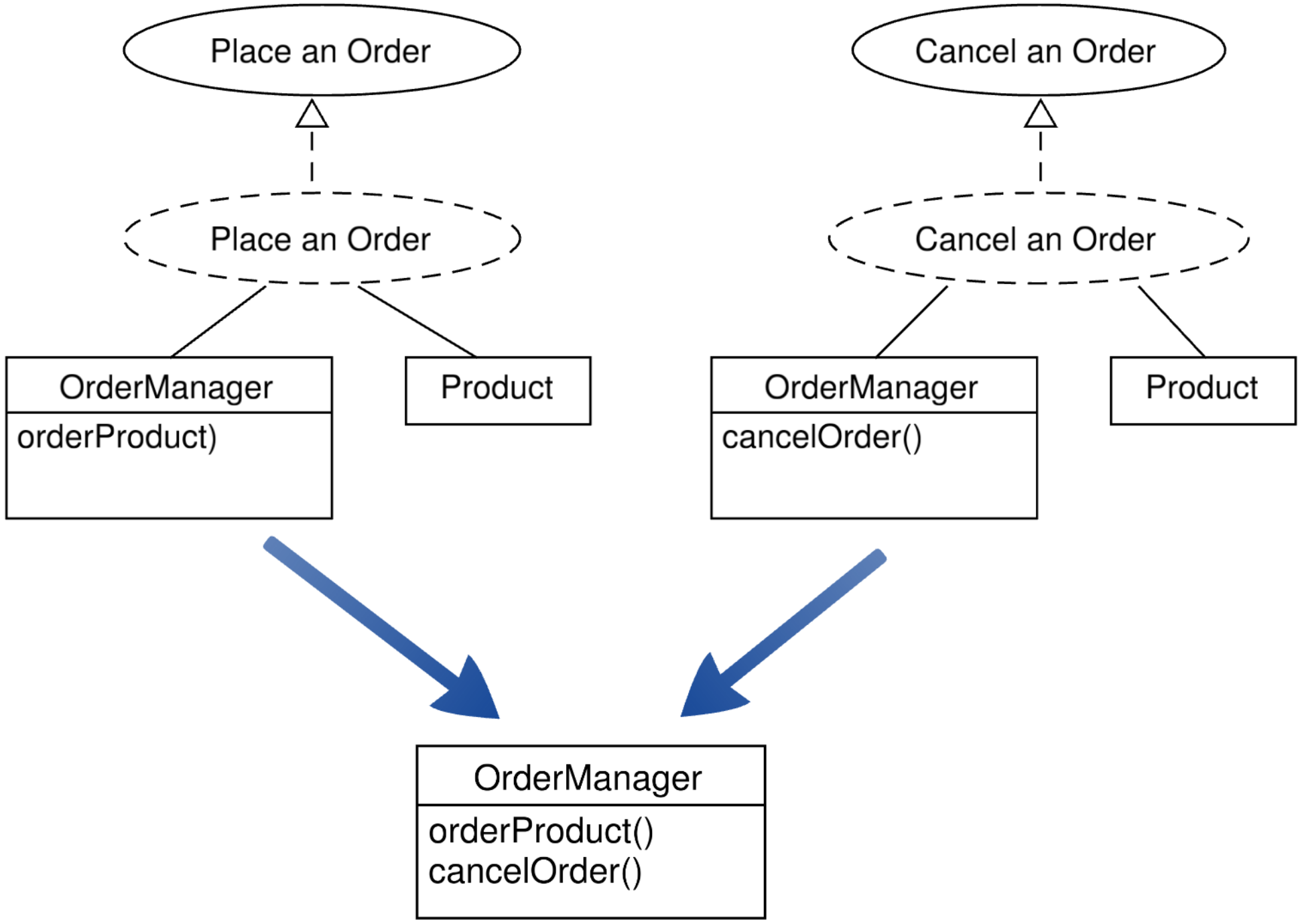






```
var markerVP1 = { "global": {}, "inner": { "p": 0 } };
@AnnotationVP1.variableVP() var newVariable: string = "Hallo";
var markerVP2 = { "global": { "globalVariables": [{ "name": "newVariabl
};
@AnnotationVP2.functionVP() function a(param1: number, param2: number)
    var markerVP3 = { "allAvailableCalls": ["a([%[param1: number]%], [%
                }] }, "inner": { "p": 737, "allAvailableCalls": ["a([%[
    @wholeClass({ "outsideGallery": "true" })
    class GG {
        @AnnotationVP18.classVariableVP()
        markerVP6 = { "allAvailableCalls": ["a([%[param1: number]%], [%
            ], "global": { "globalVariables": [{ "name": "newVariable",
        @wholeClassMethod({ "outsideGallery": "true" })
        callMe() {
            var markerVP4 = { "allAvailableCalls": ["a([%[param1: numbe
                        }] }, "inner": { "p": 351, "allAvailableCalls":
            console.log("Called");
            var markerVP5 = { "allAvailableCalls": ["a([%[param1: numbe
                        }] }, "inner": { "p": 351, "allAvailableCalls":
        }
        @AnnotationVP19.classVariableVP()
        markerVP7 = { "allAvailableCalls": ["a([%[param1: number]%], [%
            ], "global": { "globalVariables": [{ "name": "newVariable",
    }
    var markerVP8 = { "allAvailableCalls": ["a([%[param1: number]%], [%
                }] }, "inner": { "p": 737, "allAvailableCalls": ["a([%[
    new GG().callMe();
    var markerVP9 = { "allAvailableCalls": ["a([%[param1: number]%], [%
                }] }, "inner": { "p": 737, "allAvailableCalls": ["a([%[
    @AnnotationVP25.variableVP() let local = 5;
    var markerVP10 = { "allAvailableCalls": ["a([%[param1: number]%], [
    @AnnotationVP35.variableVP() var globalOne = 6;
    var markerVP11 = { "allAvailableCalls": ["a([%[param1: number]%], [
    @AnnotationVP45.variableVP() let local2 = 5;
    var markerVP12 = { "allAvailableCalls": ["a([%[param1: number]%], [
    @AnnotationVP55.variableVP() var globalOne2 = 6;
    var markerVP13 = { "allAvailableCalls": ["a([%[param1: number]%], [
    return self;
    var markerVP14 = { "allAvailableCalls": ["a([%[param1: number]%], [
}
var markerVP15 = { "allAvailableCalls": ["a([%[param1: number]%], [%[pa
console.log(a(null, null).local);
var markerVP16 = { "allAvailableCalls": ["a([%[param1: number]%], [%[pa
@wholeClass({ "outsideGallery": "true" })
class BB {
    @AnnotationVP144.classVariableVP()
    markerVP31 = { "allAvailableCalls": ["a([%[param1: number]%], [%[pa
    @variableDeclarationClass({ "outsideGallery": "true" })
```

```
var newVariable = "Hallo";

function a(param1, param2) {
    class GG {
        callMe() {
            console.log("Called");
        }
    }
    new GG().callMe();
    let local = 5;
    var globalOne = 6;
    let local2 = 5;
    var globalOne2 = 6;
    return self;
}
console.log(a(null, null).local);

class BB {
    nnnnn = 4;
    constructor(ccc: number, ddd: number) {

    }

    funnnc(eee): string {

    }

    inner(): void {
        let wwwww = 4;

        function cfunc(pr, pr2) {
            console.log(pr + pr2);
            console.log("wwwww");
            console.log(wwwww);

            function sssss() {
                console.log("Done");
            }
        }
        sssss();
        cfunc(wwwww, this.nnnnn);
    }

    rrrr = 4;

}

new BB().inner();
```


Place an Order
Cancel an Order
Place an Order
Cancel an Order
OrderManager
orderProduct)
Product
OrderManager
cancelOrder()
Product
OrderManager
orderProduct()
cancelOrder()

Dynamically extending class

Prototype-based programming

1. Declaring class

```
class VerticePair {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    getX() { return this.x; }
    getY() { return this.y; }
}
```

2. Instantiating class

```
var verticePair = new VerticePair(5, 6);
var newX = verticePair.x;
console.log(newX); //newX //(or) //to prints newX
```

3. Extending class dynamically

-possibly with new features… that can be then evolved independently

```
VerticePair.prototype.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
```

4. Using the extension

```
verticePair.checkPoint();
```

Test

```
> class VerticePair {

      constructor(x, y) {
          this.x = x;
          this.y = y;
      }

      getX() { return this.x; }
      getY() { return this.y; }
  }

  var verticePair = new VerticePair(5, 6);
  var newX = verticePair.x;
  console.log(newX); //newX //to prints newX


  VerticePair.prototype.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
  verticePair.checkPoint();
```

```
▶ Uncaught Error: Coordinate X is greater!                    VM266:17
      at VerticePair.checkPoint (<anonymous>:17:73)
      at <anonymous>:18:13
```

Dynamically extending object

Dynamic Object Modification

1. Declaring class

```
class VerticePair {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    getX() { return this.x; }
    getY() { return this.y; }
}
```

2. Instantiating class

```
var verticePair = new VerticePair(5, 6);
var newX = verticePair.x;
console.log(newX); //newX //(or) //to print newX

var verticePair0 = new VerticePair(1, 6);
```

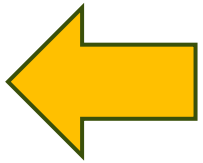
3. Extending class dynamically

-possibly with new features… that can be then evolved independently

```
verticePair.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
```

4. Using the extension

```
verticePair0.checkPoint();
verticePair.checkPoint();
```



Is it successfull? Is extension bound to particular object?

Test

```
> class VerticePair {

      constructor(x, y) {
          this.x = x;
          this.y = y;
      }

      getX() { return this.x; }
      getY() { return this.y; }
  }

  var verticePair = new VerticePair(5, 6);
  var newX = verticePair.x;
  console.log(newX); //newX //to prints newX

  var verticePair0 = new VerticePair(1, 6);
  verticePair.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }

  verticePair.checkPoint();
```

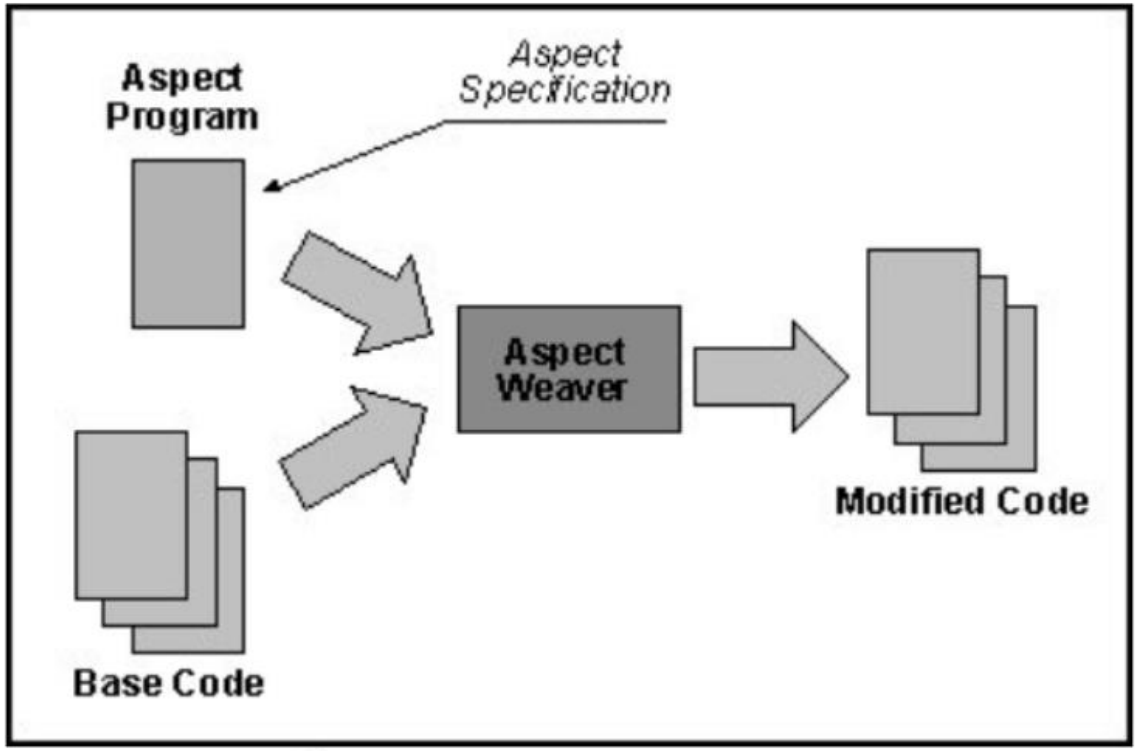
```
▶ Uncaught Error: Coordinate X is greater!                    VM311:17
    at verticePair.checkPoint (<anonymous>:17:63)
    at <anonymous>:19:13
```

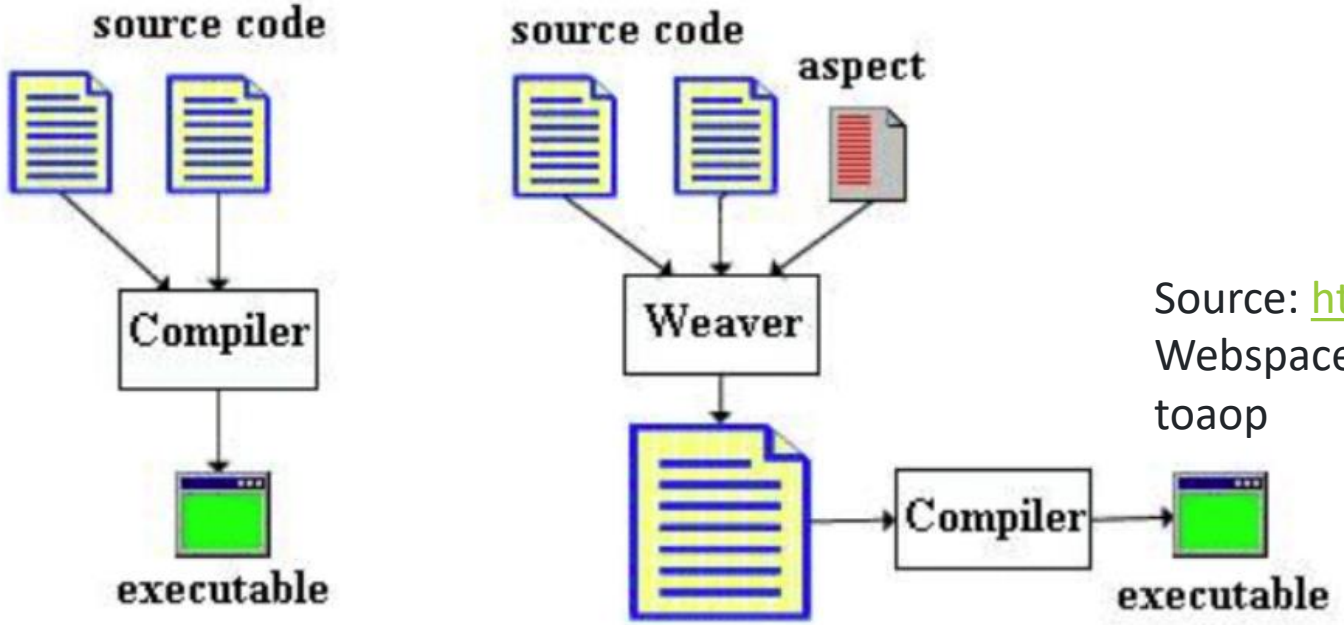
```
> class VerticePair {

      constructor(x, y) {
          this.x = x;
          this.y = y;
      }

      getX() { return this.x; }
      getY() { return this.y; }
  }

  var verticePair = new VerticePair(5, 6);
  var newX = verticePair.x;
  console.log(newX); //newX //to prints newX

  var verticePair0 = new VerticePair(1, 6);
  verticePair.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
  verticePair0.checkPoint();
  verticePair.checkPoint();
```

```
▶ Uncaught TypeError: verticePair0.checkPoint is not a function    VM300:18
    at <anonymous>:18:14
```

AspectJ Basics

Weaving





Source: https://sites.google.com/site/sites/system/errors/WebspaceNotFound?path=%2Fjavatouch%2Fintroductiontoaop

Advice

- The advice is the action and decision-making part which it allows for the expression of intersecting action at the joining point (Laddad 2003). Point connection is captured by the corresponding pointcut. It is implemented as construction in the form of a method called before (before), behind (after) or wrapping (around) the specified join point.
- To express modification of original code

```
Player around(): myPointcut {
    Scanner reader = InputReader.getReader();
    System.out.println("Set player name:");
    String playerNameLine = reader.nextLine().replace("\n", "");

    Player createdPlayer = proceed();
    createdPlayer.setName(playerNameLine);

    System.out.println(createdPlayer.getName());

    return createdPlayer;
}
```

Pointcut

-selects sets of join points and collects context in their place

Example

To capture constructor call, when Player instance is going to be created

```
call(Player.new(..))
```

To check condition (for example for variability handling)

To capture constructor call, when Player instance is going to be created, but setNames from Configuration has to be true

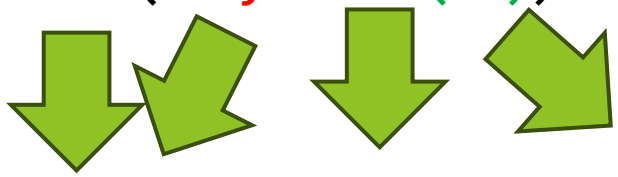
- `call(Player.new(..)) && if(Configuration.setNames);`

Complex examples with whole declaration (within aspect):

```
pointcut manageDifficultyDuringInstantiationOfPlayerOpponent(
        String opponentID, int[] playerShips, BoardManager boardManager):
    call(AbstractPlayer Battleship.instantiateOpponent(String, int[], BoardManager))
            && args(opponentID, playerShips, boardManager)  && !within(DifficultyManagement);

pointcut manageDifficultyDuringInstantiationOfPlayerOpponent2(
        Battleship battleship, String opponentID, BoardManager boardManager):
    call(AbstractPlayer Battleship.instantiateOpponent(String, BoardManager))
            && args(opponentID, boardManager) && this(battleship);
```

Pointcut language

```
call(Player.new(..))
```

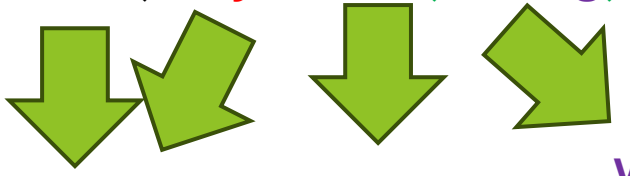


Player class is called

To capture constructor

Any number, name, and type of arguments

```
execution(Player.new(String))
```



Player class is executed

To capture constructor

With one argument from which the first one has type String

Introduction

-a static crosscutting instruction
-to introduce relation such as specialization

Example [USED FOR EXAMPLE JUST TO MARK SOME CLASSES FOR FURTHER PROCESSING]:

declare parents: Human **implements** taxDodger;

Compile-time declaration

-a static crosscutting instruction
-to introduce compile time warnings and errors to warn or prevent some unwanted development practices

Example [TO PREVENT AND CHECK UNWANTED DEVELOPING PRACTICES]:

-a compiler warns when any of System.print or System.println method is used

```
declare warning: call(void System.out.print*(..)) :
"Do not output directly to the console. Use logger instead!";
```

Example: Intertype declaration in AspectJ

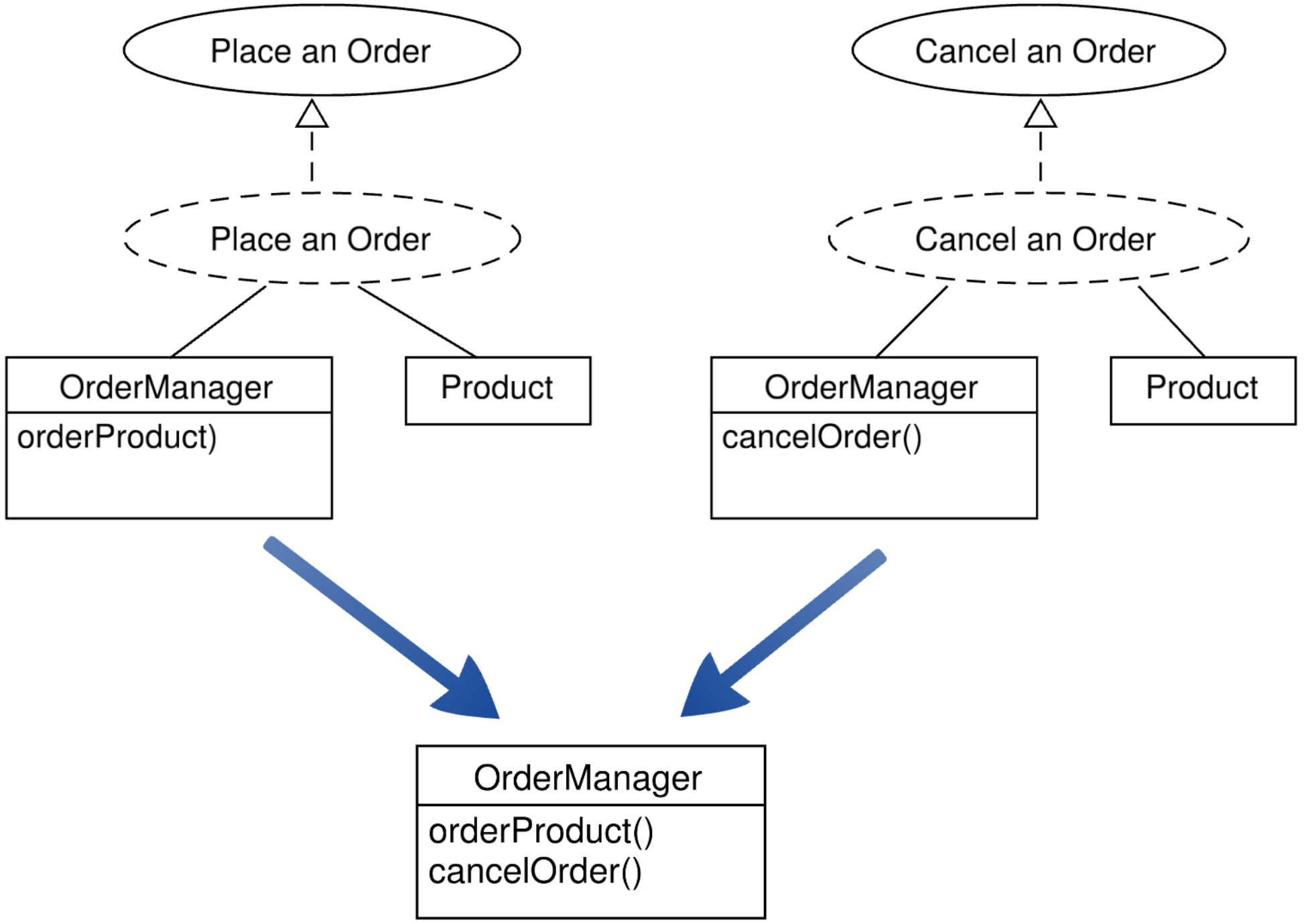
```
public aspect PlayerName {

    private String Player.name;

    private void Player.setName(String playerName) {
        this.name = playerName;
    }

    private String Player.getName() {
        return this.name;
    }
}
```

Extends Player class with ***variable*** called name with String type

Extends Player class with **function** setName(String playerName)

Extends Player class with **function** getName()

Possibly to set up functionality additively and incrementally, only empty classes that will be extended are required.


Place an Order
Cancel an Order
Place an Order
Cancel an Order
OrderManager
orderProduct)
Product
OrderManager
cancelOrder()
Product
OrderManager
orderProduct()
cancelOrder()

Example: Aspect in AspectJ

```
public aspect PlayerName {

    Player around(): call(Player.new(..)) && if(Configuration.setNames){
        Scanner reader = InputReader.getReader();
        System.out.println("Set player name:");
        String playerNameLine = reader.nextLine().replace("\n", "");

        Player createdPlayer = proceed();
        createdPlayer.setName(playerNameLine);

        System.out.println(createdPlayer.getName());

        return createdPlayer;
    }

}
```

How to run AspectJ?

1. Download Eclipse 2024 - 06: -> https://www.eclipse.org/downloads/packages/release/2024-06/r -> Eclipse IDE for Java Developers 328 MB
2. If it is ZIP unpack it to directory: C:\Users\{profile}\eclipse\java-2024-06, otherwise follow instalation steps
3. Launch eclipse.exe positioned in above directory
4. In menu Help -> Eclipse marketplace ->AJDT -> install
5. In menu File -> new Project -> Other -> AspectJ -> AspectJ Project

Compatibility information: https://marketplace.eclipse.org/content/aspectj-development-tools

The Complexity of Asymmetric Aspect Oriented Programming

In its pointcut language.

Aspect-Oriented Programming is Quantification And Obliviousness.

R. E. Filman and D. P. Friedman, 2000

Use of Aspects in Software Product Lines

Aspect-Oriented Product Line

- *An Incremental Aspect-Oriented Product Line Method for J2ME Game Development*

- exploring the **platform variation arising** due to use of **proprietary API and limited memory**. In particular, we consider three platforms (***PA***, ***PB***, ***and PC***) on which the same game GM is run.
  - PA relies solely on MIDP 1.0,
  - whereas PB and PC rely on MIDP 1.0 and proprietary API.
  - Further, PC constrains bytecode size to half of the other platforms.

Figure 1: Platform variation of the GM game

Source: Alves, V., Jr, P.M., Borba, P.: An Incremental Aspect-Oriented ProductLine Method for J2ME Game Development p. 3 (Jan 2004)

Approach

- The goal is to structure a product line around GM so that it can be easily configured for any of the platforms PA, PB, or PC . The outline of our approach is as follows. First, given GM in PA and PB, we identify variation points, refactor code to encapsulate these points, and extract the specialized behavior into AspectJ aspects. The outcome is an aspect for introducing the specifics of each of these two platforms and an abstract GM, which we refer to as GMAbs. Next, the resulting product line is considered with the remaining product, and we reapply the previously described procedure. The result is a new product line encompassing all three products.

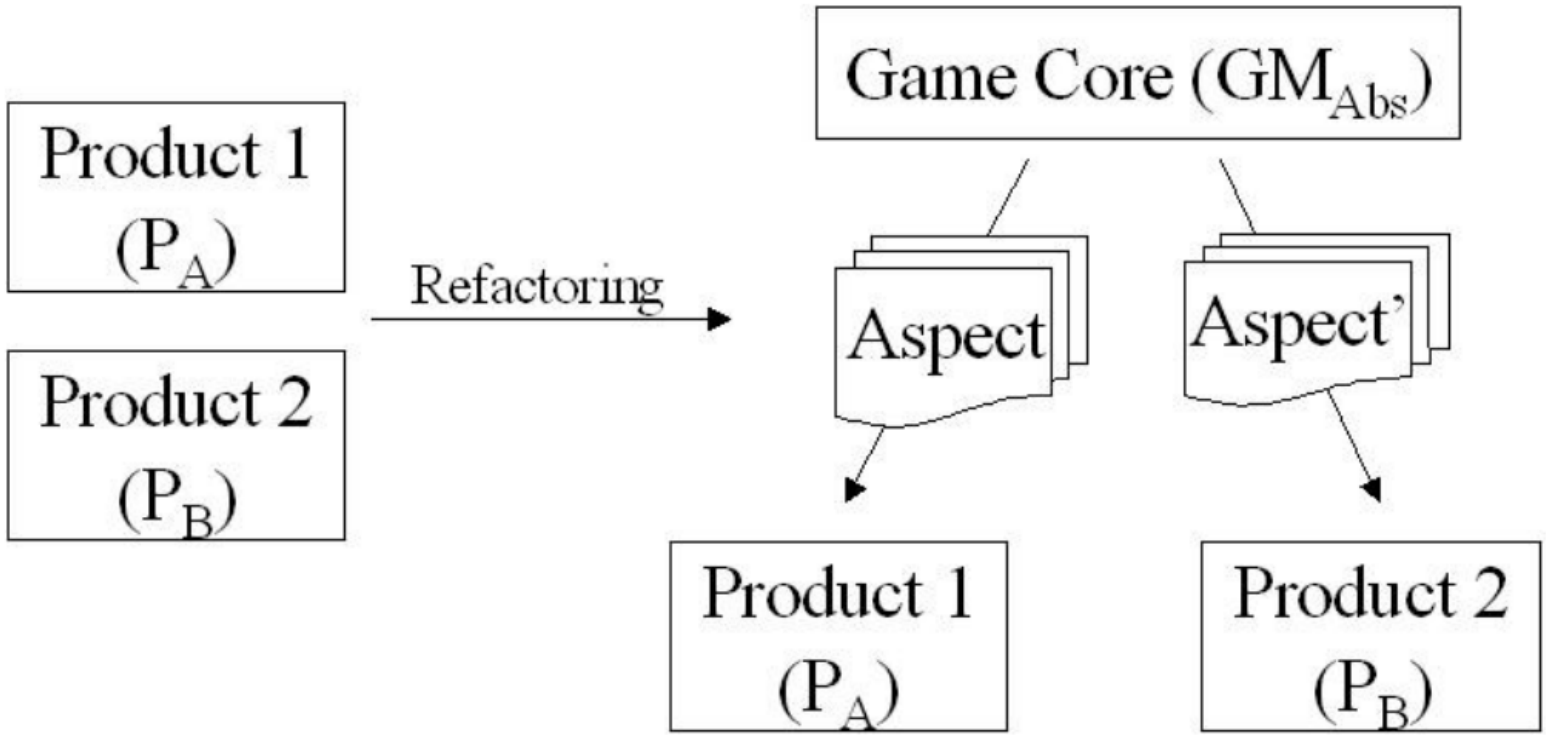


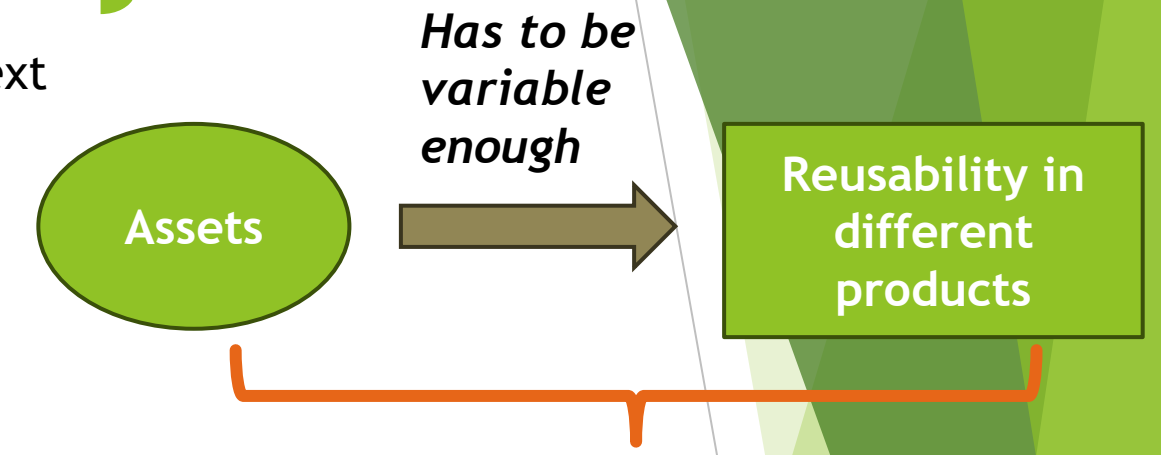
Figure 2: Approach outline

Source: Alves, V., Jr, P.M., Borba, P.: An Incremental Aspect-Oriented ProductLine Method for J2ME Game Development p. 3 (Jan 2004)

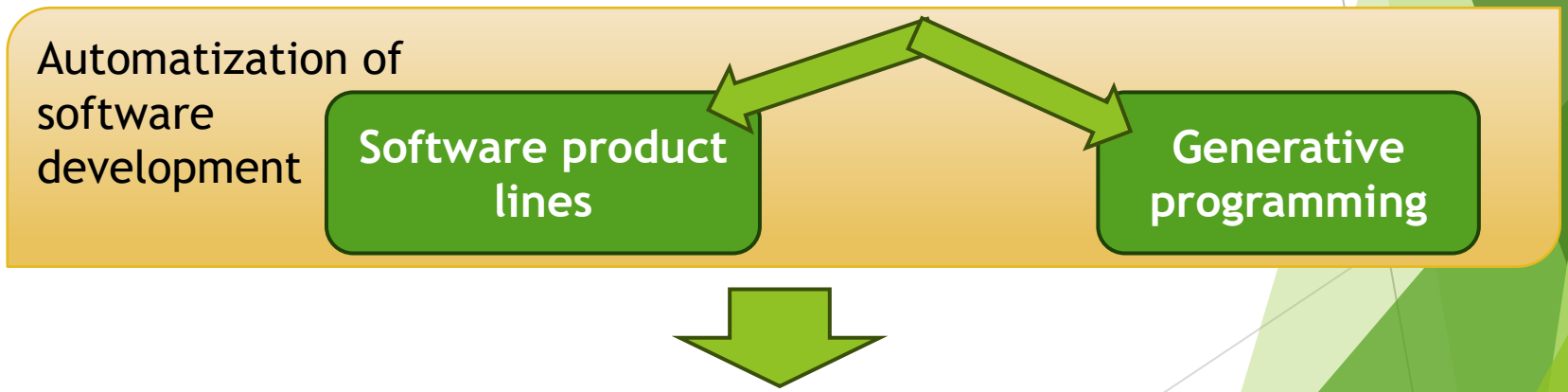
Improving Reusability

Using the same functionality in different context

- **Increase in size and complexity of software systems**
  - Need for effective modularity
  - Need for effective abstraction mechanisms
  - Need for effective composition mechanisms



SUPPORTING VARIABILITY
– as attribute of modern development practices

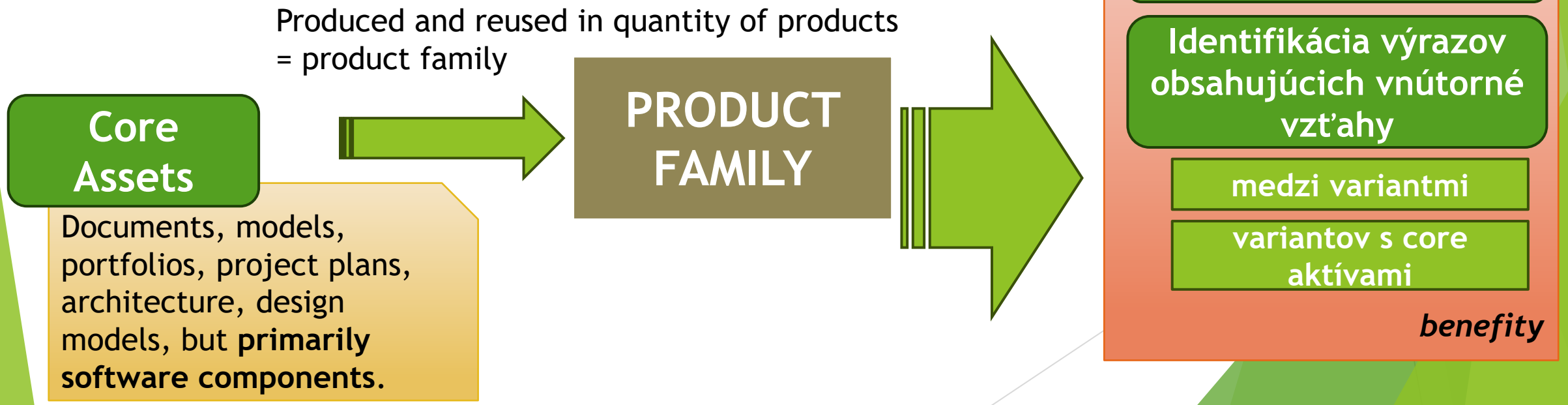


Speeding up time to market

Software Product Lines

- A **software product line** is a set of **software-intensive systems sharing a common**, **managed set of features** that **satisfy the specific needs** of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way

Systematically managing variability in a set of software products



New Dimension in Software: Variability

Software product lines as complex software intensive systems

Increased complexity:

-evolution of common assets
-independent evolution of products and their instantiation

Independent on structure, behaviour, and its state

- problem with modularization

Modularizing different kinds of variability

Preserving encapsulation and communication over explicit interfaced

Feature-Oriented Decomposition

Analysis only of those modules of the features that are available in product

Design benefits

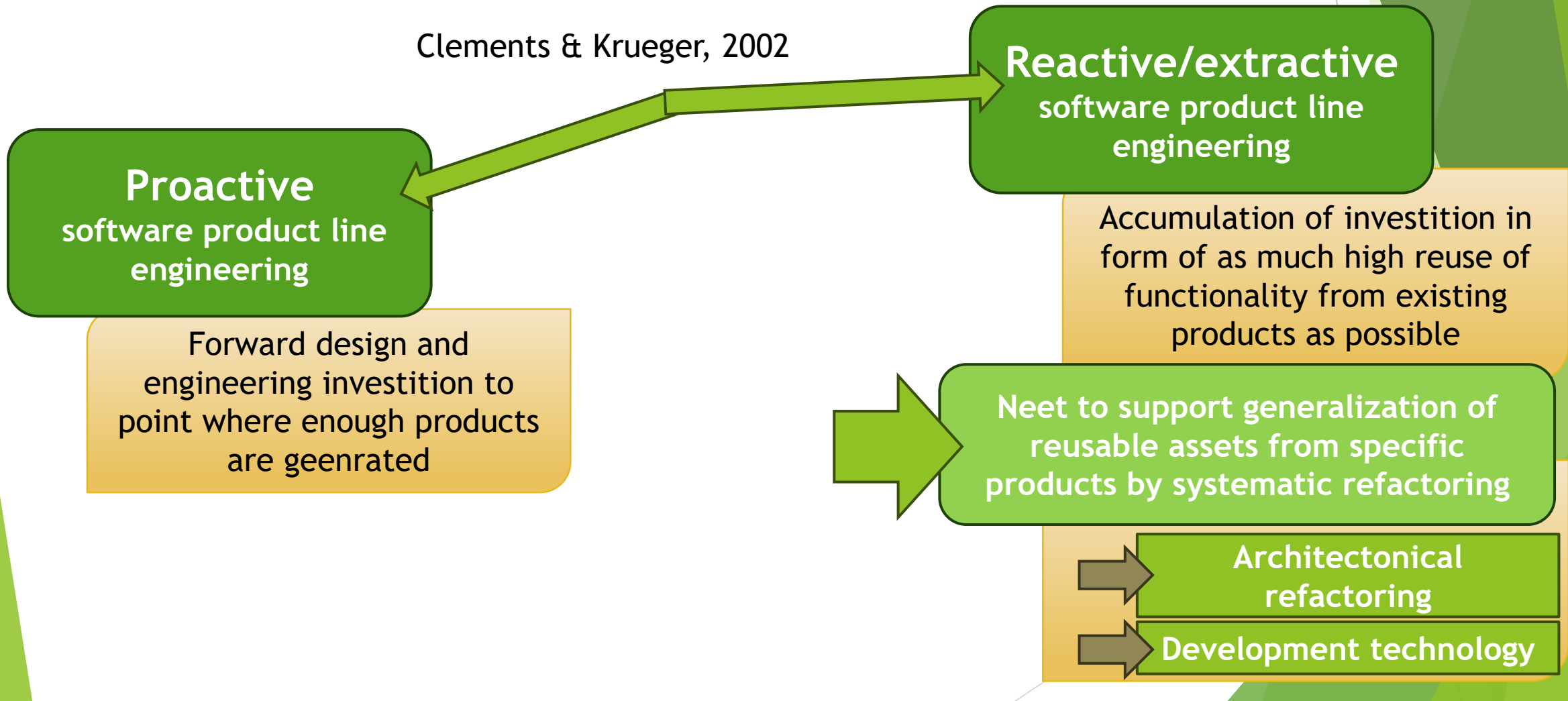
- *Independent evolution of product line features*
- *Modular introduction of new features*
- *Does not always reflect on technical commonalities between features*

Does not rely on one-to-one correspondence of features from feature model and features describing variations in software product line and implementation modules

Taken from: *Rashid, A., Royer, J.C., Rummler, A. (eds.): Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way (09 2011)*

Views on Software Product Line Development

Clements & Krueger, 2002



Problems with repeatable benefiting from software product lines

Awais Rashid, Jean-Claude Royer and Andreas Rummler 2011

1. Challenge with scaling

HIGH NUMBER OF VARIANTS

Exponential increase of inner dependencies and mutual relations

Problem to comprehend interactions between variants (from requirements to implementation)

2. Systematicity of variations

V ktorej majú tendenciu ovplyvňovať architektúru celého radu softvérových výrobkov

Tackled by AMPLE project

3. Focus on various business contexts

Each with own complexity

Product Line Engineering (PLE)

Výrobok (product)

Expressing model in PLE context

shares → COMMON CHARACTERISTICS (COMMONALITIES)

Is differentiated by → VARIABLE CHARACTERISTICS (VARIABILITY)

Managed in PLE (within platform)

Vlastnosť (feature)

Differentiates the products from each other

User can see it

Platforma (platform)

Set of components that are reused during the development

(regardless of the narrowness of their integration)

-combines and manages all available artifacts (for product creation)

-serves as technological basis

- A set of core artifacts
- Products created on top of core artifacts
- Set of technologies for product derivation

Product derivation (derivation)

Process of product creation

Technológie umožňujú jeho plnú automatizáciu

Usually Semi-automatically

User-visible aspect

Configuration

A choice amongst multiple options during the management of variable features by the platform

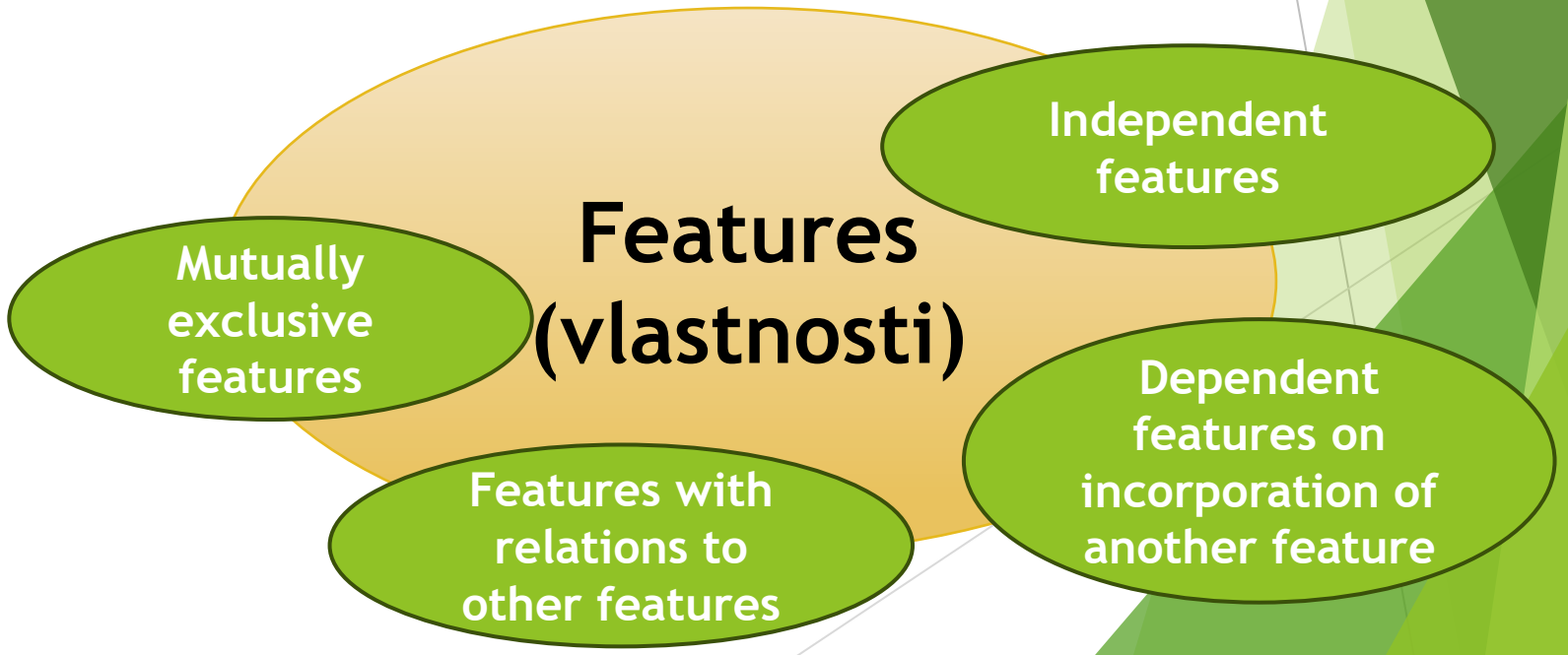
Variačný bod (Variation Point)

Characteristics exposed by a platform and modified into some extent

Varianty (Variants)

Formed when variation point is bound to specific value

Features



Domain engineering

Introduction of platform including all associated activities

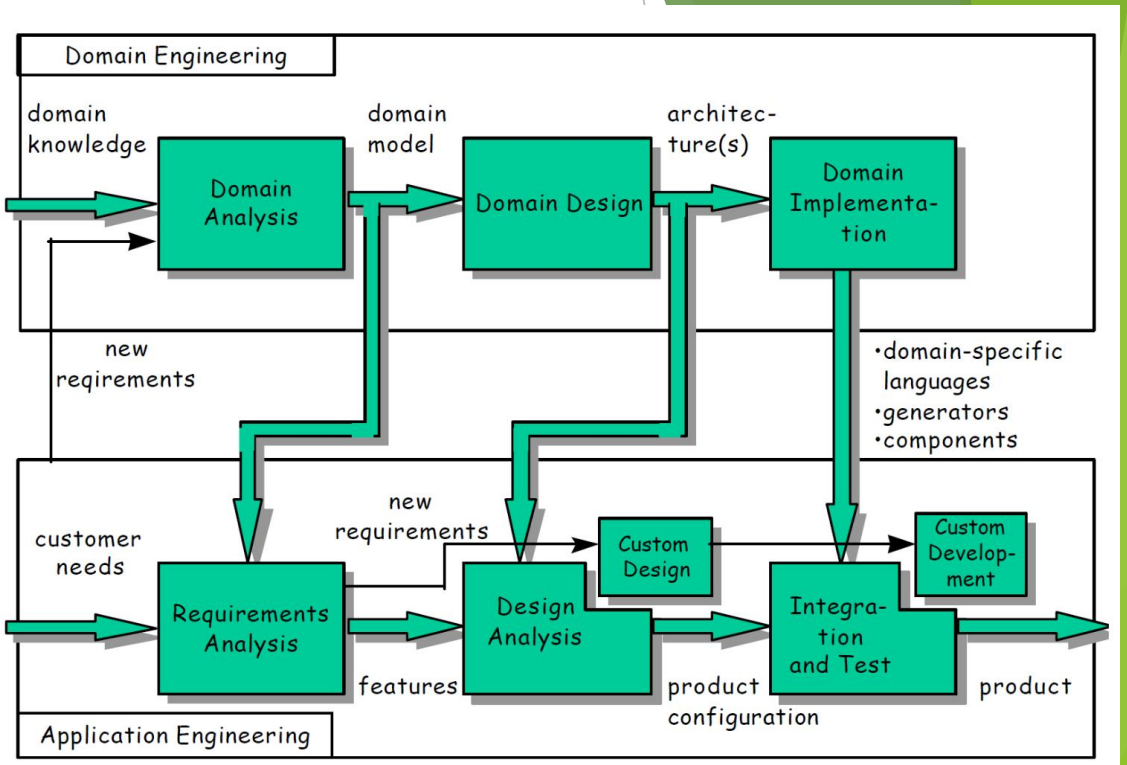
- Defining common and variable features
- Creating the components responsible for variability management
- Creation of tools for differentiating and tracing of variability (tools and methods of reusable components management)
- Defining the range in which products can be constructed

Application engineering

Creation of product/products

- Deriving products with help of tools from previous phase
- Customization of existing product: creating application specific components for one particular product

Development of components - **For reuse**



Product instantiation - **With reuse**

Software product lines and domain engineering:
K. Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. Ph.D. Thesis,

Rashid, A., Royer, J., & Rummler, A. (Eds.). (2011). *Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way*. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139003629

NEWLY REQUESTED FEATURE

Candidate for introducing the platform

Domain engineering

Application engineering

Provided by components to increase reuse

MULTIPLE PREDICTABLE FEATURES

Both phases run in parallel

Designing the platform to support application engineering

Reusable templates, elements (models, documentation, source code,…), and parts usable in application engineering

Other artifacts are on output

Uses templates and elements during the creation of resulting products

IT IS NOT ABOUT COLLECTING ELEMENTS IN REPOSITORIES AIMING TO REUSE THEM

Need to describe variability

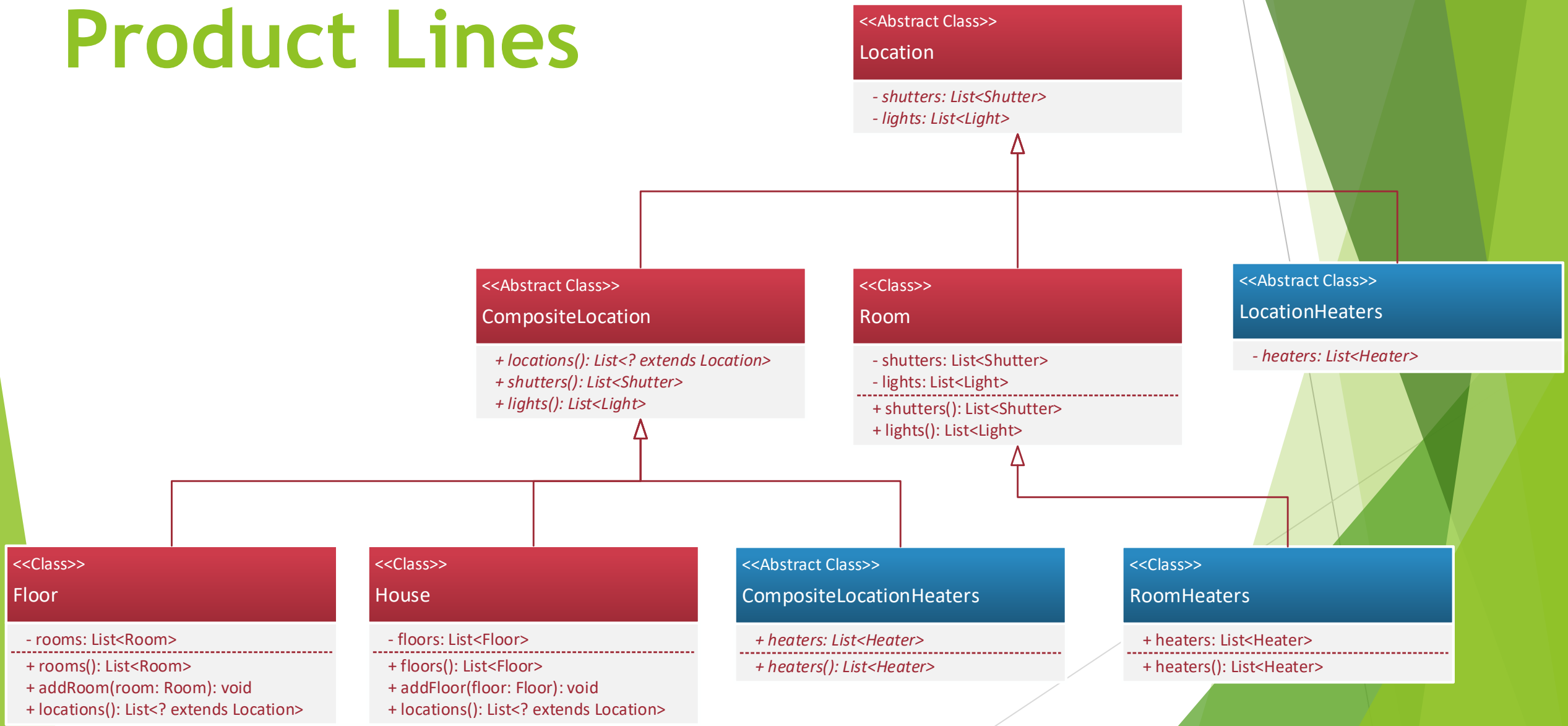
Variability modelling

Difference in comparison with software engineering of one product

-available components
-variable characteristics of components
-differentiating/managing variability

Feature models

ECaesarJ - Evolving Software Product Lines



Modularization of Static Structures

1. No replacement of classes in inheritance relationship.

-redeclaration of inheritance relationship between classes

MULTIPLE INHERITANCE Impossible in Java

- introduction of additional glue code (for example in C++)

2. Incorporation of planned functionality (domain knowledge)

INVASIVE CHANGES OTHERWISE

- *If open-closed principle cannot be used = implementation of core features has to be changed during introduction of product-specific feature*

3. Instantiation of extended classes instead of original ones

Here:

```
40
41    static House house = new House();
```

REDEFINITION IN ALL PLACES WHERE EXTENDED CLASS IS INSTANTIATED

Handling Event Using Observer Design Pattern

Modularization of behaviour

1. Much glue code – registration and notification of observers

2. Incorporation of planned functionality (domain knowledge)

INVASIVE CHANGES OTHERWISE

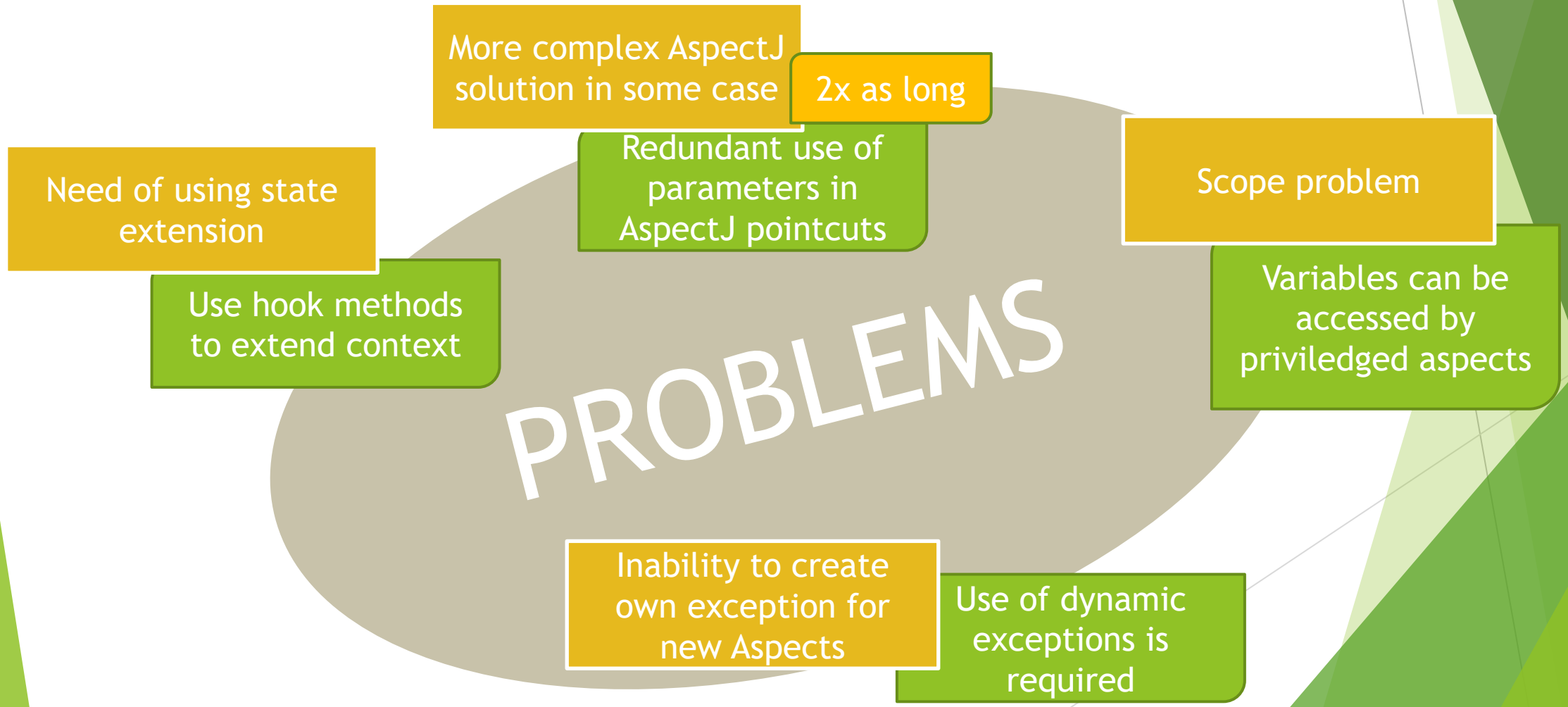
- *If open-closed principle cannot be used = implementation of core features has to be changed during introduction of product-specific feature*

3. No support of declarative definition of events

- *Difficult to reuse for defining other kind of events*

EXPLICIT TRIGGERING IS NECESSARY

Quality of Applying Voluntary Features Using AspectJ



```
1  public class IN {
2    public int insertEntry1(CR entry) { //...
3      if (nEntries < entryTargets.length) { //...
4        updateMemorySize(0, getInMemorySize(index));
5        adjustCursorsForInsert(index); //...
6      }
7    }
```

```
8  public aspect MemoryBudget {
9    before(IN in, int index):
10     call(void IN.adjustCursorsForInsert(int)) &&
           this(in) && args(index) &&
           withincode(int IN.insertEntry1(CR)) {
11     in.updateMemorySize(0, in.getInMemorySize(index));
12   }
13 }
```

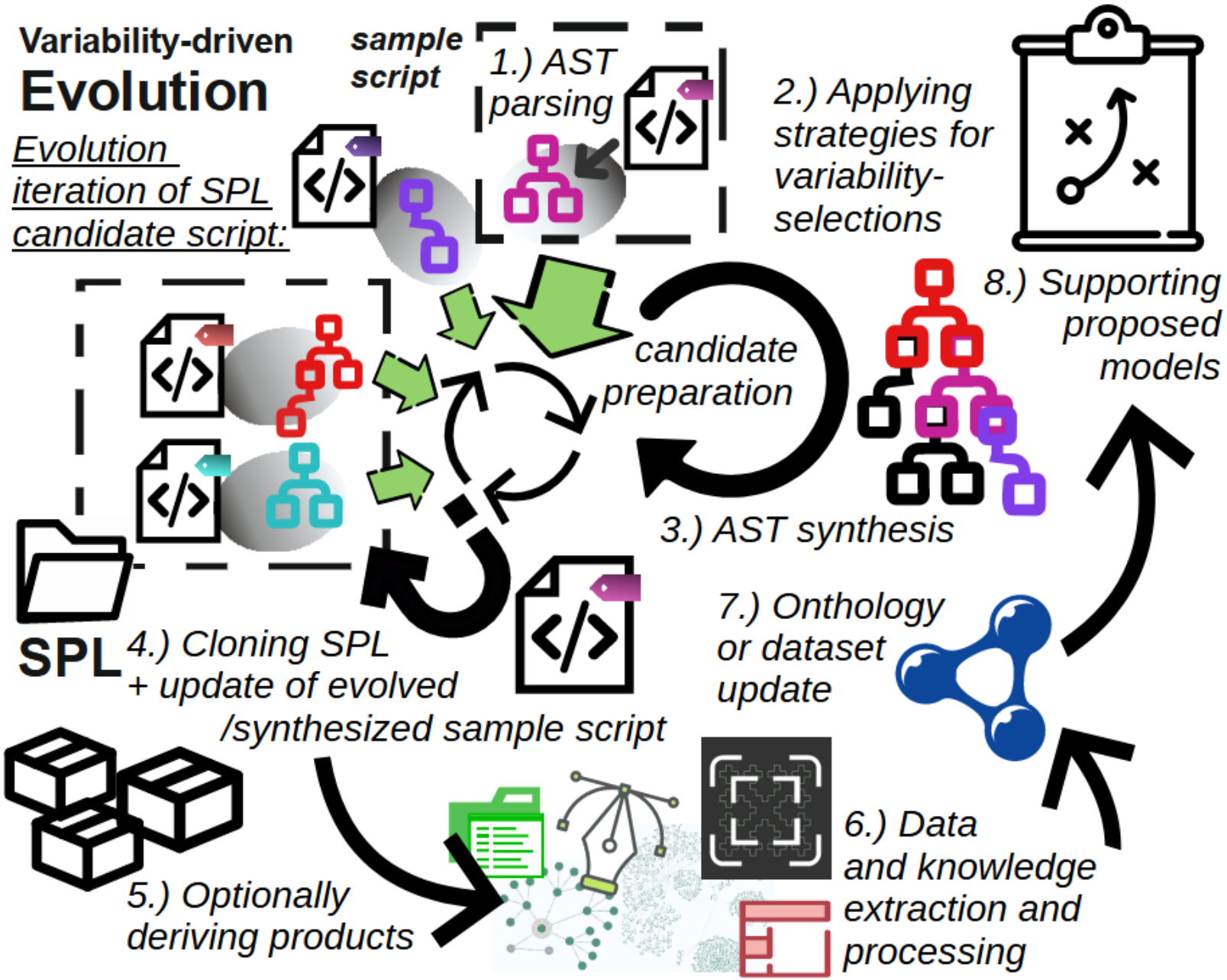
Figure 3. *Extract Before Call* **Refactoring.**

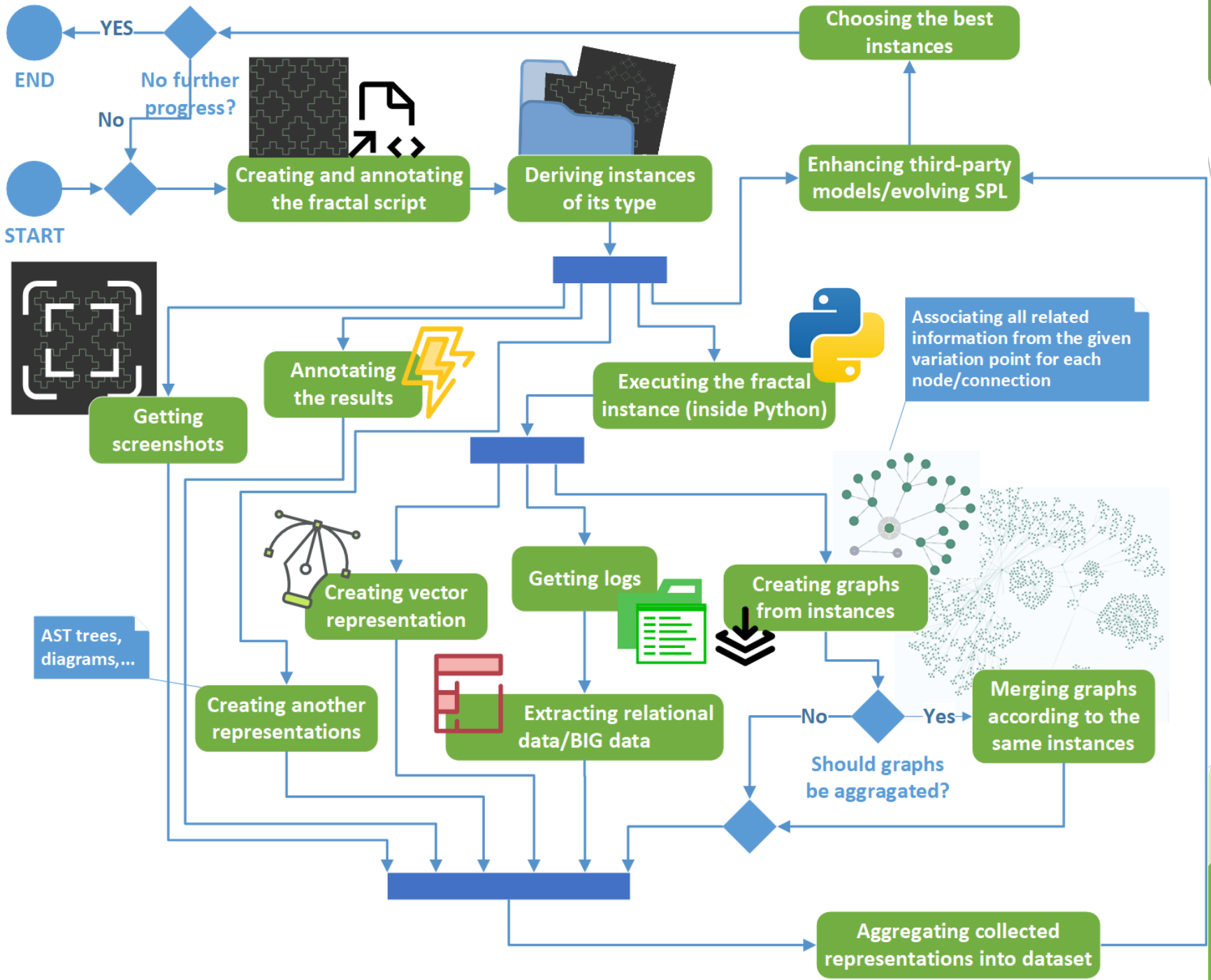
```
1  public class Tree {
2    public long insert(LeafNode ln, byte[] key, ...) {
3      BottomNode bin = findBINForInsert(key, ...);
4      long position = ln.log(key, ...);
5      bin.updateEntry(ln, position, key);
6      bin.clearKnownDeleted();
7      trace(bin, ln, position);
8      ...
9    }
10 }
```

```
11 public class Tree {
12   public long insert(LeafNode ln, byte[] key, ...) { ...
13     bin.clearKnownDeleted();
14     hook(bin, ln, position);
15     ...
16   }
17   void hook(BottomNode b, LeafNode l, long p) {}
18 }
19 public aspect TreeLogging {
20   before(BottomNode bin, LeafNode ln, long pos):
21     execution(void Tree.hook(...)) && args(bin,ln,pos) {
22     trace(bin, ln, pos)
23   }
24 }
```

Figure 5. Local Variables Access Problem.

Our Approach




YES
END
No further progress?
No
START
Creating and annotating the fractal script
Deriving instances of its type
Choosing the best instances
Enhancing third-party models/evolving SPL
Getting screenshots
Annotating the results
Executing the fractal instance (inside Python)
Associating all related information from the given variation point for each node/connection
Creating vector representation
Getting logs
Creating graphs from instances
AST trees, diagrams,...
Creating another representations
Extracting relational data/BIG data
No
Yes
Should graphs be aggragated?
Merging graphs according to the same instances
Aggregating collected representations into dataset

References

- **CaesarJ:** https://caesarj.org/
- **Product Line Implementation with ECaesarJ:** *Rashid, A., Royer, J.C., Rummler, A. (eds.): Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way (09 2011)*
- **Aspect-oriended recreation of design patterns, application of patterns:** *R. Miles, AspectJ cookbook, 1st ed. Sebastopol, CA; Farnham: O'Reilly Media, 2004.*
- **Aspect-oriented recreation of Observer design pattern:** *E. Piveta and L. Zancanella, "Observer pattern using aspect-oriented programming," Proceedings of the Third Latin American Conference on Pattern Languages of Programming, p. 12, 12 2003*
- **Complexity of In-code Variability - Evaluating Complexity of Cariability Management Constructs:** *Perdek, Jakub, and Valentino, Vranić. "Complexity of In-Code Variability: Emergence of Detachable Decorators.", In Reuse and Software Quality (pp. 51-71). Springer Nature Switzerland, 2024.*
- **Framed Aspects:** *Loughran, N., Rashid, A.: Framed aspects: Supporting variability and configurability for AOP. In: Proceedings of 8th International Conference on Software Reuse, ICSR 2004. LCNS 3107, Springer, Madrid, Spain (2004)*

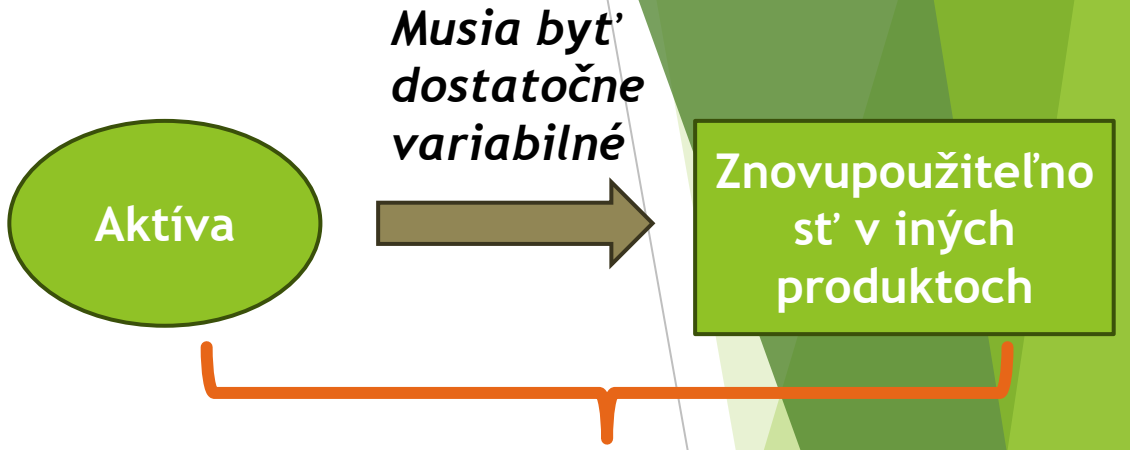
References

- **Supporting Product Line Evolution With Framed Aspect:** *Loughran, N., Rashid, A., Zhang, W., Jarzabek, S.: Supporting product line evolution with framed aspects p. 5 (2004)*
- **YOUNG, Trevor J a B MATH, 1999. Using AspectJ to Build a Software Product Line for Mobile Devices.** 1999, s. 73.
- **ZHANG, Tao, Lei DENG, Jian WU, Qiaoming ZHOU a Chunyan MA,** 2008. Some Metrics for Accessing Quality of Product Line Architecture. V: 2008 International Conference on Computer Science and Software Engineering: 2008 International Conference on Computer Science and Software Engineering [online]. Wuhan, China: IEEE, s. 500–503. ISBN 978-0-7695-3336-0. Dostupné na: doi:10.1109/CSSE.2008.500
- **FILMAN, Robert E a Daniel P FRIEDMAN. Aspect-Oriented Programming is Quantification and Obliviousness.** 2001.
- **Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines.** IEEE Computer 41, 93–95 (01 2008). Paper
- **Alves, V., Jr, P.M., Borba, P.: An Incremental Aspect-Oriented Product Line Method for J2ME Game Development** p. 3 (Jan 2004)

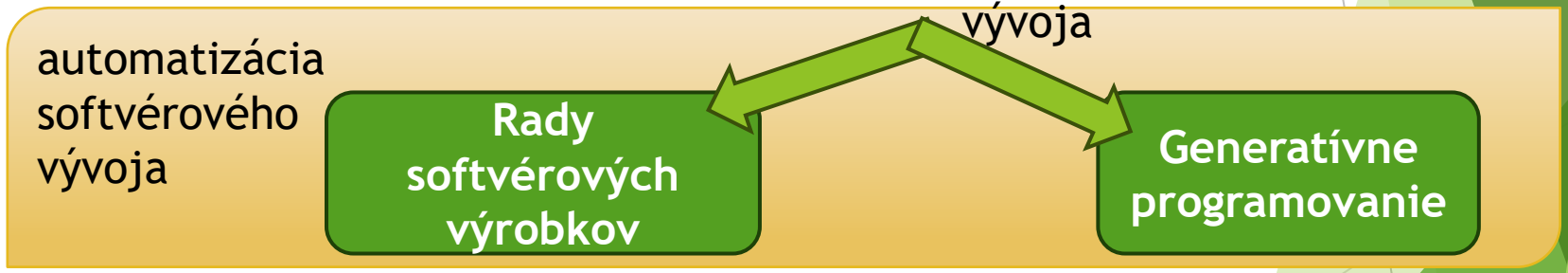
Zlepšenie znovupoužiteľnosti

Použitia tej istej funkcionality v inom kontexte

- Nárast komplexnosti a veľkosti systémov
  - potreba efektívnej modulárnosti
  - potreba efektívnych mechanizmov abstrakcie
  - potreba efektívnych kompozičných mechanizmov



PODPORA VARIABILITY
– ako atribút moderných praktík vývoja

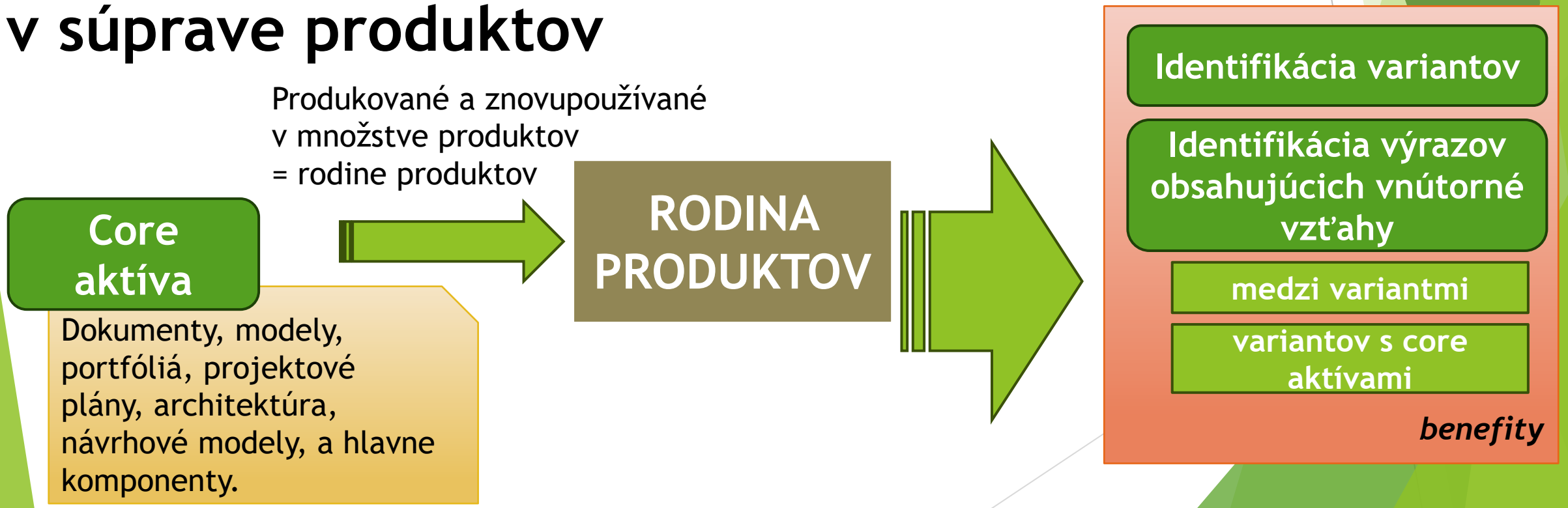


Urýchlenie času predaja
– time to market

Rady softvérových výrobkov

- A **software product line** is a set of **software-intensive systems** **sharing a common**, **managed set of features** that **satisfy the specific needs** of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way

Systematické manažovanie variability v súprave produktov



Pohľady na vývoj radov softvérových výrobkov

Clements & Krueger, 2002

Proaktívne
inžinierstvo radov softvérových výrobkov

Dopredný návrh a inžinierska investícia do bodu keď sa vygeneruje dostatočné množstvo výrobkov

Reaktívne/extraktívne inžinierstvo radov softvérových výrobkov

Akumulácia investícií v podobe čo najväčšieho znovupoužitia súčastí/funkcionality z existujúcich výrobkov

Potreba podpory zovšeobecnenia znovupoužiteľných aktív z konkrétnych výrobkov systematickým refaktorovaním

- **Architektonický refaktoring**
- **Vývojová technológia**

Problémy v opakovanom benefitovaní z radov softvérových výrobkov

Awais Rashid, Jean-Claude Royer and Andreas Rummler 2011

1. Výzva so škálovaním

VYSOKÝ POČET VARIANTOV

Exponenciálny nárast vnútorných závislostí a vzájomných vzťahov

Problém pochopiť interakcie medzi variantmi
(od požiadaviek k implementácii)

2. Systematickosť variácií

V ktorej majú tendenciu ovplyvňovať architektúru celého radu softvérových výrobkov

Riešené v AMPLE projekte

3. Zameranie sa na rozličné biznis kontexty

Každý s vlastnou zložitosťou

Inžinierstvo radov softvérových výrobkov

Product line engineering (PLE)

Výrobok (product)

Označenie modelu v kontexte PLE

Zdieľa → **SPOLOČNÉ CHARAKTERISTIKY (COMMONALITIES)**

Je rozlíšený → **ROZDIELNE CHARAKTERISTIKY (VARIABILITY)**

Manažované PLE (v rámci platformy)

Odvodenie výrobkov (derivation)

Proces tvorby produktov

Technológie umožňujú jeho plnú automatizáciu

Zvyčajne Semi-automaticky

Vlastnosť (feature)

Odlíšenie výrobkov jeden od druhého

Istým spôsobom viditeľná/rozlíšiteľná používateľom

User-visible aspect

Platforma (platform)

Množina komponentov znovupoužitých počas vývoja

(bez ohľadu na úzkosť ich integrácie)

-kombinuje a manažuje všetky dostupné artefakty (pre vytvorenie výrobku)

-slúži ako technologická

Množina core artefaktov

Výrobky vytvárané na vrchu core artefaktov

Množina technológií pre odvodenie výrobkov

Konfigurácia

Výber medzi viacerými možnosťami pri manažovaní variabilných vlastností platformou

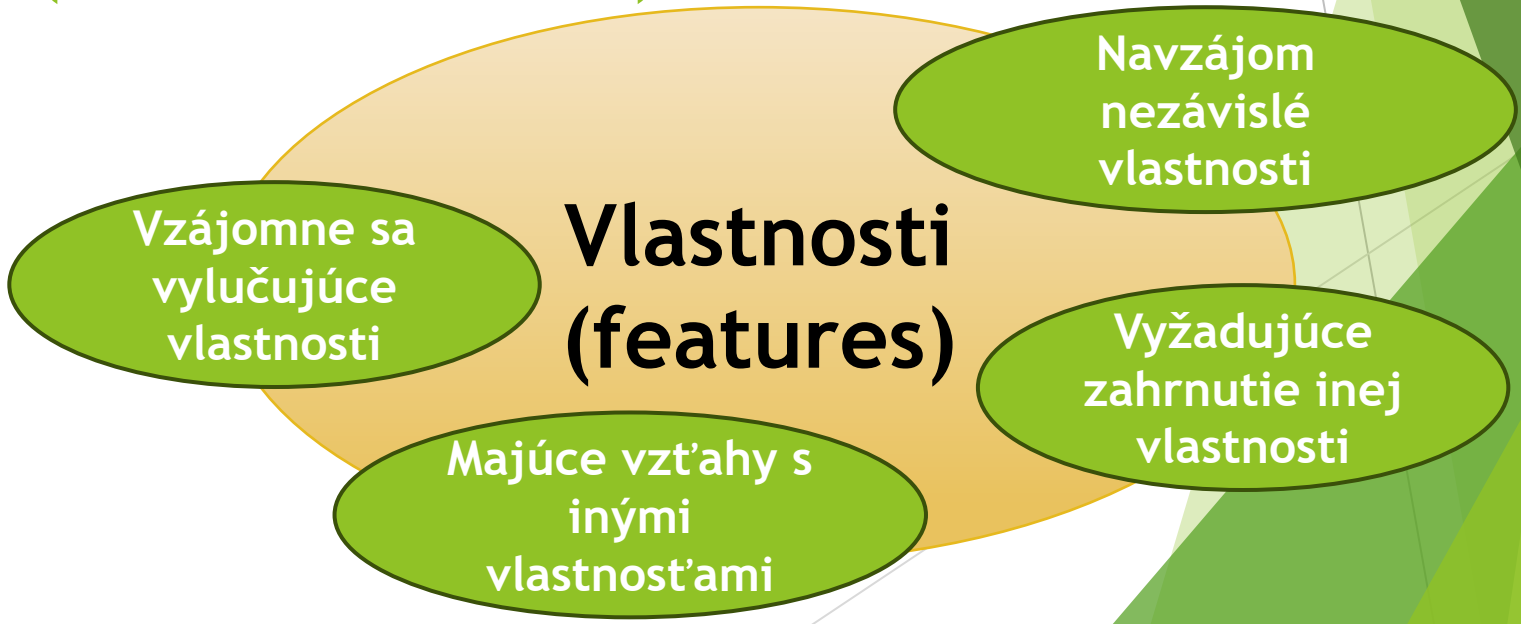
Variačný bod (Variation Point)

Charakteristika vystavená/ exponovaná platformou a pozmeniteľná do istej miery

Varianty (Variants)

Formované keď variačný bod sa naviaže s istou hodnotou

Vlastnosti (features)



Doménové inžinierstvo

Proces zavedenia platformy ane všetkých pridružených aktivít

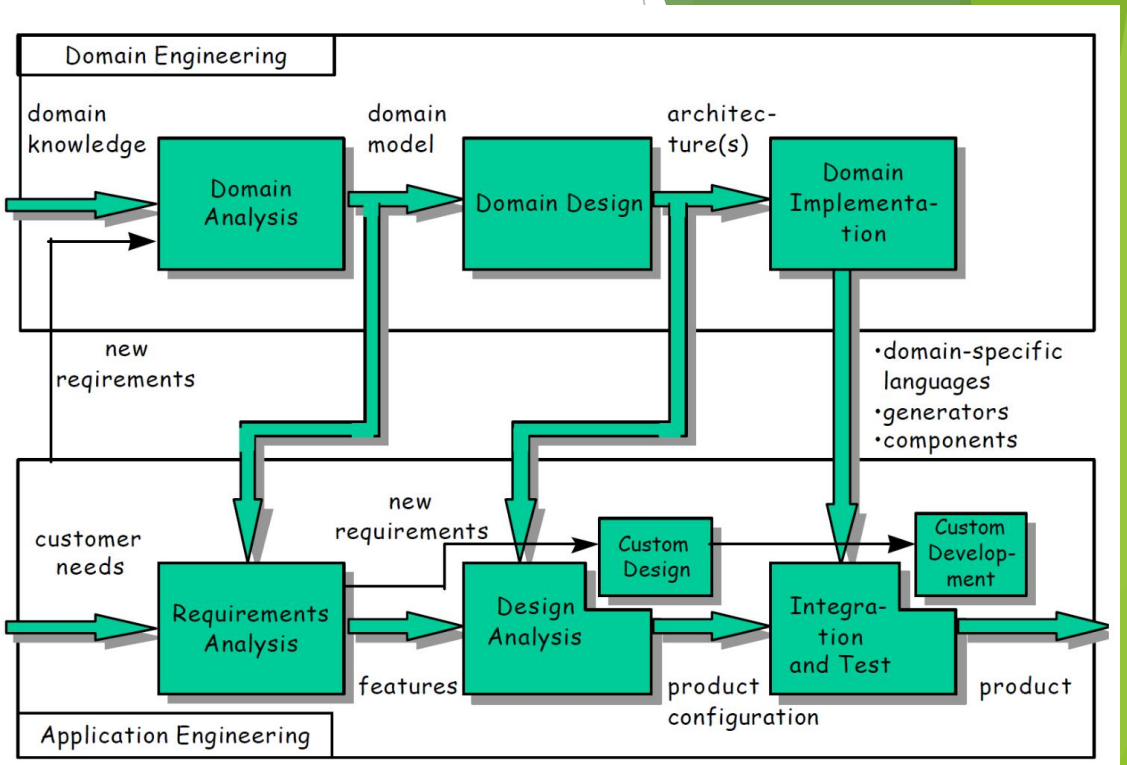
- Definovanie spoločných a rozdielnych čŕt
- Tvorba komponentov zodpovedných za manažovanie variability
- Tvorba nástrojov pre rozlíšenie a amapovanie variability (nástroje a metódy manažmentu znovupoužiteľných komponentov)
- Definovanie rozsahu v akom môžu byť produkty konštruované

Aplikačné inžinierstvo

Tvorba produktu/produktov

- Odvodenie produktov za pomoci nástrojov z predchádzajúcej fázy
- Kustomizácia existujúceho výrobku: tvorba aplikačno-špecifických komponentov v rámci iba jedného výrobku

Development of components - **For reuse**



Product instantiation - **With reuse**

Software product lines and domain engineering:
K. Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. Ph.D. Thesis,

Rashid, A., Royer, J., & Rummler, A. (Eds.). (2011). *Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way*. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139003629

NOVO-VYŽIADANÁ VLASTNOSŤ

Kandidát na zavedenie do platformy

Doménové inžinierstvo

Aplikačné inžinierstvo

Poskytnutie komponentmi pre zvýšenie úrovne znovupoužitia

VIACERÉ PREDVÍDATEĽNÉ VLASTNOSTI

Obe fázy bežia paralelne

Návrh platformy pre podporu aplikačného inžinierstva

Znovupoužiteľné šablóny, elementy (modely, dokumentácia, zdrojový kód,...) , a časti použiteľné v aplikačnom inžinierstve

Výstupom sú iné artefakty

Využíva šablóny a elementy pri tvorbe výsledných produktov

NIE JE TO O ZBIERANÍ ELEMENTOV V REPOZITÁROCH S CIEĽOM ICH ZNOVUPOUŽIŤ

Nutný popis variability

Modelovanie variability

Rozdiel oproti bežnému softvérovému inžinierstvu jedného výrobku

-dostupných komponentov
-variabilné charakteristiky komponentov
-rozlíšenie/manažovanie variability

Modely vlastností