Title: Discovering Mereological Relationships in Java

Student: Rezart Veliaj

Supervisor: Dr James Power

NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

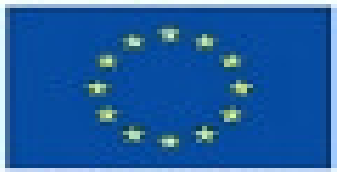


Table of Contents…

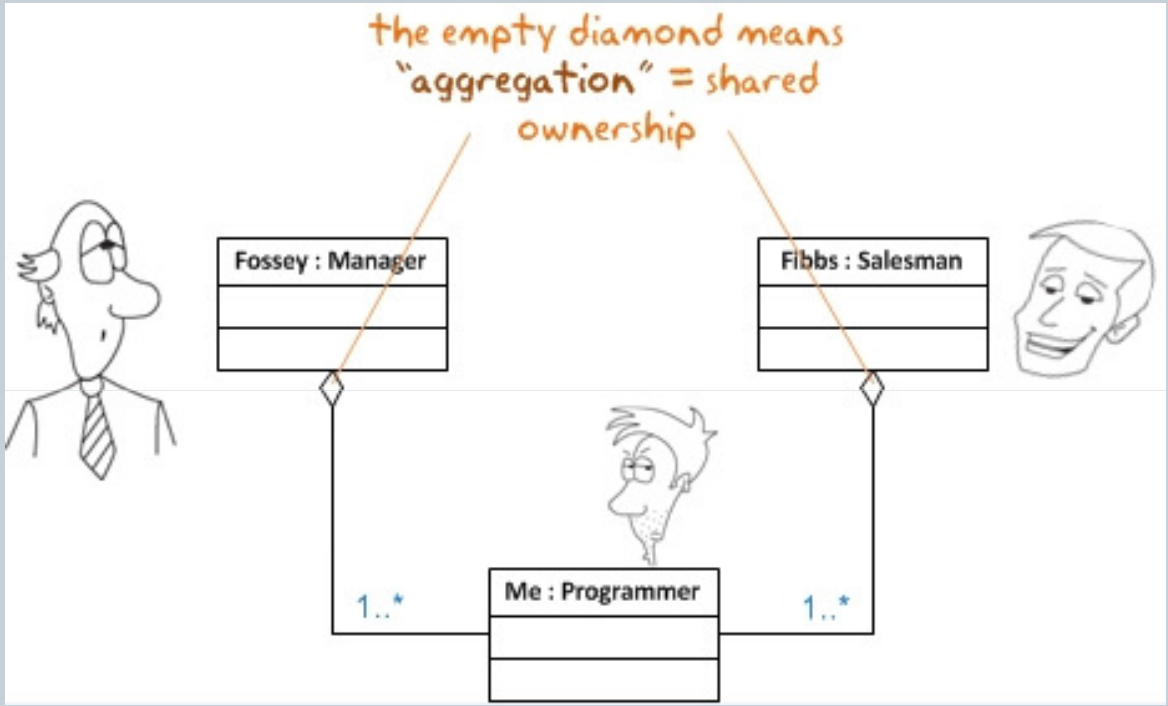
1.INTRODUCTION

- **What is mereology?**
- **Background …**
- **Problem statement…**

Mereology vs Set Theory

- The word “mereology” derives from the Greek words: μέρος, root: μερε(σ)-, "part" and the suffix -logy "study, discussion, science") and is centred around the parts and the wholes they form.
- **Mereology vs Set theory?**
- Mereology emphasizes the relation between the entities and differs from the set theory that is centered on the relation between a set and its elements (Stanford, 2009).

Connection of mereology with binary relationships:



Binary relationships can be expressed as well, in terms of ownership and whole/part relation, so we can express and define binary relationships in terms of mereology notions.

Mereological relationships are reflexive, anti-symmetric and transitive

- **Reflexive-Everything is part of itself:**
  part_of(x,x)

- **Anti-symmetric-If x is part of y and y is part of x , then x and y are the same**
  (part_of(x,y) ^ part_of(y,x) => x=y)

- **Transitive-If x is part of y and y is part of z , then x is part of z**
  (part_of(x,y)^part_of(y,z)) => part_of(x,z)

**part_of(a,b) means a is part of b*

UML representation vs implementation

For many UML tools like ArgoUML, Rational Rose, StarUML, etc which describe the source code in higher level there is never a total precision (1-to-1 mapping) between the code itself and the UML representation they give.

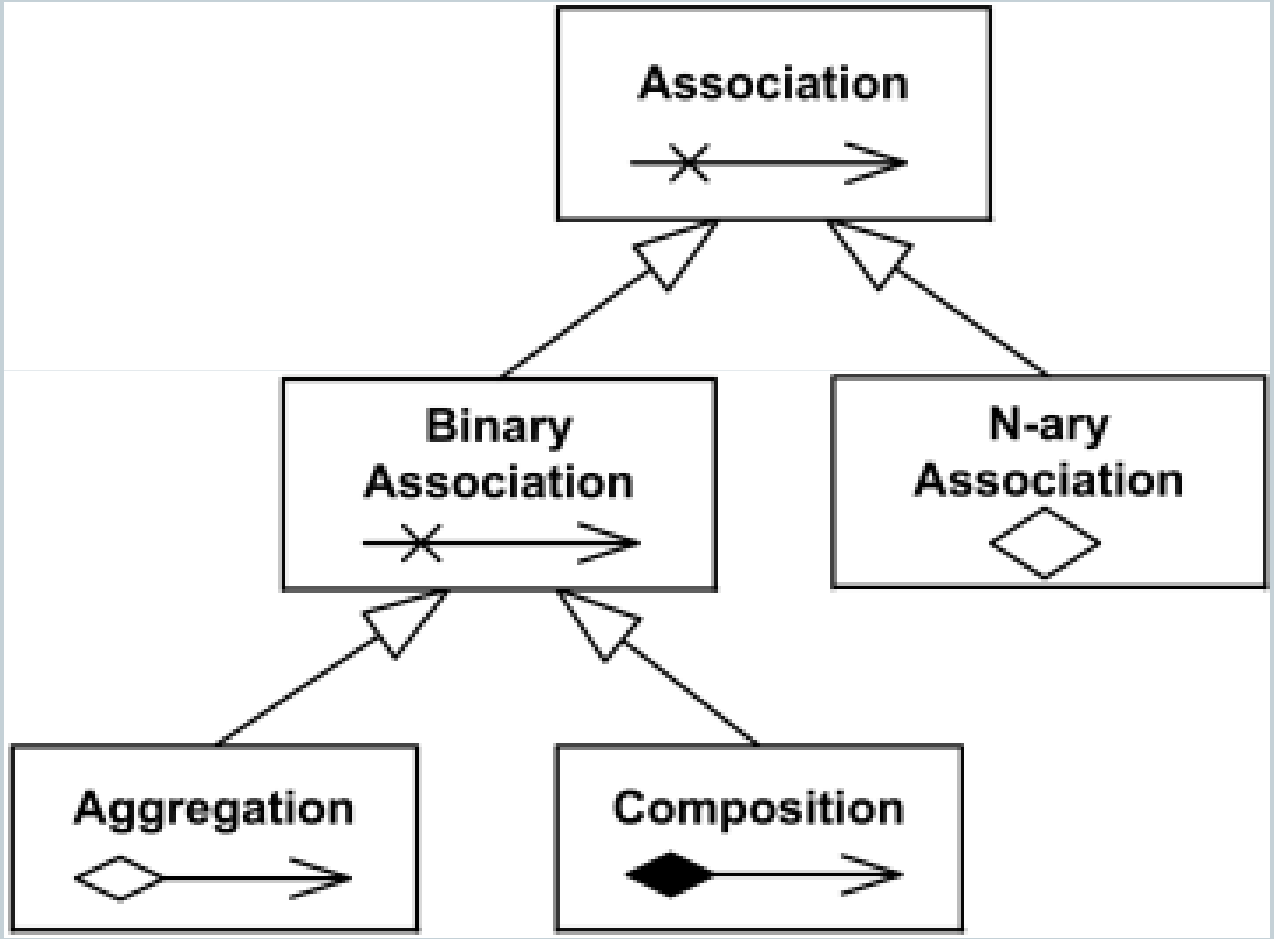
……..! 100% PRECISE

•Detection of binary relationships is an open research field and different methods and solutions are developed in bridging the gap between the design and the source code.

2.Binary relationships…

- **Classification**
- **Properties**
- **Re-definitions based on the properties**

Common relationships in programming practices:



4 Properties to define binary relationships:

1. **Multiplicity (A,B)- Denotes the number of class B which are allowed to be in relation with class A**

2. **InvocationSite(A,B)- It is concerned with the messages that instances of class B can send to class A**

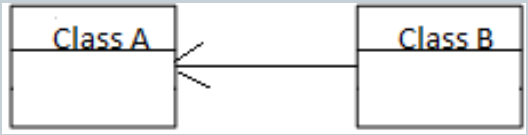
3. **Lifetime(A,B) – It expresses the lifetime of instances of class B with respect to instances of class A**

4. **Exclusivity(A,B) – It describes the number of instances of class B which can be involved at the same time with the instances of class A.**

Association relationship

- It is the least strict binary relationship among the three.
- It is enough to know that there is a message sending from class B to class A. It can be bi-directional as well.

Bi-directional Association

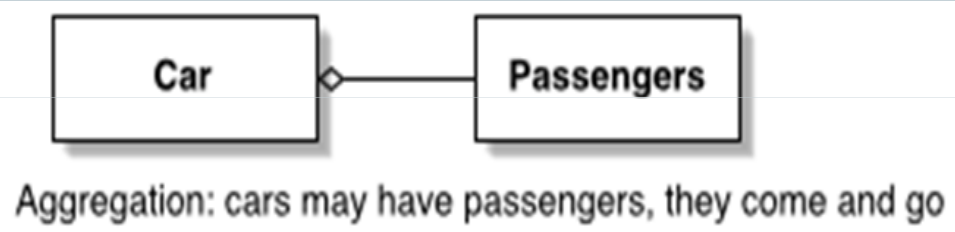


One-directional Association

Definition of association based on the 4 properties:
AS (A, B) =
EX (A, B) ∈ B
IS (A, B) = any
LT (A, B) ∈ -
MU (A, B) = [0, +∞]

Aggregation relationship

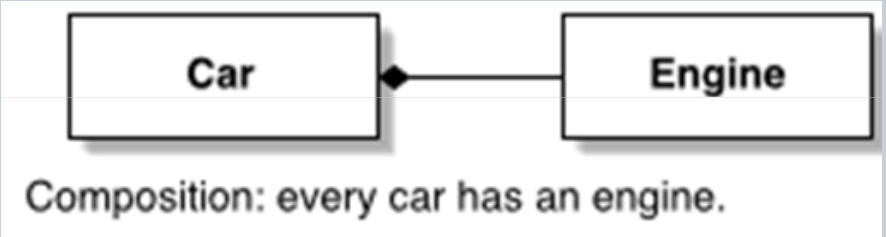
It is stronger than association, since it requires only one instance of variable which can be of any type field, array or collection. Exclusivity and lifetime are not mandatory properties.



Definition of aggregation
Based on 4 properties:
AG (A, B) =
EX (A, B) ∈ B
IS (A, B) ⊆ {field, array field, collection field}
LT (A, B) ∈ -
MU (A, B) = [0, +∞]

Composition relationship

Composition is the strongest among the 3 relationships. Composition is a type of aggregation, which have the restriction that the life of part is related to the one of the whole, as well in terms of exclusivity a part can be part of 1 and only 1 whole.



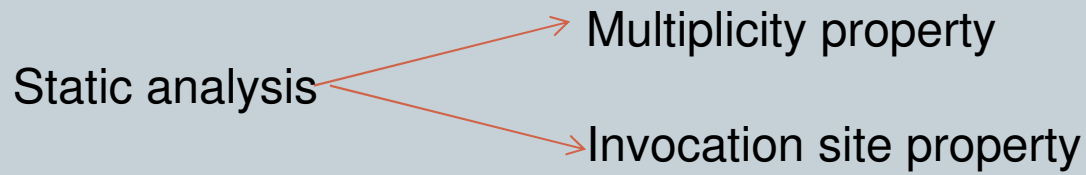
Definition on composition based on the 4 properties:
CO (A, B) =
EX (A, B) = true)
IS (A, B) ⊆ {field, array field, collection field})
LT (A, B) = +
MU (A, B) = [1, +∞]

3.Static and dynamic analysis of java code

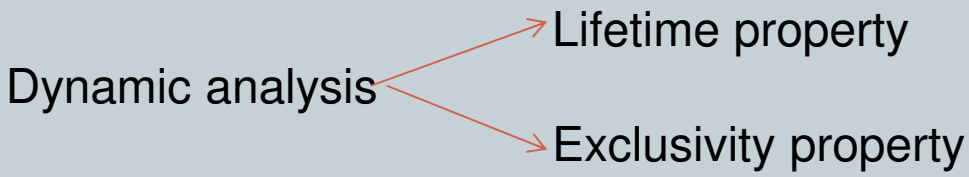
- **Static analysis**
- **Dynamic analysis**
- **ASM framework**
- **Design of solution**

Detecting binary relationships in practice

-Static analysis=>It is the code analysis performed without running the program. It can be made on the source code directly or a level lower in bytecode.



-Dynamic analysis=> It is analysis performed while the program is running. In our case will be achieved through bytecode instrumentation.

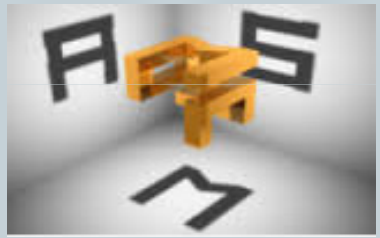


ASM Framewok

•It comes with an API that is simple, well designed and modular.

•ASM relies on a new approach that consists in using the “visitor” design pattern. (No use of trees to represent!)

• It provides support for Java 7, the latest Java version.



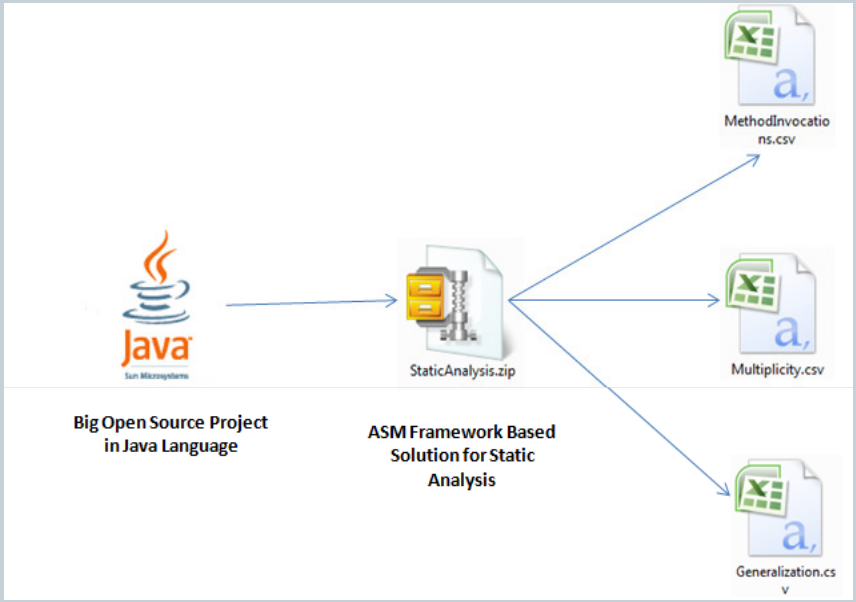
• It is small, but very fast and robust at the same time.

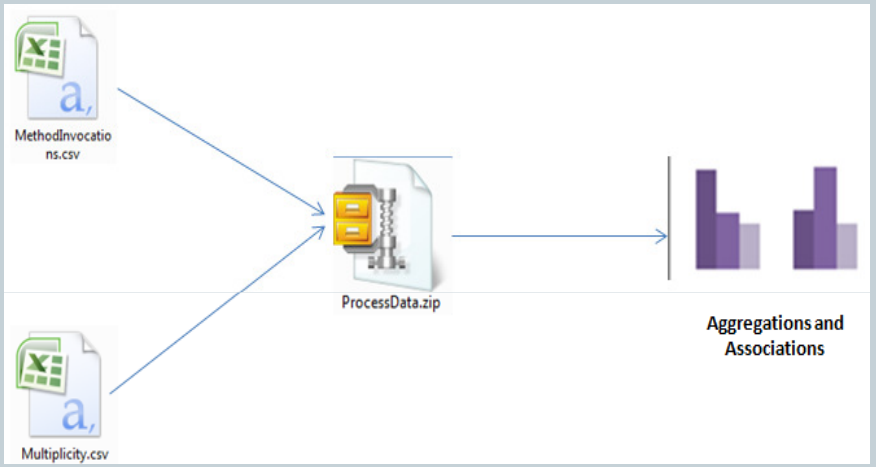
• It has a large user community behind.

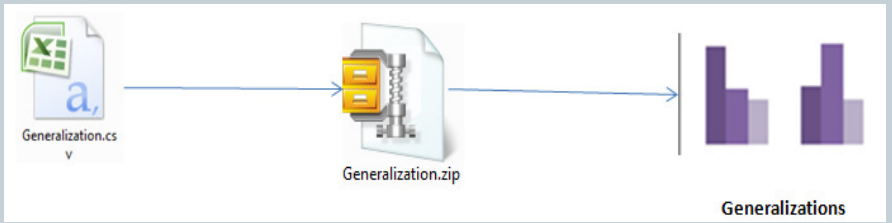
• ASM is very well documented and has a plug-in for Eclipse.

•Moreover, it has an open source license that gives any end user the freedom to use it any way they want to.

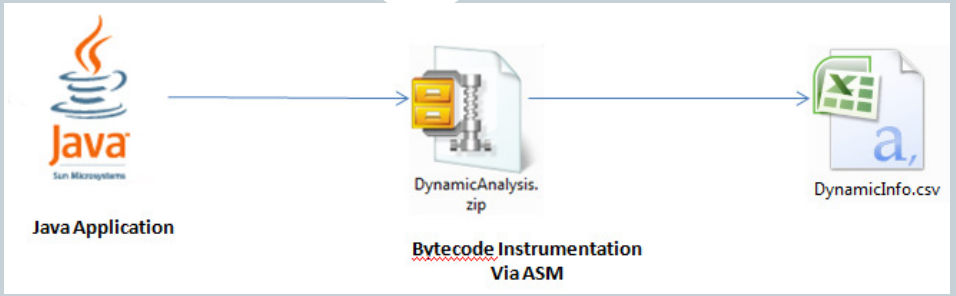
Static analysis(detecting Associations and Generalizations)







Dynamic analysis (Detecting between aggregations and compositions)



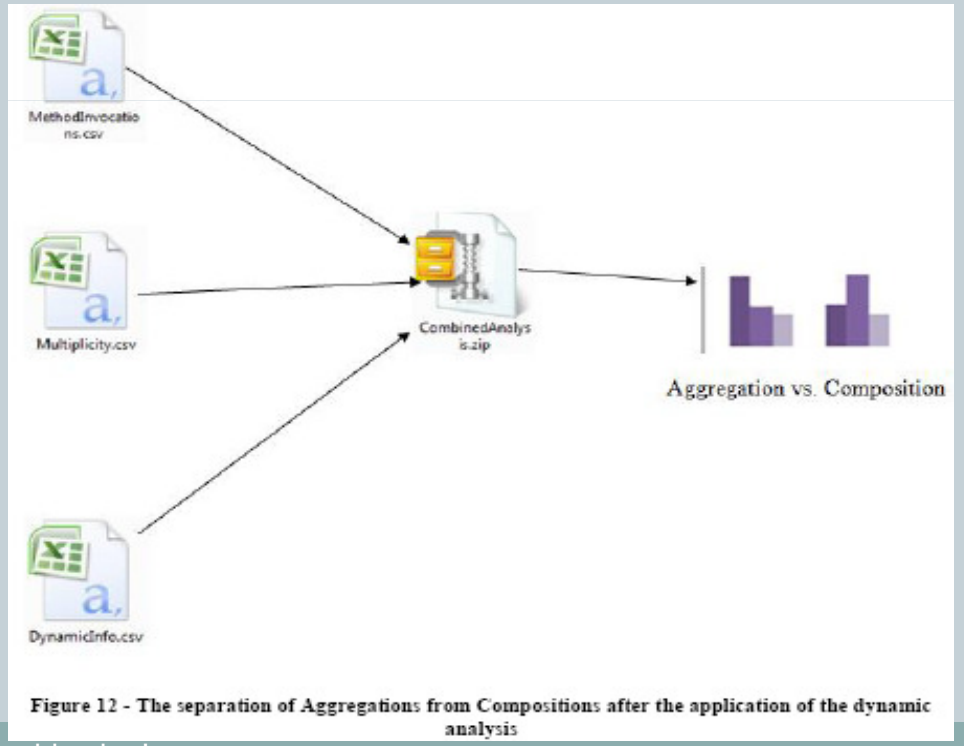
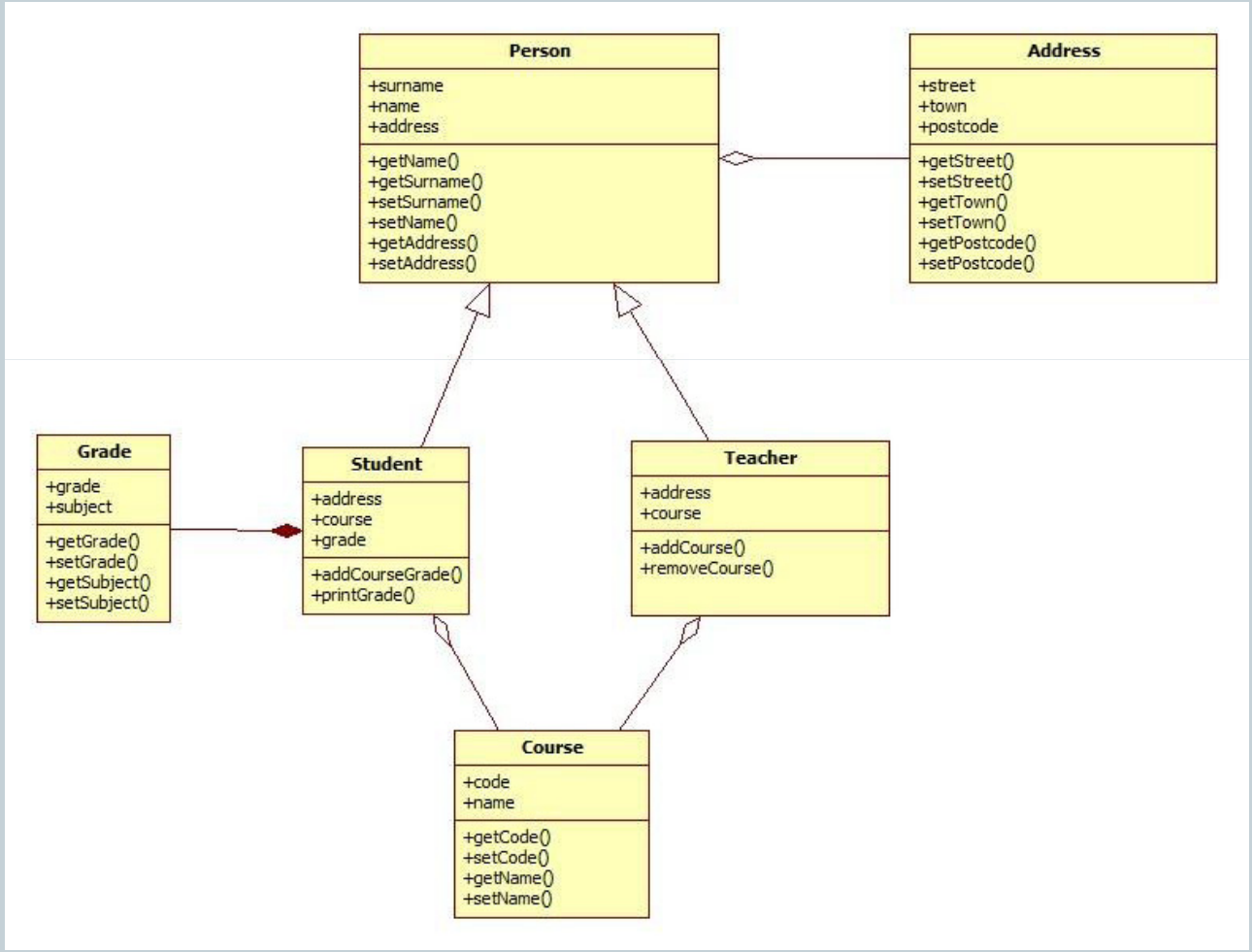


Figure 12 - The separation of Aggregations from Compositions after the application of the dynamic analysis

4.Proof of concept by Example

- **Schema of the example program**
- **Generating files**
- **Analyzing data**

Class(UML) diagram of example program



Static analysis through the ASM based tools

Detection of Generalizations

Class	Superclass
Course	No
Grade	No
Teacher	Person
Address	No
Person	No
Student	Person

The information obtained regarding method invocation is given in the below:

INVOKESPECIAL,<init>,Teacher,Person
INVOKESPECIAL,<init>,Teacher,Course
INVOKESPECIAL,<init>,Teacher,Course
INVOKESPECIAL,<init>,Student,Person
INVOKESPECIAL,<init>,Student,Grade
INVOKEVIRTUAL,getName,Student,Course
INVOKEVIRTUAL,getGrade,Student,Grade

Multiplicity

Current Class	Accessability	varName	Type	Generics	Multiplicity
Teacher	private	course	Course	Not using generics	0..1
Person	private	address	Address	Not using generics	0..1
Student	private	course	Course	Not using generics	0..1
Student	private	grade	Grade	Not using generics	0..1

Dynamic analysis after bytecode instrumentation

Detection of lifetime property

Event	Class	Hash code (in case Object Create)
Entering,Student.<init>()	Student	
Object Create Grade	Student	3288014
Exiting,Student.<init>()	Student	

Detection of exclusivity property

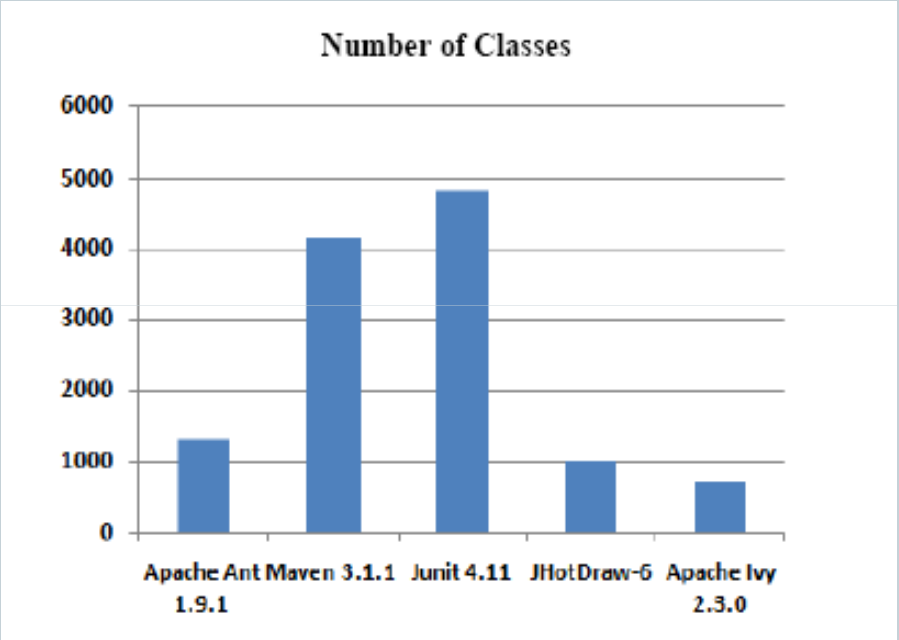
	Event Type	Class	Hash code
1	Object Create	Student	3325285
2	Object Create	Course	29525730
3	Object Create	Grade	29089096
4	Object Create	Student	5912867
5	Object Create	Grade	27766975
6	Object Create	Student	30866355
7	Object Create	Grade	2102960
8	Object Create	Teacher	12582949
9	Object Create	Teacher	30587319
10	Object Create	Address	18615648

5.Statistics and results (static analysis on big open source projects)

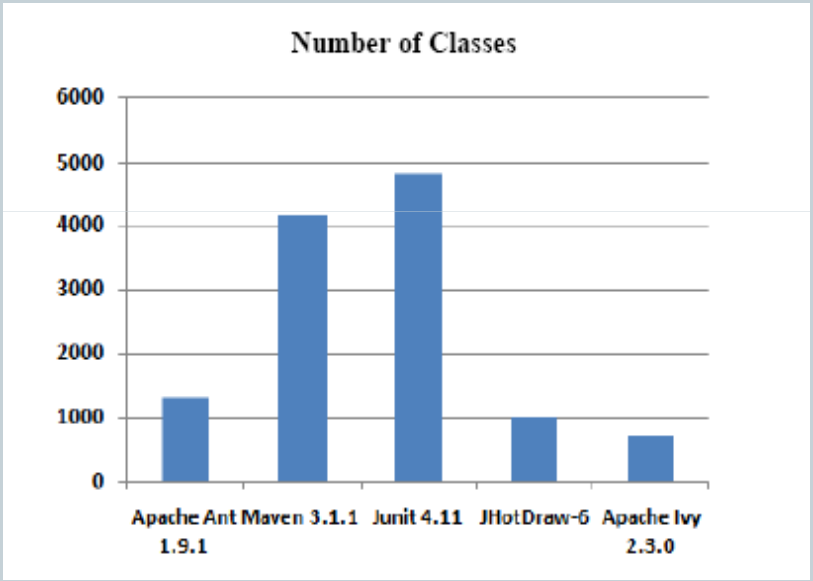
- **Charts on program size**
- **Number of generalization**
- **Bi-directional associations**
- **One directional associations**
- **Aggregation with Composition**

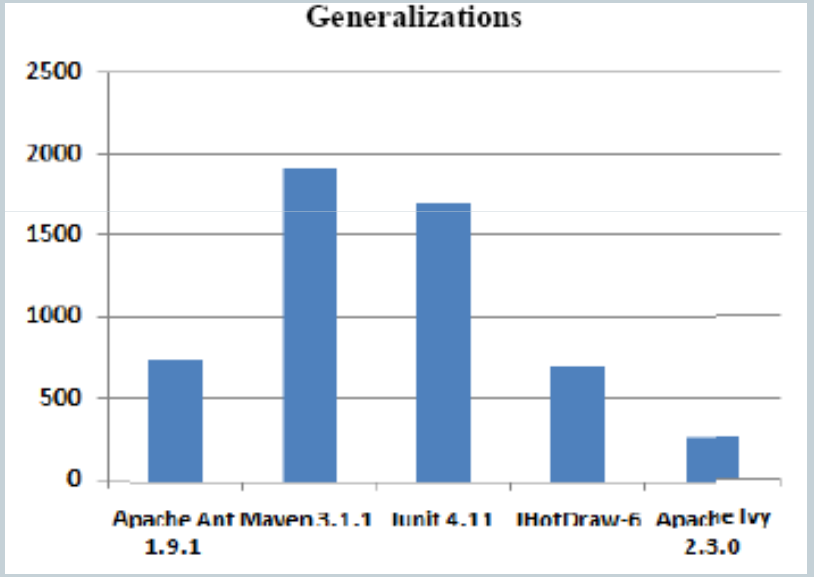
Projects chosen and their size (in terms of number of classes)

- **Apache Maven 3.1.1**
- **Junit 4.11**
- **Apache Ant 1.9.1**
- **HotDraw 6 beta version**
- **Apachelvy 2.3.0**



Generalizations



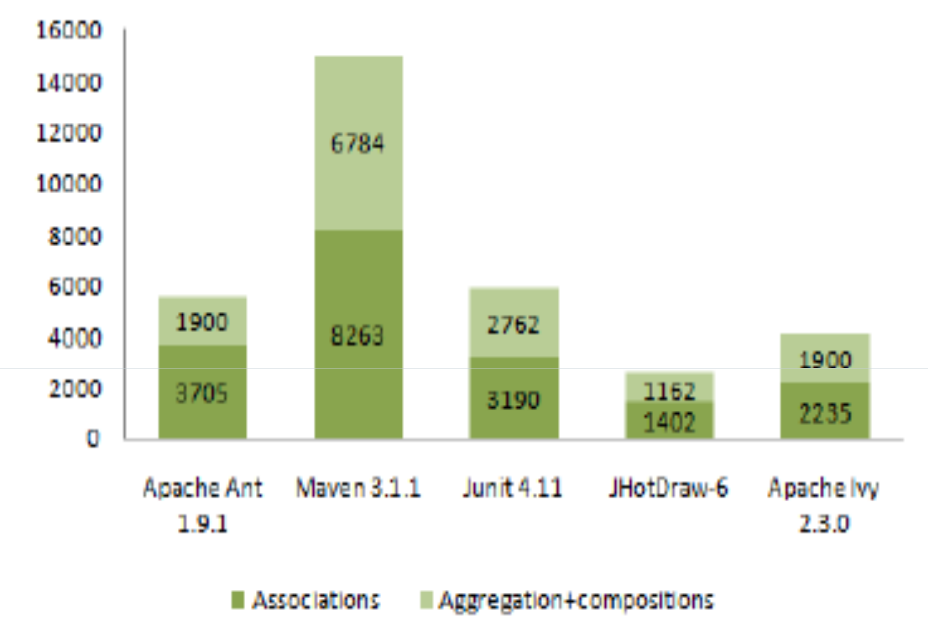


Bidirectional relationships

Bidirectional relationships

Project	Number of Bi-directional relationships
Apache Ant 1.9.1	49
Maven 3.1.1	50
Junit 4.11	11
JHotDraw-6	1
Apache Ivy 2.3.0	17

Associations and Aggregations



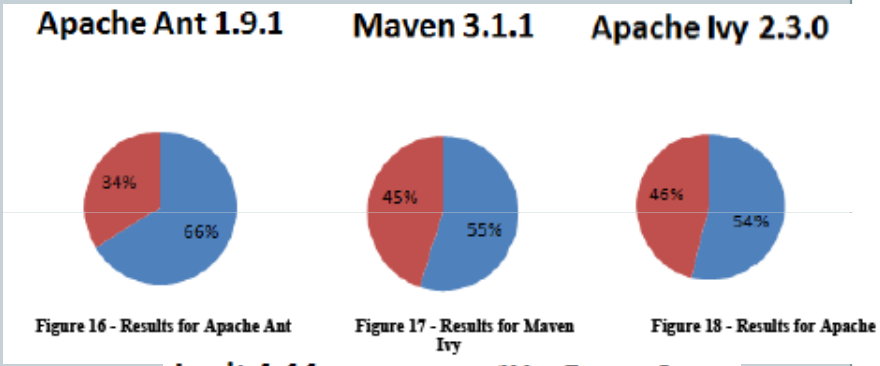


Figure 16 - Results for Apache Ant

Figure 17 - Results for Maven

Figure 18 - Results for Apache Ivy

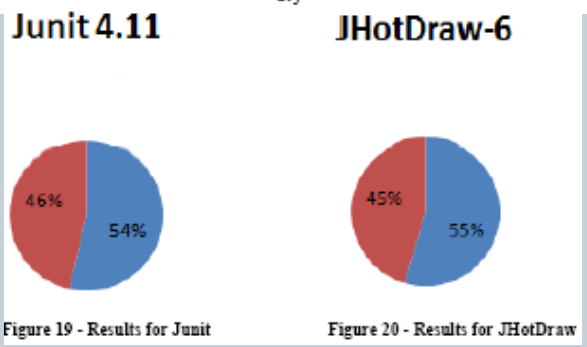


Figure 19 - Results for Junit

Figure 20 - Results for JHotDraw

6.Conclusions

- **Analytical overview of results**
- **Threat to validity**
- **Future work**

Some conclusions

- ***It seems that there is a connection between the number of classes and the number of generalizations.***

- ***Number of associations is greater than aggregations with compositions.***

- ***No direct connection between bi-directional relationships and the number of classes.***

- ***The study was proven to be 100% correct in the case of a small example, since the relationships were detected accurately by combing both the static and dynamic analysis.***

Threat to validity

- ***A manual check of the accuracy — error prone? ☺***

- ***Using different tools can be error prone, since the combination of two or more low error prone tools can lead to bigger errors.***

- ***For the study 5 big open source projects were taken into consideration. It is possible that for 5 other projects the results would differ.***

- ***Even more in the case of projects written in different programming languages the results are likely to change (different design patterns, etc).***

Future work

- ***Apply dynamic analysis on big open source projects to differ between aggregation vs compositions in big projects.***

- ***Compare the results with some other study done on the same projects with different methods or framework***

- ***In order to make the study more generic, more projects should be chosen for analysis(at least 10 more).***

- ***Improve UML tools with a more detailed study on the mereoligical relationships.***

7.QUESTIONS?

THANK YOU !