Ben-Gurion University of the Negev

Using Reference-based Framework for Improving the Utilization of Software Engineering Principles

Oded Kramer(odedkr@bgu.ac.il)
Department of Information Systems Engineering
Ben-Gurion University of the Negev

Arnon Sturm(sturm@bgu.ac.il)
Department of Information Systems Engineering
Ben-Gurion University of the Negev

Outline

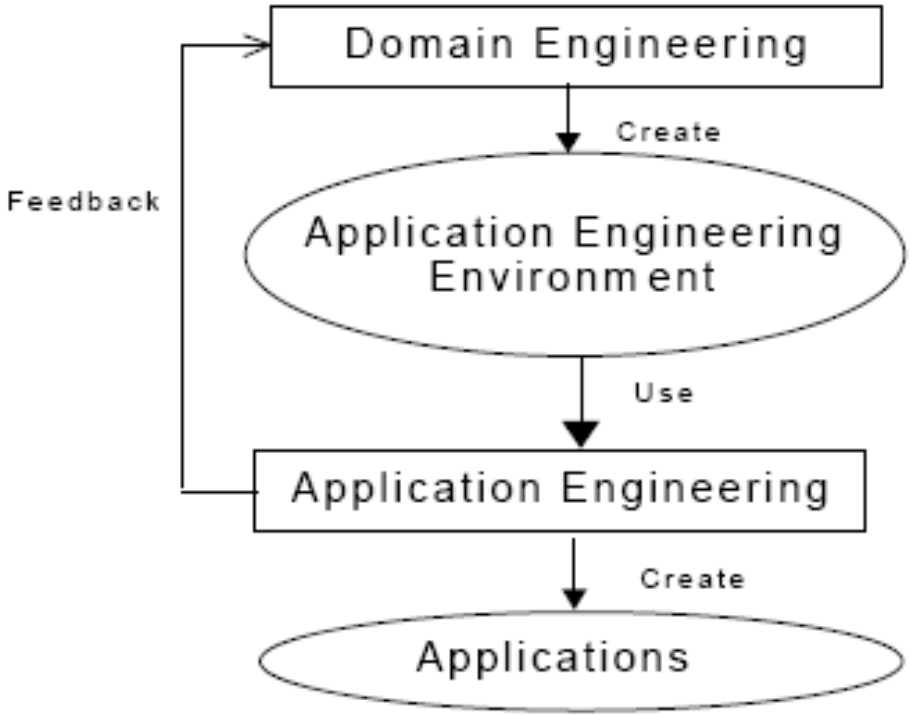
- Introduction
- Related work
  - Domain Engineering
  - DSLs
- The Proposed Approach
  - Application-Based Domain Modeling (ADOM)
  - ADOM-Java: Applying ADOM to Java
- Evaluation
- Summary

- We teach our students many software engineering principles.
  - Low Coupling
  - High Cohesion
  - Encapsulation
  - Layering
  - …….
- Do they really understand and ready to implement these principles?
- Well, they do …..yet to a limited extent.

- In this work, we propose a framework that better guides students to implement SE principles.
- The framework relies on the notion of patterns, domain engineering, software product lines, and DSL.
- We apply the approach to programming language – Java.

Domain Engineering: Definition and Process

- "The purpose of **domain engineering** is to provide the reusable core assets that are exploited during **application engineering** when assembling or customizing individual applications." [Harsu, 2001]



[Ardis, 2000]

Domain specific languages (DSLs)

- DSLs provide **abstraction** over the domain.
- The domain semantics are handled by **code generators**.
- DSLs support a closed set of concepts allowing **validation** when specifying specific application.
- As a consequence, DSLs leads to improved **quality** and **productivity** [Kieburtz , 1996]

- Two type of DSLs
  - External
  - Internal

External DSLs: Pros and Cons

Pros

1. External DSLs enable abstraction.
2. External DSLs reuse expert knowledge.
3. External DSLs can validate the developer's code.

Conclusion: Productivity is high.

Cons

1. External DSLs are usually abandoned in the development process.
2. External DSLs introduce a significant change of the programming paradigm.
3. External DSLs are suitable for large scale product lines companies.
4. External DSLs limit the application developer's expressiveness.

Conclusion: Applicability is low.

[Freeman, 2006]

Internal DSLs: Pros and Cons

Pros

1. Internal DSLs require less efforts in the domain engineering activities.
2. Internal DSLs require a less radical change in the programming paradigm.
3. Internal DSLs keep the application developer's expressiveness.

Conclusion:
Applicability is high.

Cons

1. Internal DSLs lack in their abstraction capabilities.
2. Internal DSLs lack in their support for validation.

Conclusion:
Productivity not as high.

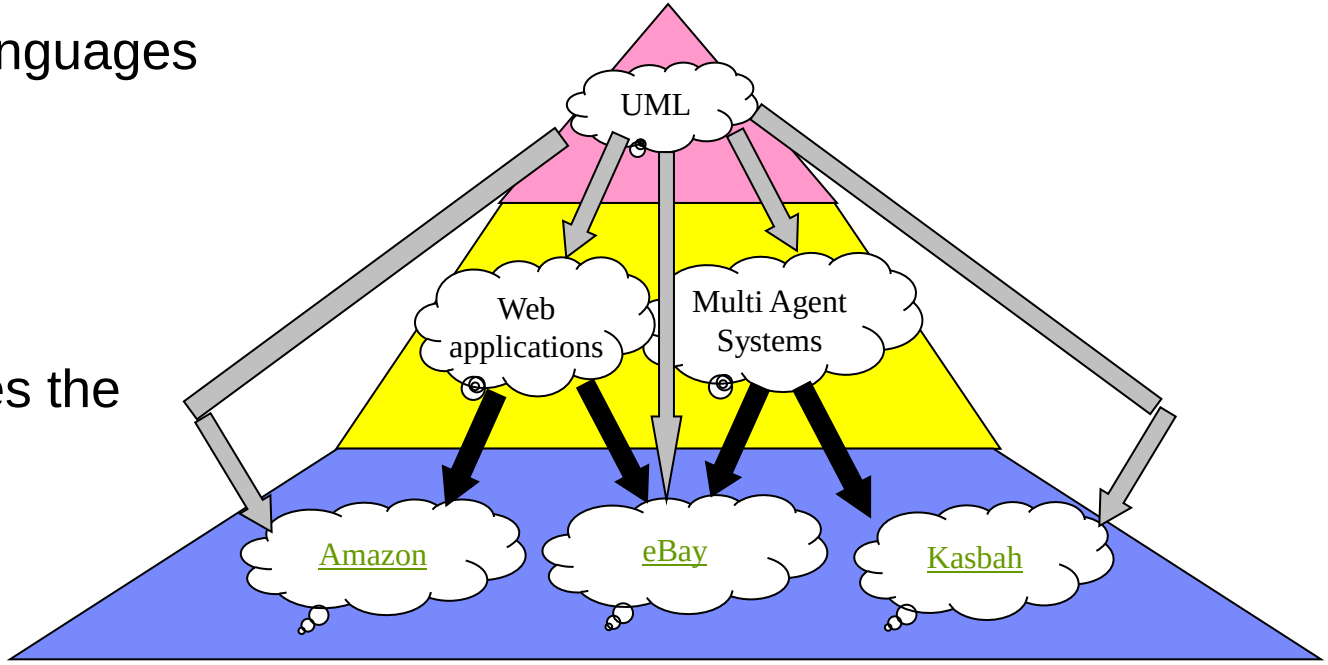
The underlying Framework: Application-based Domain Modeling (ADOM)

Language layer:

defines the used languages in both layers.

Domain layer:

guides and validates the application models.



Application layer:

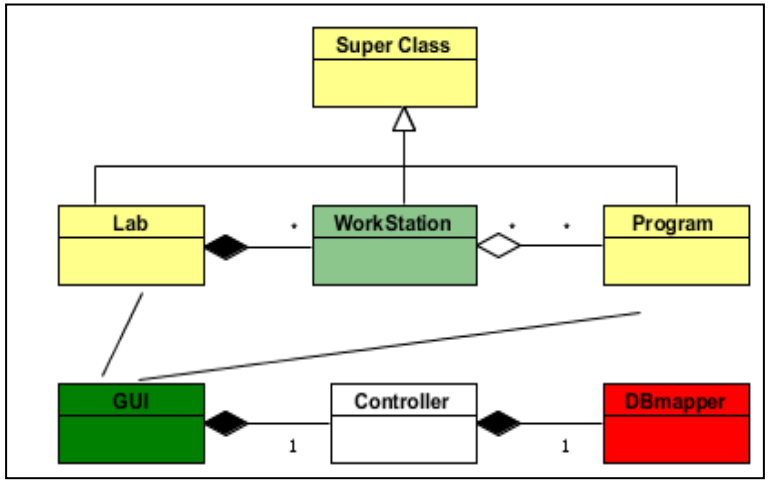
domain-specific applications.

[Reinhartz-Berger & Sturm (2004…)]

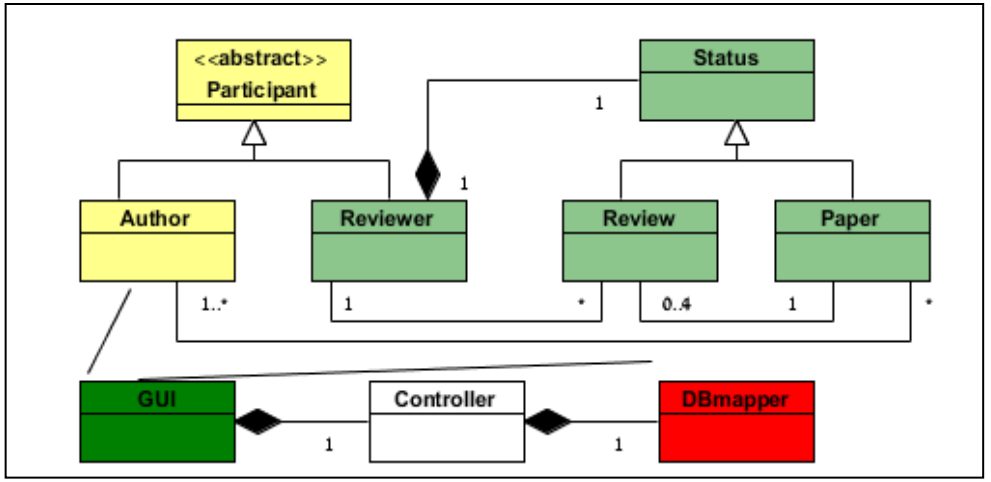
- In this work we use Java as the modeling language - this is called ADOM-Java.

ADOM-Java: Domain Abstraction

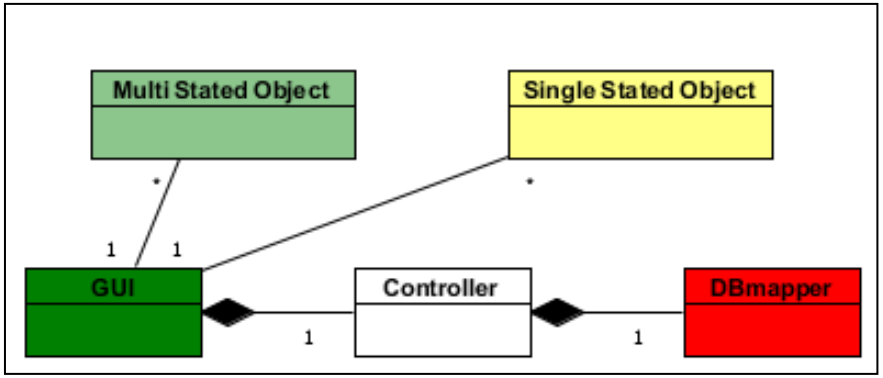
Labs System:



Conference System:



Domain:



ADOM-Java: Indicators and Classification via Java Annotation

Domain

```
@Indicators
public class Controller {

 DBmapper db;

 public boolean
  changeStateDomainObject()

 public boolean
  addDomainAssociation()

 public DomainObject
  addDomainObect()
```

Application

```
@Controller
Public class AppController {

 @db
  AppMapper db;

 @changeStateDomainObject
  public boolean fixWS()

 @addDomainAssociation
  public boolean assignProgtoWS()

 @addDomainObect
 public Lab addLab()
 @addDomainObect
 public Program addProgram()

 public boolean
 reportMalfuctionWS()
```

ADOM-Java: Indicators

Indicator	Constraint	Goes where?	Attributes
Multiplicity	Multiplicity	All ADOM-Java elements	Min – minimum number of allowed instances Max- maximum number of allowed instances
Final	Language	Class, Field and Method	*ModifierOptions Enum* which has three possible values: (TRUE, FALSE, ALL)
Static	Language	Field and Method	*ModifierOptions Enum* which has three possible values: (TRUE, FALSE, ALL)
Access	Language	Class, Constructor Field and Method	Array of *AccessOptions Enum* which has five possible values: (PRIVATE, PACKAGE, PROTECTED, PUBLIC, ALL)
Typing	Language	Field, Method and Parameter	Array of Java and ADOM-Java types
Generic Typing	Language	Field, Method and Parameter	Array of Java and ADOM-Java types
Statement	Language	Statement	Array of *StmtOptions* which have 17 possible values: 16 according to the different Java statement types and ALL.
Ordering	Ordering	-	-

ADOM-Java Constraints: Multiplicity and Language


```
//domain code:
@Multiplicity (min = 1, max = 1)
public class Controller {
```

(Controller must be public and non-final and single!)

```
 @Access({AccessOptions.PRIVATE, AccessOptions.PACKAGE})
 @Multiplicity (min = 1, max = 1)
 DBmapper db;
```

(db must be non static and non-final and single! May be either private or package)

```
 @Multiplicity (min = 1)
 @Typing({DomMultiStatedObj.class, DomSingleStatedObj.class})
 public DomSingleStatedObj addDomainObect()
```

(add Domain Object must be public, non-static, and non-final! may return either a DomMultiStatedObj or a DomSingleStatedObj) and must appear al least once!

```
}
```

ADOM-Java Constraints: Constraining Behavior via Statements


```
//domain code:
@Multiplicity (min = 1)
@Typing ({DomMultiStatedObj.class, DomSingleStatedObj.class })
public DomainMultiStatedObject addDomainObect(
 @Multiplicity (min = 1) String ObjectsData) {
 DomainMultiStatedObject dom = new
  DomainMultiStatedObject(ObjectsData);
 DBupdate: db.addDomainObject(dom);
 objectReturn: return dom;
}
//application code:
@addDomainObect_String_ObjectsData
public Lab addLab(@ObjectsData String id,@ObjectsData String maxWS) {
  @dom
  Lab l = new Lab(Integer.parseInt(id), Integer.parseInt(maxWS));
  @DBupdate
  db.addLab(l);
  @objectReturn
  return 1;     }
```

ADOM-Java Constraints: Multiplicity, Statements and Ordering


```
//domain code:
@Multiplicity (min = 1)
@Typing ({DomMultiStatedObj.class, DomSingleStatedObj.class })
public DomainMultiStatedObject addDomainObect(
@Multiplicity (min = 1) String ObjectsData) {

  @Multiplicity (min = 1, max = 1)
  DomainMultiStatedObject dom = new DomainMultiStatedObject(ObjectsData);

  @Multiplicity (min = 1, max =1 )
  DBupdate: db.addDomainObject(dom);

  @Multiplicity (min = 1, max =1)
  objectReturn: return dom;
}
```

dom must appear once and of expression type!

Dbupdate must appear once and of expression type!

objectReturn must appear once and of return type!

Statements order must be kept!!

ADOM-Java: Guidance and Instantiation (Structural)

Domain

```
@Multiplcity (min = 1)
public class Controller {

 @Multiplcity (min = 1)
 DBmapper db;

 @Multiplcity (min = 1)
 public boolean
  changeStateDomainObject()

@Multiplcity (min = 1)
 public boolean
  addDomainAssociation()

 @Multiplcity (min = 1)
 public DomainObject
  addDomainObect()
```

Application

```
@Controller
Public class AppController {

 @db
 AppMapper db;

@changeStateDomainObject
 public boolean fixWS()

@addDomainAssociation
 public boolean assignProgtoWS()

@addDomainObect
public Lab addLab()
@addDomainObect
public Program addProgram()

public boolean
reportMalfuctionWS()
```

ADOM-Java: Guidance and Instantiation (Behavioral)


```
//domain code:
@Multiplicity (min = 1)
@Typing ({DomMultiStatedObj.class, DomSingleStatedObj.class })
public DomainMultiStatedObject addDomainObect(
 @Multiplicity (min = 1) String ObjectsData) {
 DomainMultiStatedObject dom = new
  DomainMultiStatedObject(ObjectsData);
 DBupdate: db.addDomainObject(dom);
 objectReturn: return dom;
}
//application code:
@addDomainObect_String_ObjectsData
public AppDomMulti addDomainObect(String AppObjectData){
  @dom
  AppDomMultiStated app = new AppDomMultiStatedObj(AppObjectData );
  @DBupdate
  Appdb.addDomainObject(app);
  @objectReturn
  return app;}
```

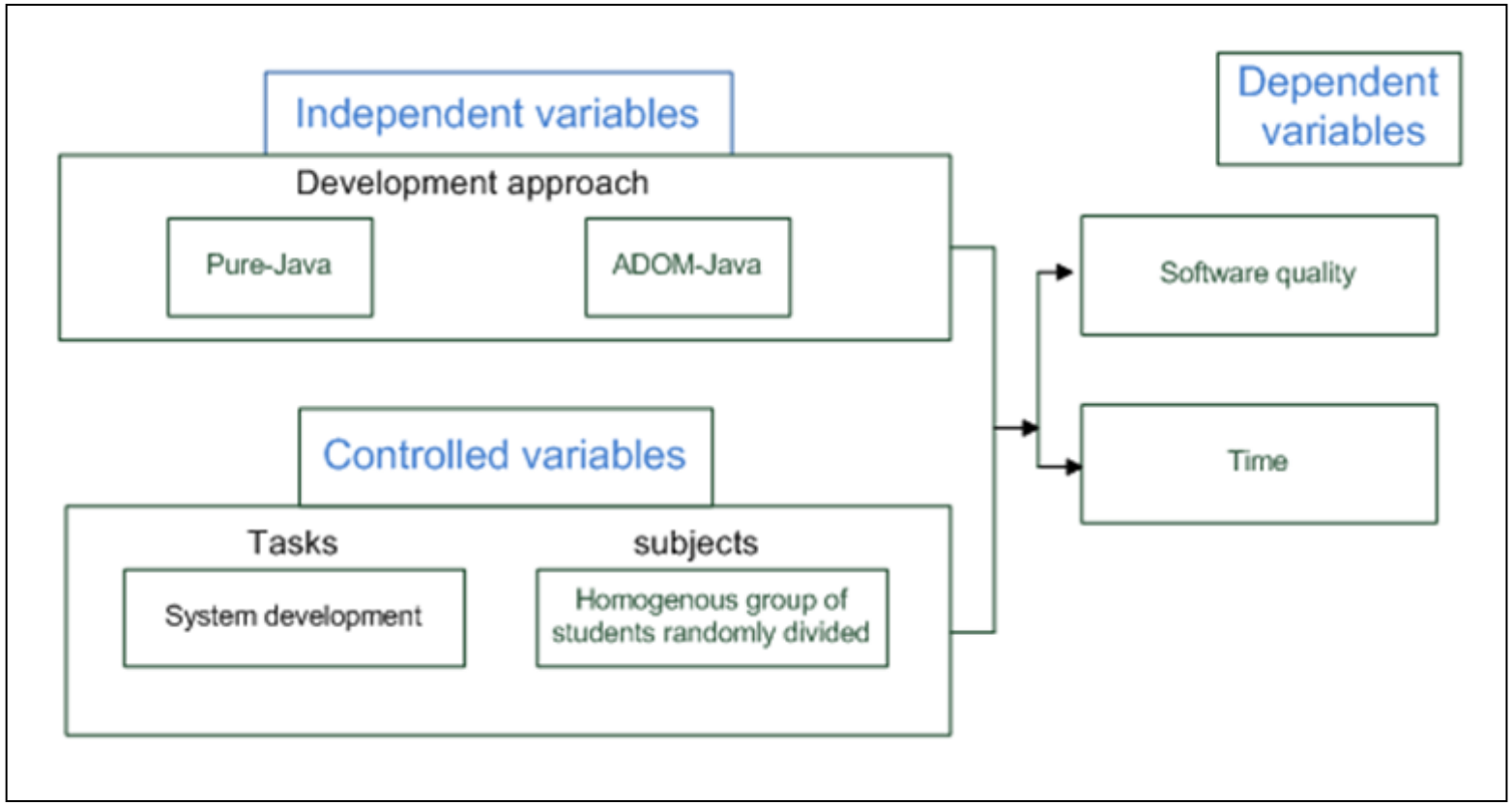
Goal and Objectives

To summarize, we propose a framework with the following capabilities from both DSL approaches:

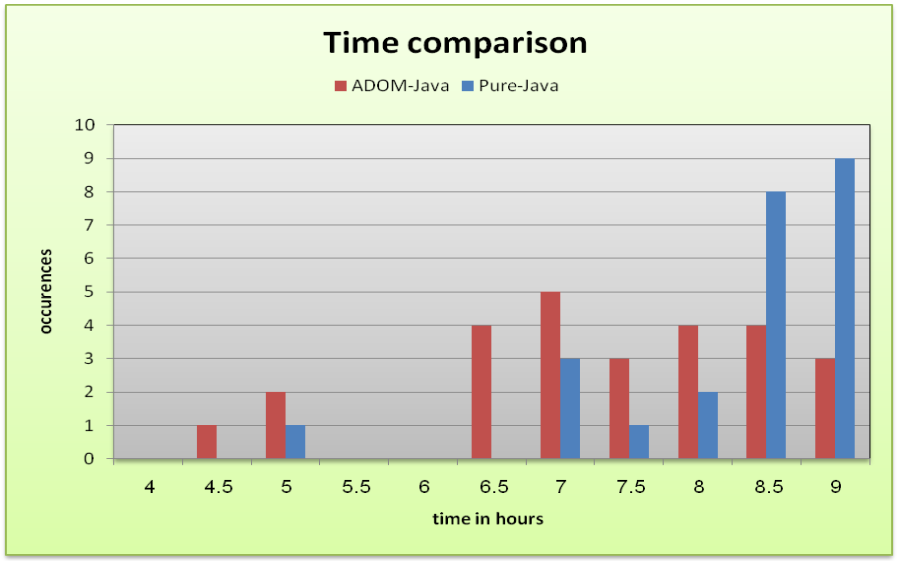
- Capturing and reusing domain experts' knowledge in the form of domain rules and constraints.
- Validating the application according to domain rules and constraints.
- Keep the developer expressiveness as the domain specification is embedded into a GPPL.

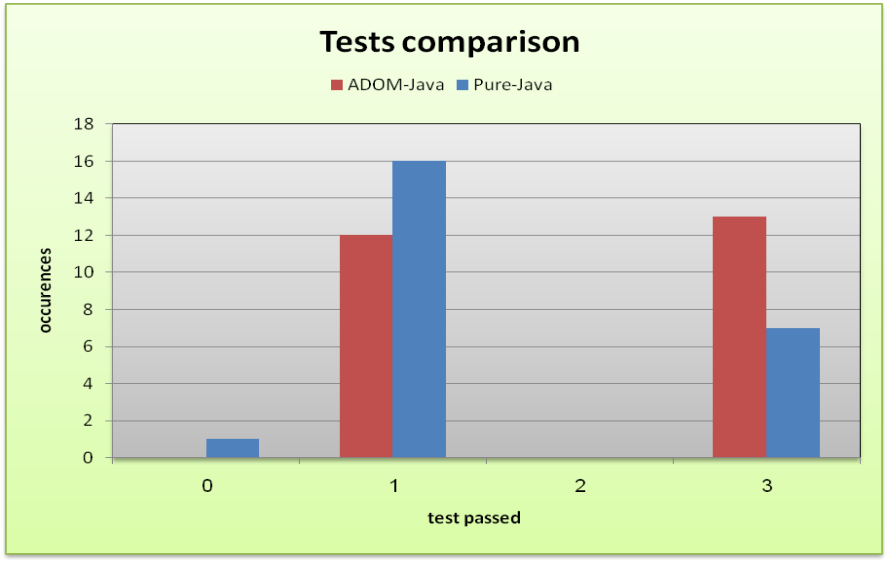
We expect that the proposed framework will increase the **productivity** and **quality** of the application code and still maintain **applicability** and **expressiveness.**

Quantitative Research: The Research Model



Quantitative Research: Results





	Development time (hours)	Functional quality (number of passed tests)
ADOM-Java	7.49	2
Pure-Java	8.43	1.54
Significance	p<0.001	p>0.05

Quantitative Research: Qualitative Research of the Design

- Data Collection
  - Various comments were assigned:
    - Over permissive access modifiers.
    - Deviation from the layers separation paradigm.
- Data Coding
  - The comments were coded according to the given domain model:
    - Layers separation.
    - Responsibility assignments.
    - General coding issues.
- Data Analysis

Problem Type	Pure-Java	ADOM-Java
Layered separation	8	0
Responsibility assignments	-	-
• Skinny objects	12	0
• Partial responsibility	4	5

Qualitative Research

- 20 students in 3 sessions.
- Data Collection
  - Several guiding questions.
  - Advantages and disadvantages questions.
- Data Coding
  - Answers were coded with 1-3 descriptive words.
- Data Analysis
  - Similar codes were abstracted into a common attribute.
- Findings
  - Validation perceived as complex, cumbersome and frustrating.
  - Positive comments were related to guidance and code design.
  - Negative comments were related to complexity, usability and a learning curve.

Evaluation Conclusions

- The guidance in ADOM-Java was significant.
- Quantitative results were in favor of ADOM-Java in both development time and passed tests.
- **Incorporated design principles were treated better in ADOM-Java.**
- Usability and learning curve were perceived as significant disadvantages.
- Improved results were achieved **despite of significant** disadvantages.

Summary

- The results indicate that formalizing and guiding students while implementing SE principles would allow them to better utilize such principles.

- Additional evaluation of the suggested approach and other alternatives is required.