Behavior-derived Reuse: Conceptual Foundations and Practical Tools for Increasing Software Reuse

Iris Reinhartz-Berger & Anna Zamansky
University of Haifa, Israel
Outline

Part A: Similarity and reuse - terminologies and background
- Software Reuse: clone-and-own and SPLE
- Similarity: clone types and variability mechanisms
- Motivation: the renting applications example

Part B: The behavior-derived reuse approach and the VarMeR tool
- The notion of behavior
- Behavior-derived similarity analysis
- The VarMeR tool

Part C: Discussion
- Possible applications & future research
- Questions & Answers
Part A: Similarity and reuse - terminology and background

- Software Reuse: clone-and-own and SPLE
- Similarity: clone types and variability mechanisms
- Motivation for behavior-derived reuse: the renting applications example
Software Reuse

- Software reuse - using existing software artifacts (such as requirements, design models, implementation/code, test cases, and so on) in order to produce new software.

- Software reuse has the potential to:
  - Increase productivity
  - Reduce costs and time-to-market
  - Improve software quality

- Two types of reuse are:
  - Ad-hoc: clone-and-own
  - Systematic: software product line engineering (SPLE)
Software Reuse: Clone-and-Own

- **Essence:** copying an existing artifact and adapting it to the requirements of the new software

- **Advantages:**
  - simple to apply
  - fast and immediate for addressing changes in requirements

- **Drawbacks:**
  - high maintenance costs
  - bug propagation
  - negative impact on design and understandability
  - strain on resources
Software Reuse: SPLE

**Essence:** managing artifacts at two levels
- Domain engineering - *core assets* management
- Application engineering - *product artifacts* creation

**Advantages:**
- Effective & efficient when developing similar software products
- Enable fast response to new opportunities and changing markets

**Drawbacks:**
- Profitability over time - high up-front investment in the development of core assets
- Increased complexity and intense negotiation
Software Reuse: ISO/IEC 26520 for SPLE
**Similarity as a key concept for Reuse**

- **Observations:**
  - High similarity decreases the amount and complexity of adaptation
  - Low similarity may complicate reuse

- **Applications:**
  - *Clone detection techniques* use similarity metrics (mainly syntactic and semantic) for identifying similar artifacts, or artifacts that originate from the same source
  - SPLE methods use similarity analysis techniques and *variability mechanisms* to extractively or reactively create product lines and support systematic reuse
# Similarity clone detection techniques, Rattan et.al (2013)

<table>
<thead>
<tr>
<th>Type of clones</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1 (exact clones)</td>
<td>Identical except for variations in <strong>white space and comments</strong></td>
</tr>
<tr>
<td>Type 2 (renamed/parameterized clones)</td>
<td>Structurally/syntactically similar except for changes in <strong>identifiers, literals, types, layout and comments</strong></td>
</tr>
<tr>
<td>Type 3 (near miss clones)</td>
<td>“Copies” with further modifications like <strong>statement insertions/deletions</strong> in addition to changes in identifiers, literals, types and layouts</td>
</tr>
<tr>
<td>Type 4 (semantic clones)</td>
<td><strong>Functionally similar</strong> without being textually similar</td>
</tr>
</tbody>
</table>

**Structural clones**

Patterns of **interrelated classes** emerging from design and analysis space at architecture level

**Function clones**

Limited to the granularity of a **function/method/procedure**

**Model based clones**

For **graphical languages** which replace the code as core artifacts for system development
Variability mechanisms are techniques used to encapsulate the variable parts and to provide appropriate support for creating product artifacts.

The asset developer has to decide what variability mechanisms to choose in order to increase potential reuse.

Several catalogs of variability mechanisms have been proposed:

- Jacobson et al. (1997)
- Gacek & Anastasopoulos (2001)
- Svahnberg et al. (2005)
- Bachmann & Clements (2005)
- Becker et al. (2007)
- Vom Brocke (2007)
**Similarity variability mechanism, Bachmann & Clements (2005)**

<table>
<thead>
<tr>
<th>Variability mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurators</td>
<td>assembling whole product assets by putting together pieces that are core assets</td>
</tr>
<tr>
<td>Parameters</td>
<td>keeping several small variation points for each variable feature</td>
</tr>
<tr>
<td>Inheritance</td>
<td>defining classes that are used in the product and inherited from generic classes defined for the product line</td>
</tr>
<tr>
<td>Component substitution</td>
<td>selecting from existing variants and inserting into core assets</td>
</tr>
<tr>
<td>Plug-ins</td>
<td>selecting and inserting at runtime</td>
</tr>
<tr>
<td>Templates</td>
<td>filling in product-specific parts in a generic body</td>
</tr>
<tr>
<td>Generators</td>
<td>producing components based on specifications</td>
</tr>
<tr>
<td>Aspects</td>
<td>selecting and inserting either at precompile or compile time</td>
</tr>
<tr>
<td>Runtime conditionals</td>
<td>specifying (at runtime) under which condition a core asset is included in a product</td>
</tr>
</tbody>
</table>
Polymorphism in OOP - refers to an ability to process objects differently depending on their data type or class.

Types of polymorphism:
- Parametric - similar behaviors
- Subtyping - refined or extended behaviors
- Overloading - different behaviors with similar interfaces
## Motivation: the renting applications example

<table>
<thead>
<tr>
<th></th>
<th>WeWork</th>
<th>RentCom</th>
<th>FindRoommate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renting objects</td>
<td>Offices</td>
<td>Houses</td>
<td>Rooms in a house</td>
</tr>
<tr>
<td>Renting subjects</td>
<td>Clients, an office can be rented to multiple clients</td>
<td>Clients, an house can be rented to multiple clients</td>
<td>Roommate, a room can be rented to a single client</td>
</tr>
<tr>
<td>Rental inclusion</td>
<td>Amenities</td>
<td>Amenities</td>
<td>Facilities which have statuses that are checked on return</td>
</tr>
<tr>
<td>Rental prices</td>
<td>Per month</td>
<td>Per year</td>
<td>Per week</td>
</tr>
<tr>
<td>Rental constraints</td>
<td>According to minimal and maximal numbers of employees</td>
<td>According to the number of beds</td>
<td>According to gender preference</td>
</tr>
<tr>
<td>Rental status</td>
<td>empty (not rented), partial (can be rented to more clients), or full</td>
<td>Implicit (satisfying rental constraints)</td>
<td>free, rented</td>
</tr>
</tbody>
</table>
WeWork

WeWork is a global network of workspaces where companies and people grow together. We transform buildings into dynamic environments for creativity, focus, and connection. More than just the best place to work, though, this is a movement toward humanizing work. We believe that CEOs can help each other, offices can use the comforts of home, and we can all look forward to Monday if we find real meaning in what we do.

What will it cost?

<table>
<thead>
<tr>
<th>Desks</th>
<th>Starting Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Desk</td>
<td>$1,050/mo</td>
</tr>
<tr>
<td>Hot Desk</td>
<td>$1,070/mo</td>
</tr>
</tbody>
</table>

Private Offices

<table>
<thead>
<tr>
<th>Seating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Seat</td>
</tr>
<tr>
<td>2 Seats</td>
</tr>
<tr>
<td>3 Seats</td>
</tr>
<tr>
<td>4 Seats</td>
</tr>
<tr>
<td>5 Seats</td>
</tr>
<tr>
<td>6 Seats</td>
</tr>
<tr>
<td>7 Seats</td>
</tr>
</tbody>
</table>

Building Amenities

- **Onsite Staff**: Our team is here for you throughout the workweek, from front-desk service to personalized support.
- **Cleaning Services**: Around the clock, our cleaning crew helps keep common areas, meeting rooms, and private offices looking their spiffiest.
- **Phone Booths**: These soundproofed alcoves provide comfortable sanctuaries for conducting private calls and video chats.
- **Conference Rooms**: Dedicated meeting spaces include A/V gear and unexpected details like custom wallpaper and marble tables.
- **Printing**: Every floor has its own space stocked with a business-class printer, office supplies, and paper shredder.
RentCom

Unknown
1972 Scotts Crossing Way, Annapolis, MD 21401
$1,600 2 Beds 1126 sqft (844) 356-7593

Floorplans
2 Bedrooms - starting at $1600

Amenities
- Laundry: Contact for details (844) 356-7593
- Parking: Contact for details (844) 356-7593
- Pets: Contact for details (844) 356-7593
- Features: Balcony, Patio, D... Fireplace High Speed Inter...
- Community: Disability Access
- Additional: Blinds, Cable Available, Intrusion Alarm

Property Details
Lovely, bright, 2B2B condo with fireplace and outdoor patio. Condo overlooks a greenbelt, is walking distance to the Westfield Mall, walking distance to hospital and located near to historical downtown Annapolis and marina. Location, location!!! Washer and dryer included. The unit has wheelchair access.
FindRoommate

Search location
London, OH 43140, USA

Range
25 miles

Members Who
Have a room

Max rent
$2,000+

Move-in date
October 13, 2018

Search

Search all members
Like what you see? Sign up today to become a member and see even more...

35 matches!

$650 HOUSE
Springfield, Ohio
- Updated -

Debra H
Straight Female, 49 years
Active 16 hours ago

$500 CONDO
Hilliard, OH
- Updated -

Kelly P
2 Occupants
Active 18 hours ago

$500 HOUSE
Grove City, OH

Frank M
2 Occupants
Active 9 hours ago

$600 CONDO
Hilliard, OH

Kevin S
2 Occupants
Active 1 day ago

SORT BY:
Last active
WeWork-inspired Class Diagram

```
// Office
-minEmployees: int
-maxEmployees: int
-area: double
-location: String
-pricePerMonth: double
-type: OfficeType
-status: OfficeStatus

+Office(int minE, int maxE, int area, double type, OfficeType location, String price, double amenities, ArrayList<Amenity> amenities)
+getPrice(): double
+setPrice(price): void
+getAmenities(): ArrayList<Amenity>
+setAmenities(amenities): void

// Amenity
-name: String
-description: String
+getAmenityName(): String
+getAmenityDescription(): String
+isAmenityValid(): boolean

// Clients
-name: String
-phone: String
+getName(): String
+getPhone(): String
```

```
<<enumeration>>
OfficeStatus

empty
partial
full

<<enumeration>>
OfficeType

shared
personal
hot
```
RentCom-inspired Class Diagram

```
House
- beds : int
- rooms : int
- area : double
- address : String
- pricePerYear : double
- petsAllowed : boolean
- rentalEndDate : Date
- type : House.Type

+ House(beds : int, rooms : int, area : double, type : House.Type, address : String, price : double, pets : boolean, amenities : ArrayList<Amenity>)
+ changePrice(newPrice : double) : void
+ addAmenity(newAmenity : Amenity) : void
+ removeAmenity(newAmenity : Amenity) : void
+ checkAvailability() : boolean
+ rentTo(c : Client) : boolean
+ endRental(c : Client) : boolean

House.Type
- apartment
- house
- condo
```

```
Amenity
- name : String
- description : String
+ setAmenity(name : String, desc : String) : void
+ getName() : String
+ setName(name : String) : void
+ setDescription(desc : String) : void
+ equals(a : Amenity) : boolean
```

```
Client
- name : String
- mail : String
+ setName(name : String, mail : String) : void
+ getName() : String
+ setName(name : String) : void
+ setMail(mail : String) : void
+ setPhoneNumber(phone : String) : void
```
FindRoommate-inspired Class Diagram

```
<<?xml version="1.0"?>
<model name="FindRoommate" version="1.0">
  <package name="Roommates" namespace=""/>
  <namespace name=""/>
  <element name="Roommate" type="entity">
    <attribute name="name" type="String"/>
    <attribute name="phone" type="String"/>
    <attribute name="gender" type="GenderConstraint"/>
    <attribute name="firstName" type="String"/>
    <attribute name="lastName" type="String"/>
    <attribute name="getPhone" type="String"/>
    <attribute name="setPhone" type="String"/>
    <attribute name="getGender" type="GenderConstraint"/>
    <attribute name="getfirstName" type="String"/>
    <attribute name="getlastName" type="String"/>
    <attribute name="getRooms" type="Room"/>
    <attribute name="setRoom" type="Room"/>
  </element>
  <element name="Room" type="entity">
    <attribute name="periodInMonths" type="int"/>
    <attribute name="movingDate" type="Date"/>
    <attribute name="pricePerWeek" type="double"/>
    <attribute name="gender" type="GenderConstraint"/>
    <attribute name="status" type="RoomStatus"/>
    <attribute name="facility" type="Facility"/>
    <attribute name="rooms" type="Room"/>
  </element>
  <element name="RoomStatus" type="<!DOCTYPE enum PUBLIC -0x00000000 -0x00000000>
    <value name="free"/>
    <value name="rented"/>
  </element>
  <element name="GenderConstraint" type="<!DOCTYPE enum PUBLIC -0x00000000 -0x00000000>
    <value name="male"/>
    <value name="female"/>
    <value name="unspecified"/>
  </element>
  <element name="Facility" type="<!DOCTYPE enum PUBLIC -0x00000000 -0x00000000>
    <value name="facilityName"/>
    <value name="description"/>
    <value name="status" type="FacilityStatus"/>
    <attribute name="facilityName" type="String"/>
    <attribute name="description" type="String"/>
    <attribute name="status" type="FacilityStatus"/>
  </element>
  <element name="FacilityStatus" type="<!DOCTYPE enum PUBLIC -0x00000000 -0x00000000>
    <value name="new"/>
    <value name="good"/>
    <value name="bad"/>
    <value name="unvisited"/>
  </element>
  <element name="House" type="<!DOCTYPE enum PUBLIC -0x00000000 -0x00000000">
    <attribute name="houseName" type="String"/>
    <attribute name="type" type="HouseType"/>
    <attribute name="address" type="String"/>
    <attribute name="petsAllowed" type="boolean"/>
    <attribute name="rooms" type="Room"/>
    <attribute name="facilities" type="Facility"/>
    <attribute name="addFacility" type="Facility" type="void"/>
    <attribute name="removeFacility" type="Facility" type="void"/>
    <attribute name="checkAvailability" type="boolean"/>
  </element>
  <element name="HouseType" type="<!DOCTYPE enum PUBLIC -0x00000000 -0x00000000>
    <value name="apartment"/>
    <value name="house"/>
    <value name="condo"/>
  </element>
</model>
```
Motivation BUT4Reuse - Bottom-Up Technologies for Reuse

- Provides a unified framework for mining software artefact variants (Martinez et al., 2015)
  - An eclipse plug-in
- Supports different artifact types
  - Java, C, EMF Models, Textual files, File structures, JSON and CSV files, ...
- For C and Java source code, similarity is based on:
  - Feature Structure Tree (FST) positions
  - Names comparison
- Site: [https://github.com/but4reuse/but4reuse/wiki](https://github.com/but4reuse/but4reuse/wiki)
Motivation

BUT4Reuse - Bottom-Up Technologies for Reuse

- House & House Type
- Amenity & Client
Motivation BUT4Reuse - Bottom-Up Technologies for Reuse

- What about Client & Roommate?
- What about Amenity & Facility?

What about Room, House & Office?
- They are all rented and returned
- They all require check availability
- They all handle amenities/facilities
- They are rented to clients/roommates
- They have common attributes, such as area and price
Part B: The behavior-derived reuse approach and the VarMeR tool

- The notion of behavior
- Behavior-derived similarity analysis
- The VarMeR tool
The Notion of Behavior

- (Software) Systems may differ in their implementation and yet provide similar functionality.
  - Behavior refers to the (intended) functionality of the system
- A **behavior** is a transformation from an initial state to a final state due to some external event. It is represented as a triplet \((S_1, e, S^*)\), where:
  - \(S_1\) is the *initial state* of the system *before* the behavior occurs
  - \(e\) is an *external event* that *triggers* the behavior
  - \(S^*\) is the *final state* of the system *after* the behavior occurs
Behavior-derived Similarity Analysis

- Extract Behaviors
- Compare Behaviors
- Analyze Variability
- Reuse Recommendations

Ontological foundation
Products' representations
Variability mechanisms
Similarity measures
Similar elements

Object-oriented code in Java
Product Representation

- **Behavior descriptors**
  - *Shallow descriptor* - represents the behavior’s interface
    - Shallow.parameters = {(parameter, type)}
    - Shallow.returned = {(operationName, returnedType)}
  - *Deep descriptor* - represents the transformation the behavior performs on state variables
    - Deep.attUsed = {(att, type) | att is an *attribute* used (read) in the operation}
    - Deep.attModified = {(att, type) | att is an *attribute* modified (written) in the operation}
Product Representation

- A product is represented as a set of behaviors, such that for each behavior:
  - $S_1 = \text{Shallow.parameters} \cup \text{Deep.attUsed}$
  - $S^* = \text{Shallow.returned} \cup \text{Deep.attModified}$
  (currently $e = \text{operationName}$)
Example of Product Representation

- WeWork = (Office.CheckAvailability, Office.Rent, Office.EndRent, Office.AddAmenity, Office.RemoveAmenity, ...)

- Rent behavior of office:
  - Shallow.parameters
  - Shallow.returned
  - Deep.attUsed
  - Deep.attModified

```java
public boolean checkAvailability() {
    return (status != OfficeStatus.full) &&
            (clients.size() < maxEmployees);
}

public void setStatus (OfficeStatus newStatus) {
    this.status = newStatus;
}
```

```java
public boolean rent(Client c) {
    if (checkAvailability()) {
        clients.add(c);
        if (clients.size() >= minEmployees) {
            setStatus(OfficeStatus.partial);
        } else {
            setStatus(OfficeStatus.full);
        }
    }
    return true;
}
```
Example of Product Representation

- WeWork = (Office.CheckAvailability, Office.Rent, Office.EndRent, Office.AddAmenity, Office.RemoveAmenity, ...)

- Rent behavior of office:
  - Shallow.parameters: (c, Client)
  - Shallow.returned: (rent, Boolean)
  - Deep.attUsed: (clients, ArrayList), (minEmployees, int), (maxEmployees, int), (status, OfficeStatus), (partial, OfficeStatus), (full, OfficeStatus)
  - Deep.attModified: (clients, ArrayList), (status, OfficeStatus)
Behavior-derived Similarity Analysis

Extract Behaviors

Products' representations

Compare Behaviors

Similarity measures

Analyze Variability

Variability mechanisms

Reuse Recommendations

Ontological foundation

Syntactic, semantic, schematic, etc.
Similarity Measures

- Different Similarity measures can be used, e.g., semantic similarity
- Semantic (text) similarity measures are commonly classified as
  - **Corpus-based measures** identify the degree of similarity based on information derived from large corpora
  - **Knowledge-based measures** use information drawn from semantic networks
# Example of Similarity Calculation

<table>
<thead>
<tr>
<th>Shallow</th>
<th>parameters</th>
<th>{(c, WeWork.Client)}</th>
<th>{(c, RentCom.Client)}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>returned</td>
<td>{(rent, java.lang.Boolean)}</td>
<td>{(rent, java.lang.Boolean)}</td>
</tr>
<tr>
<td>Deep</td>
<td>attUsed</td>
<td>{(clients, java.util.ArrayList); (minEmployees, java.lang.Integer); (maxEmployees, java.lang.Integer); (status, WeWork.OfficeStatus); (partial, WeWork.OfficeStatus); (full, WeWork.OfficeStatus)}</td>
<td>{(clients, java.util.ArrayList); (beds, java.lang.Integer)}</td>
</tr>
<tr>
<td></td>
<td>attModified</td>
<td>{(clients, java.util.ArrayList); (status, WeWork.OfficeStatus)}</td>
<td>{(clients, java.util.ArrayList)}</td>
</tr>
</tbody>
</table>

Possible types of mappings
Behavior-derived Similarity Analysis

1. Extract Behaviors
2. Compare Behaviors
3. Analyze Variability
   - Similarity measures
   - Similar elements
   - Variability mechanisms

Polymorphism types: parametric, subtyping, overloading

Reuse Recommendations
## Similarity of Deep and Shallow behaviors

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Description</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE</td>
<td>covered and single-mapped</td>
<td><img src="image1.png" alt="Visualization" /></td>
</tr>
<tr>
<td>REFINEMENT</td>
<td>multi-mapped</td>
<td><img src="image2.png" alt="Visualization" /></td>
</tr>
<tr>
<td>EXTENSION</td>
<td>not covered</td>
<td><img src="image3.png" alt="Visualization" /></td>
</tr>
</tbody>
</table>
## Recommended Polymorphism-Inspired Mechanisms

<table>
<thead>
<tr>
<th>Shallow</th>
<th>Deep</th>
<th>Description</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE</td>
<td>USE</td>
<td>Both interfaces and transformations are similar</td>
<td>Parametric</td>
</tr>
<tr>
<td>USE</td>
<td>REF</td>
<td>Interfaces are similar and transformations are refined</td>
<td>Subtyping</td>
</tr>
<tr>
<td>USE</td>
<td>EXT</td>
<td>Interfaces are similar and transformations are extended</td>
<td>Subtyping</td>
</tr>
<tr>
<td>USE</td>
<td>REF-EXT</td>
<td>Interfaces are similar and transformations are both refined and extended</td>
<td>Subtyping</td>
</tr>
<tr>
<td>USE</td>
<td>NONE</td>
<td>Interfaces are similar and transformations are different</td>
<td>Overloading</td>
</tr>
</tbody>
</table>
The VarMeR Tool

https://sites.google.com/is.haifa.ac.il/varmer/
Part C: Discussion

- Possible applications & future research
- Questions & Answers
Product line ability decision support

Question: How to assess the ability of a set of products to form a product line?

‘Product line ability’ - the ability of a set of products to form a product line (Berger et al. 2014)

- Bottom-up: constructing a core asset out of existing product components
- Top-down: adapting existing products and creating new ones based on the generated core assets
**Product line ability decision support**

Suggested metrics assume high similarity of representations, mainly implementations or architecture models (Berger et al. 2014)

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of commonality (SoC)</td>
<td>$\text{SoC} =</td>
<td>\bigcap_{i=1..n} C_{p_i,r}</td>
</tr>
<tr>
<td>Product related reusability (PrR)</td>
<td>$\text{PrR}_i = \frac{\text{SoC}}{</td>
<td>C_{p_i,r} \cup C_{p_i,o}</td>
</tr>
</tbody>
</table>
Product line ability decision support

- VarMeR suggests more robust product line-ability analysis, which:
  - takes into account the behaviors of artifacts, rather than solely their implementations
  - allows for a more refined evaluation of the reuse effort, which reflects the possibilities to adopt specific reuse practices
Product line ability decision support

1. Behavior similarity calculation
2. Similarity degree measurement
3. Product-related variability degree measurement

Set of products
Similarity mapping
Minimal number of products*
Selected core assets**

Potential core assets
Product-related variability degree

More products? yes
Exclude products? yes

Legend:
Step
Object
Data flow
Control flow
Decision?

* For defining a core asset
** A subset of the potential core assets obtained in step 2
Product line ability decision support

Set of products

Similarity mapping

Behavior similarity calculation

Similarity graph

* Node’s Color represents the product
An *m-colored parametric asset* is a subgraph of a similarity graph representing at least m products (colors) where each two nodes are connected with a parametric edge.

An *m-color behavioral similarity degree* measures how “close” a given similarity m-colored sub-graph is to being an m-colored parametric asset.
Product line ability decision support

3-color behavioral similarity degree
\((1, 0, 0)\)

2-color behavioral similarity degree
\((0.33, 0.67)\)

Merge 3

Quick 2

Quick 1

Quick 3

Optimized Quick 3

Merge 1

Merge 2

Overloading

Parametric

Subtyping
Product line ability decision support

An \textit{m-color product-related variability degree} measures the difference between each product and the potential m-color core assets, as captured by the m-color behavioral similarity degree.

Intuitively, greater coverage of vertices indicates higher product line-ability.
Product line ability decision support

2-color product-related variability degree with respect to \( \{G_1, G_2\} \)

\[
\begin{align*}
(0, 0, 0, 0.67) & \\
(1, 0, 0, 0) & \\
(0.33, 0, 0, 0.67) & 
\end{align*}
\]
A proactive reuse framework
A proactive reuse framework

- Challenges:
  - Comparison across projects
  - Comparison across (artifact) types
  - Relevant recommendations at early stages of development
  - Querying and searching the core assets repository
  - (Semi-) automatic application of recommendations
  - Easy integration into the workflow of developers
Additional future research directions

▶ Evaluation
  ▶ With students for improving reuse educating and training capabilities
  ▶ With practitioners for improving software design and development skills

▶ Extension of the approach
  ▶ To additional variability mechanisms: parameterization, configuration, analogy, and others
  ▶ To support the application of recommendations in both directions
    ▶ Bottom-up to create core assets
    ▶ Top-down to generate and customize product artifacts
Additional feedback (questions, comments, suggestions for collaboration, etc.) can be directed to:

Iris Reinhartz-Berger  iris@is.haifa.ac.il
Anna Zamansky  annazam@is.haifa.ac.il
Own References


Additional References mentioned in the presentation

- **Clone detection:**

- **Variability mechanisms:**
Additional References mentioned in the presentation

- Variability mechanisms (cont.):
  - But4Reuse