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)

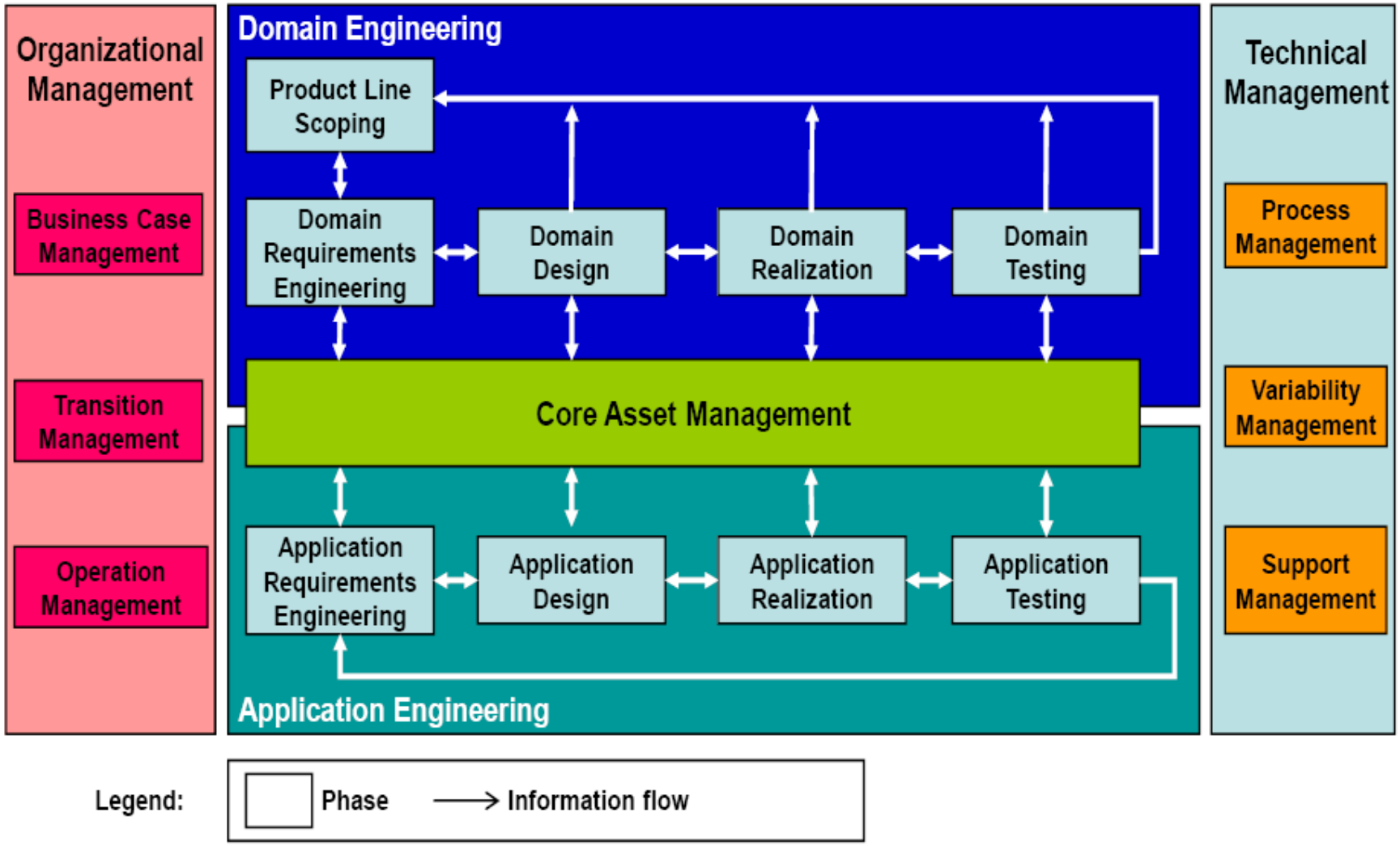
Software Reuse: Clone-and-Own

- **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)

Type of clones	Description
Type 1 (exact clones)	Identical except for variations in ***white space and comments***
Type 2 (renamed/ parameterized clones)	Structurally/syntactically similar except for changes in ***identifiers***, ***literals***, ***types***, ***layout and comments***
Type 3 (near miss clones)	"Copies" with further modifications like ***statement insertions/deletions*** in addition to changes in identifiers, literals, types and layouts
Type 4 (semantic clones)	***Functionally similar*** without being textually similar
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

Similarity variability mechanism, Bachmann & Clements (2005)

- ***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 & Anastasopoules (2001)
  - Muthig & Patzke (2002)
  - Svahnberg et al. (2005)
  - Bachmann & Clements (2005)
  - Becker et al. (2007)
  - Vom Brocke (2007)

Similarity variability mechanism, Bachmann & Clements (2005)

Variability mechanism	Description
Configurators	assembling whole product assets by putting together pieces that are core assets
Parameters	keeping several small variation points for each variable feature
Inheritance	defining classes that are used in the product and inherited from generic classes defined for the product line
Component substitution	selecting from existing variants and inserting into core assets
Plug-ins	selecting and inserting at runtime
Templates	filling in product-specific parts in a generic body
Generators	producing components based on specifications
Aspects	selecting and inserting either at precompile or compile time
Runtime conditionals	specifying (at runtime) under which condition a core asset is included in a product

Similarity polymorphism-inspired variability mechanism

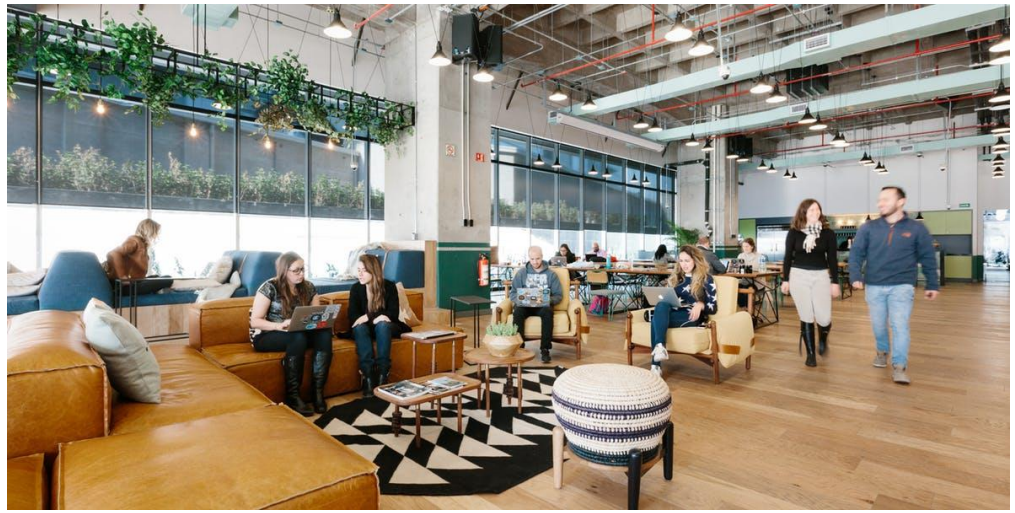
- Polymorphism in OOP - refers to an ability to process objects differently depending on their data type or class.
- Types of polymorphism:
  - Parametric - similar behaviors
  - Subtyping - refined or extended behaviors
  - Overloading - different behaviors with similar interfaces

Motivation the renting applications example

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

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?

Desks	Starting Price
Dedicated Desk per person	₪1,050/mo
Hot Desk per person	₪790/mo Sign Up Now

Private Offices
1 Seat
2 Seats
3 Seats
4 Seats
5 Seats
6 Seats
7 Seats

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

rent.com

My Saved Properties Search Moving Center

‹ Back to search

Home › Maryland › Annapolis Houses

Unknown

1972 Scotts Crossing Way, Annapolis, MD 21401 Map

$1,600 • 2 Beds • 1126 sqft • **(844) 356-7593**

Floorplans | Amenities | Details | Location

Floorplans

back to the top

2 Bedrooms - starting at $1600

NO IMG	House	$1600 /mo	2 beds / 2 baths	1126 sqft	Contact

Prices, specials, features and availability subject to change.

Amenities

back to the top

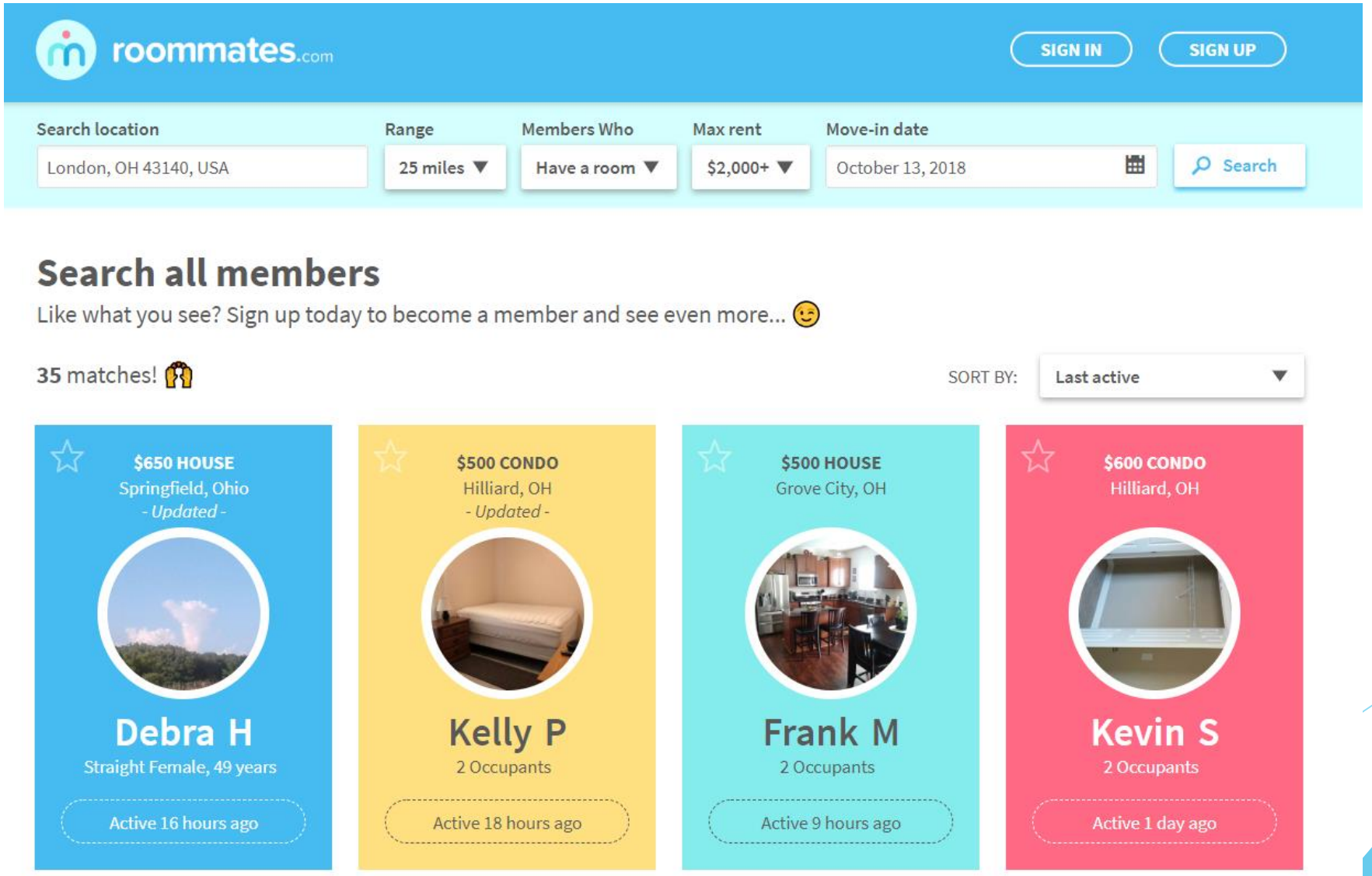
Laundry:	Contact for details (844) 356-7593
Parking:	Contact for details (844) 356-7593
Pets:	Contact for details (844) 356-7593
Features:	Balcony, Patio, D... Fireplace High Speed Inter...
Community:	Disability Access
Additional:	Blinds, Cable Available, Intrusion Alarm

Property Details

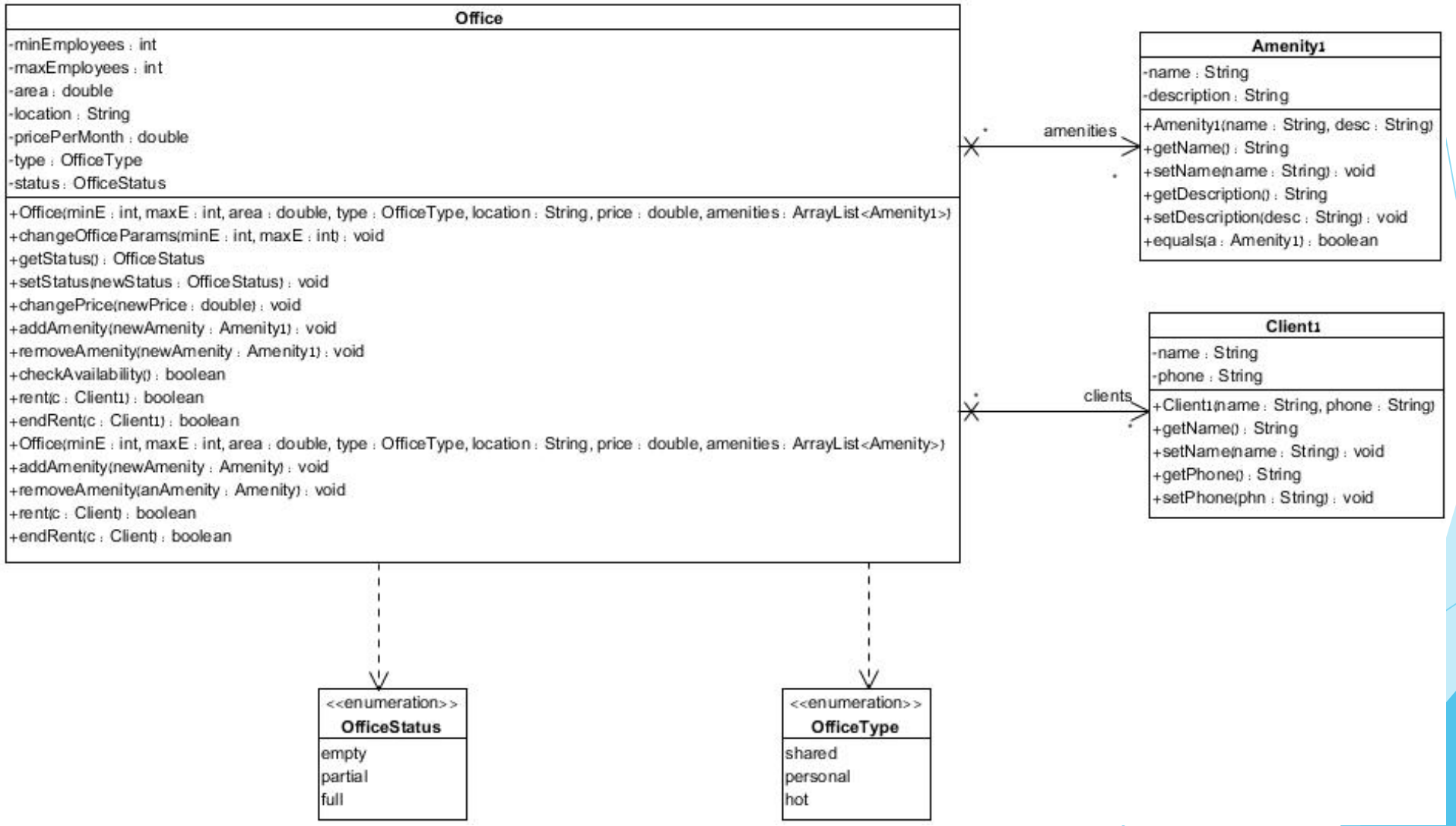
back to the top

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.

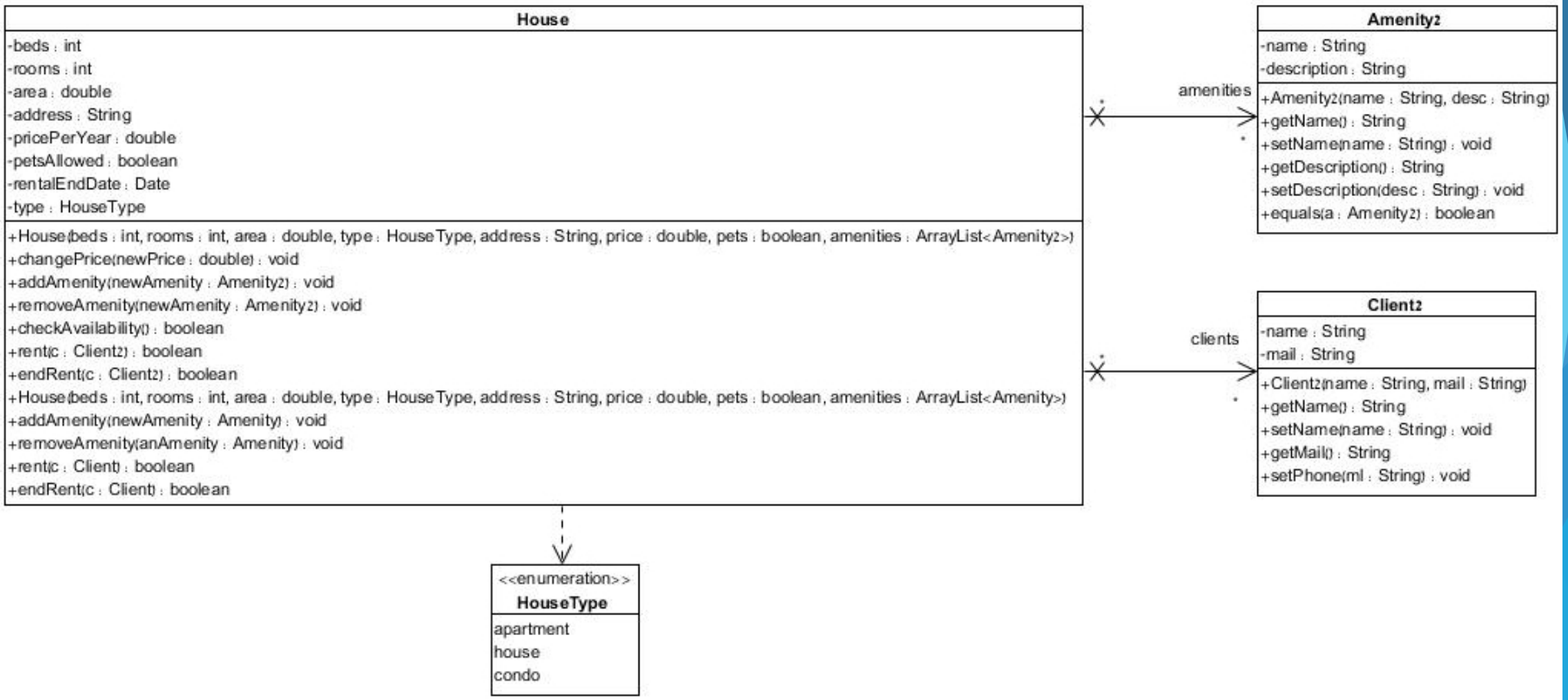
FindRoommate



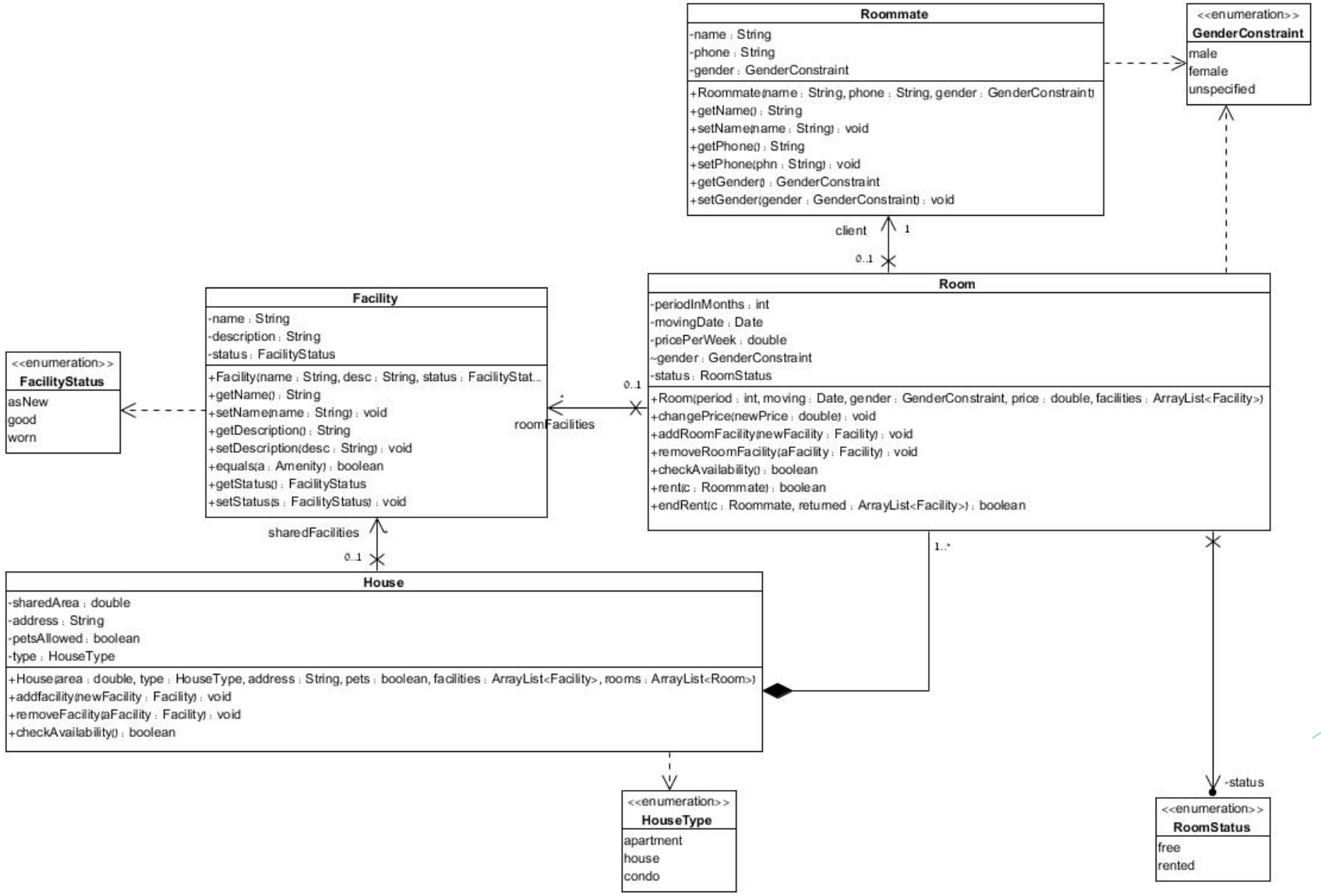
WeWork-inspired Class Diagram



RentCom-inspired Class Diagram



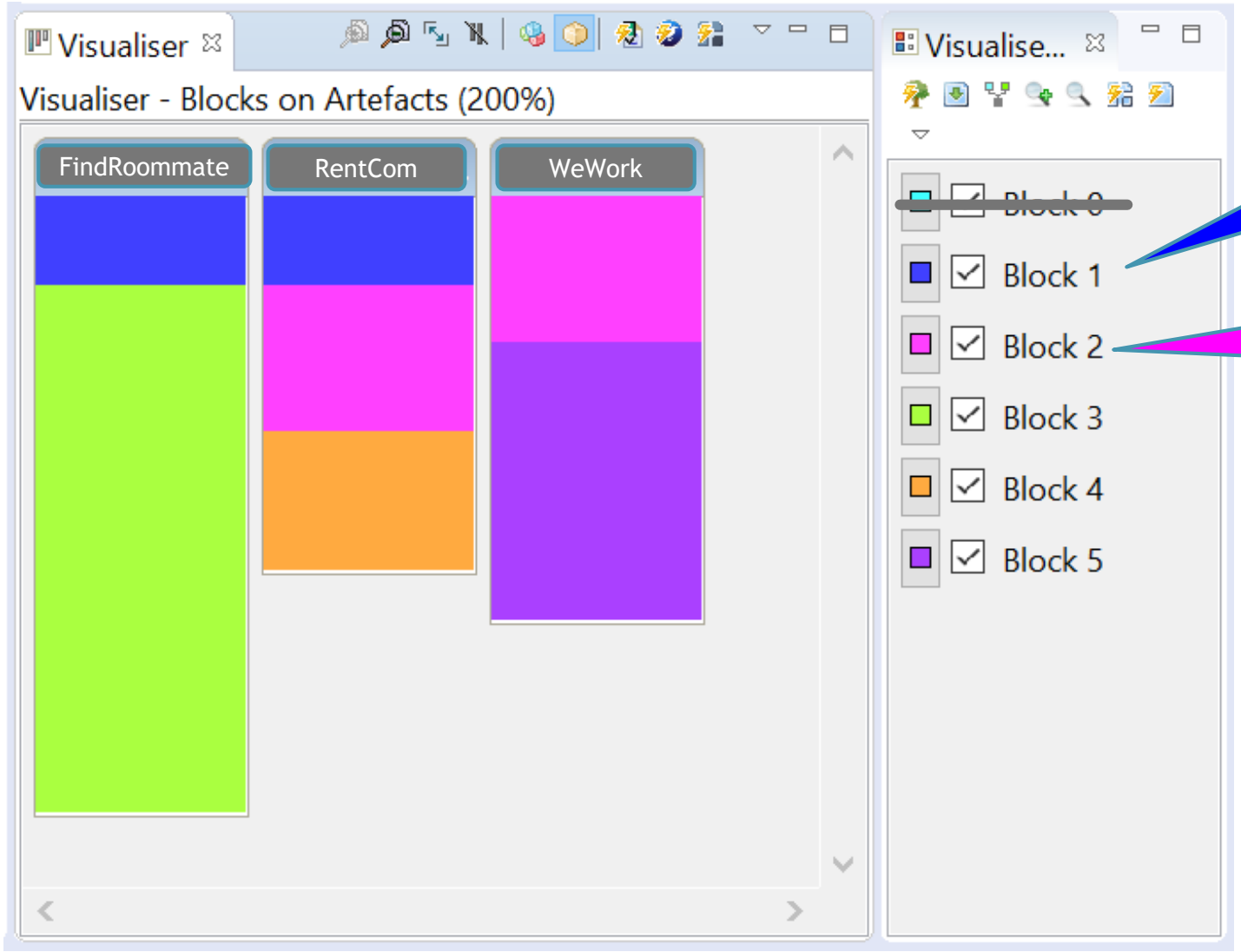
FindRommate-inspired Class Diagram



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

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

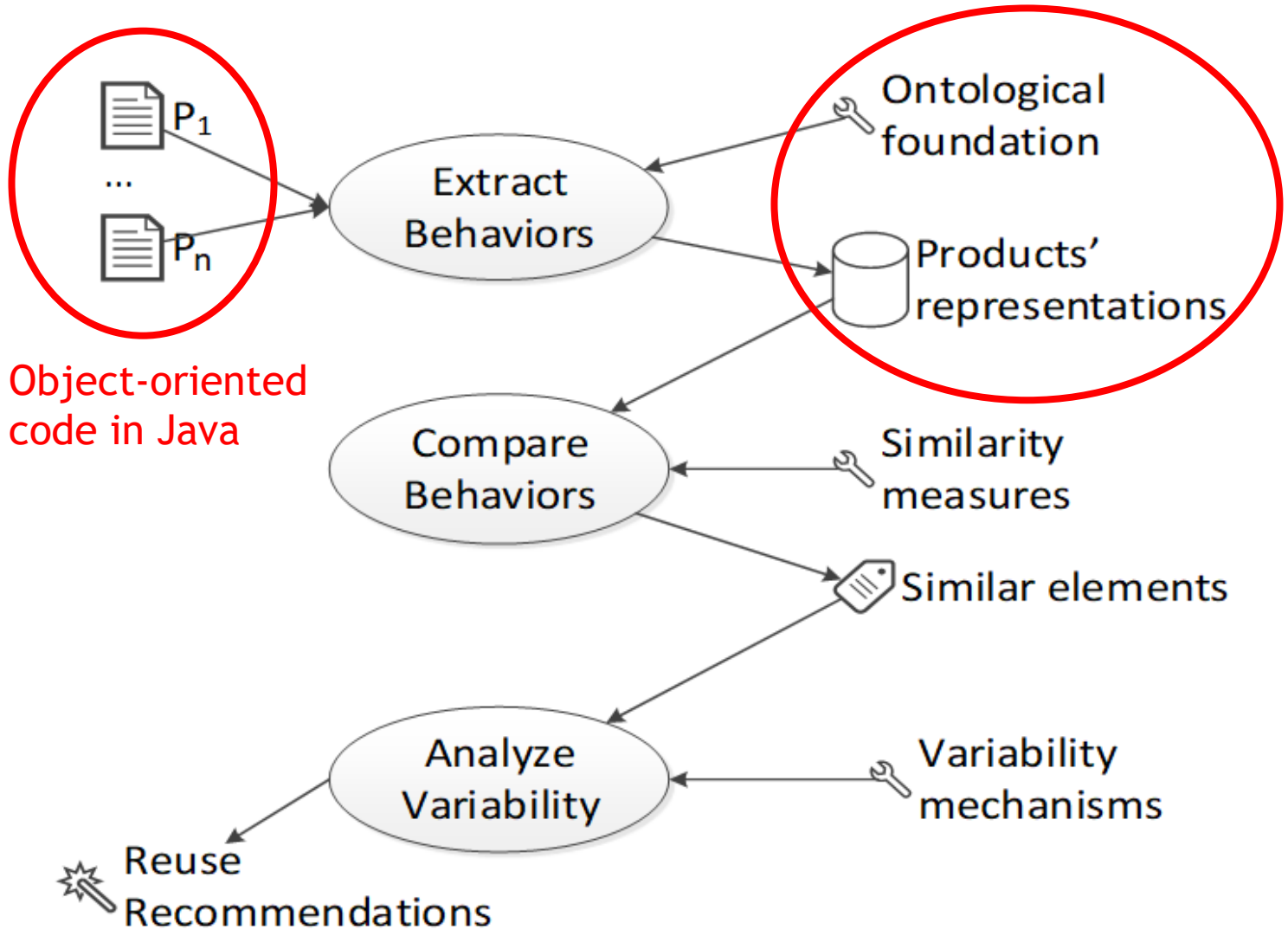
Part B: The behavior-derived reuse approach and the VarMeR tool

- ✓ 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



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
  - S1 = Shallow.parameters ∪ Deep.attUsed
  - S* = Shallow.returned ∪ Deep.attModified

  (currently e = 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

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

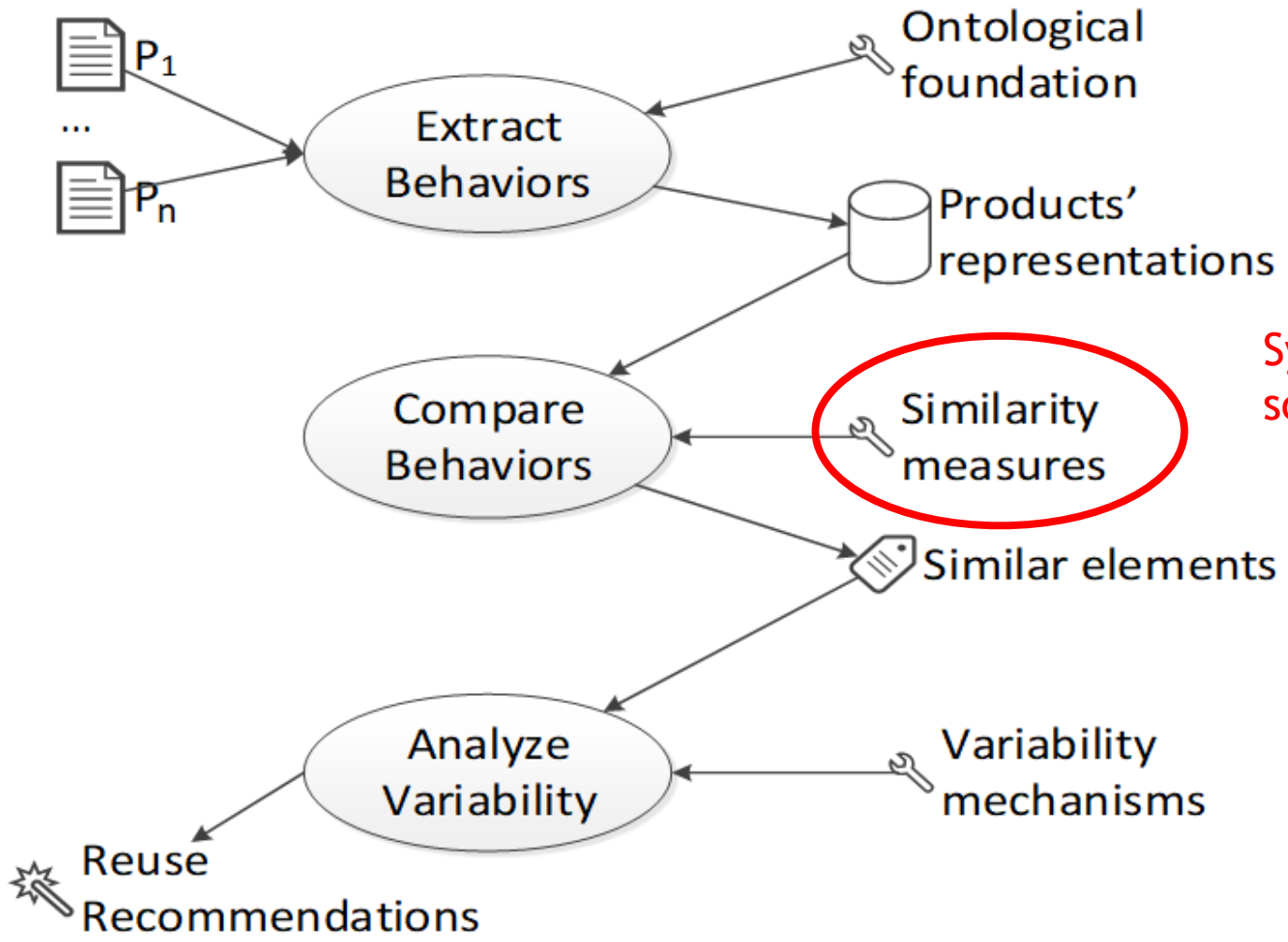
```
public boolean rent(Client c) {
     if (checkAvailability()) {
          clients.add(c);
          if (clients.size() >= minEmployees) {
               setStatus(OfficeStatus.partial);
          }
          if (clients.size() == maxEmployees) {
               setStatus(OfficeStatus.full);
          }
          return true;
     }
     else {
          return false;
     }
}
```

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

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

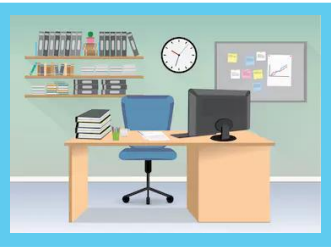


Syntactic, semantic, schematic, etc.

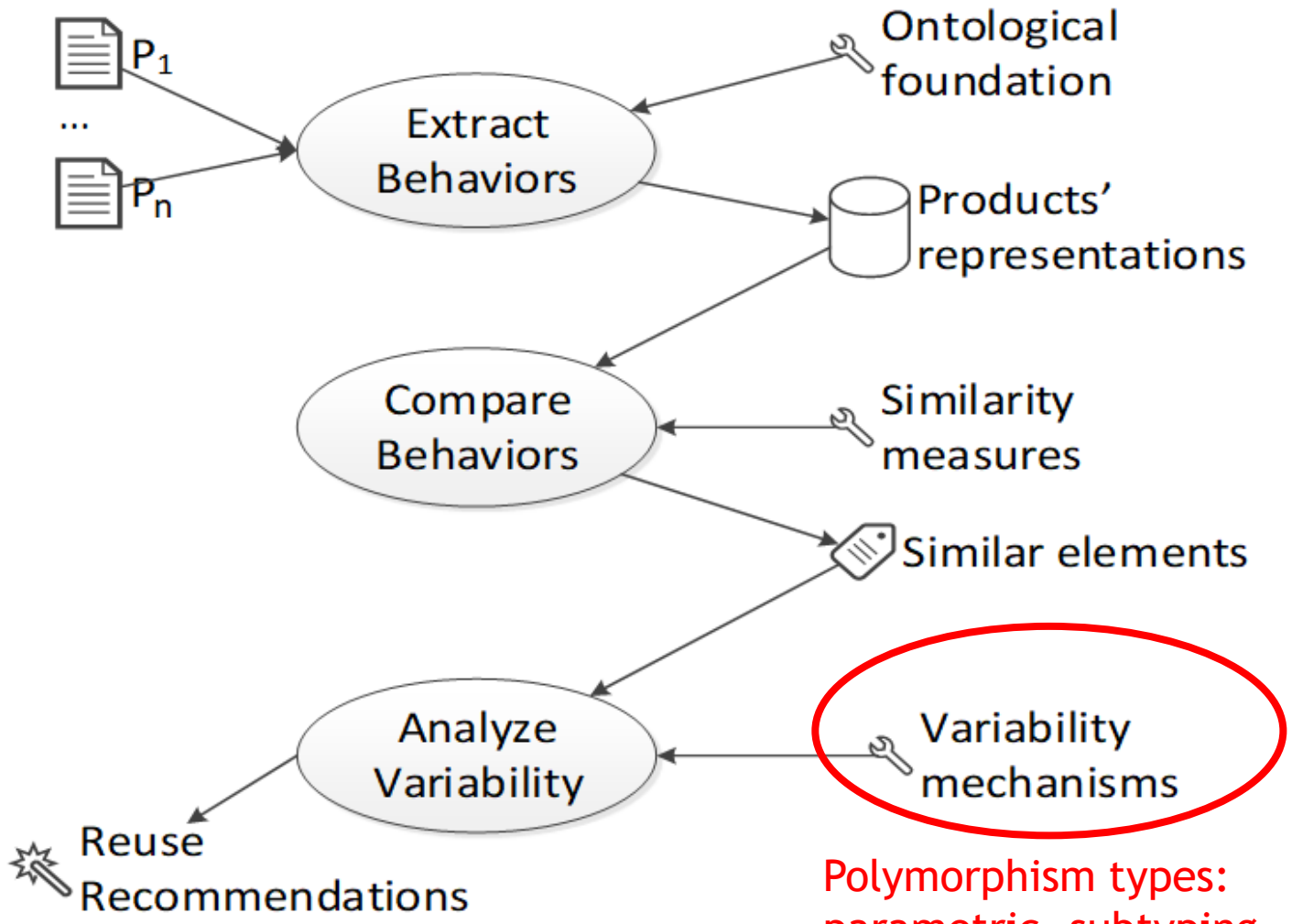
Similarity Measures

- 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

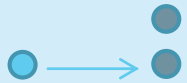
			Possible types of mappings	
Shallow	parameters	{(c, WeWork.Client)}	⟷	{(c, RentCom.Client)}
	returned	{(rent, java.lang.Boolean)}	⟷	{(rent, java.lang.Boolean)}
Deep	attUsed	{(clients, java.util.ArrayList); (minEmployees, java.lang.Integer); (maxEmployees, java.lang.Integer); (status, WeWork.OfficeStatus)} (partial, WeWork.OfficeStatus)} (full, WeWork.OfficeStatus)}	⟷	{(clients, java.util.ArrayList); (beds, java.lang.Integer}
	attModified	{(clients, java.util.ArrayList); (status, WeWork.OfficeStatus)}	⟷	{(clients, java.util.ArrayList)}

Behavior-derived Similarity Analysis



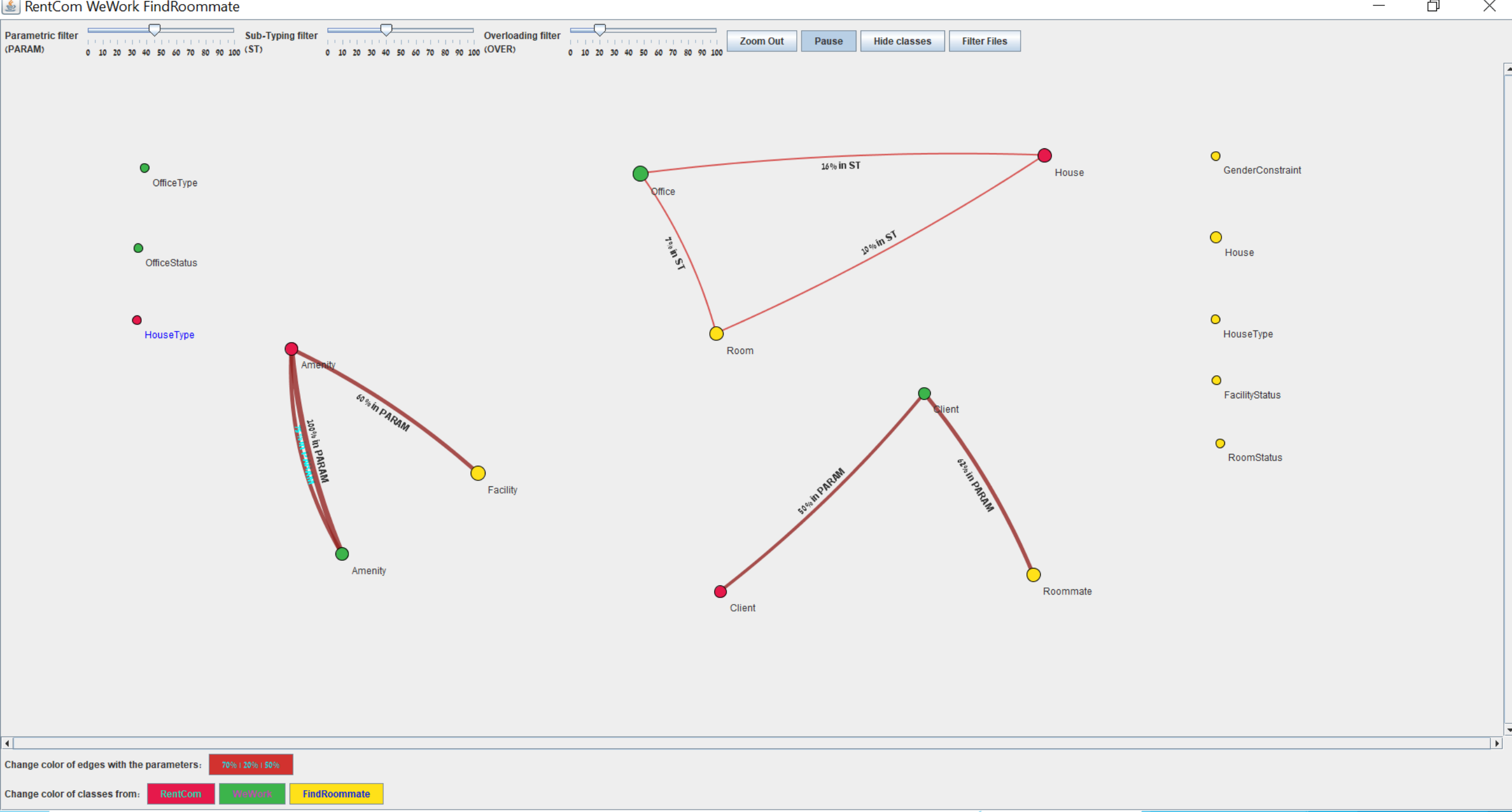
Polymorphism types: parametric, subtyping, overloading

Similarity of Deep and Shallow behaviors

Mapping Type	Description	Visualization
USE	covered and single-mapped	
REFINEMENT	multi-mapped	
EXTENSION	not covered	

Recommended Polymorphism-Inspired Mechanisms

Shallow	Deep	Description	Recommendation
USE	USE	Both interfaces and transformations are similar	Parametric
USE	REF	Interfaces are similar and transformations are refined	Subtyping
USE	EXT	Interfaces are similar and transformations are extended	Subtyping
USE	REF-EXT	Interfaces are similar and transformations are both refined and extended	Subtyping
USE	NONE	Interfaces are similar and transformations are different	Overloading


RentCom WeWork FindRoommate
Parametric filter (PARAM)
Sub-Typing filter (ST)
Overloading filter (OVER)
0 10 20 30 40 50 60 70 80 90 100
Zoom Out
Pause
Hide classes
Filter Files
OfficeType
OfficeStatus
HouseType
Amenity
Facility
Amenity
60 % in PARAM
100% in PARAM
Office
House
Room
16 % in ST
7% in ST
10 % in ST
GenderConstraint
House
HouseType
FacilityStatus
RoomStatus
Client
Client
Roommate
50% in PARAM
62% in PARAM
Change color of edges with the parameters:
70% | 20% | 50%
Change color of classes from:
RentCom
WeWork
FindRoommate

The VarMeR Tool

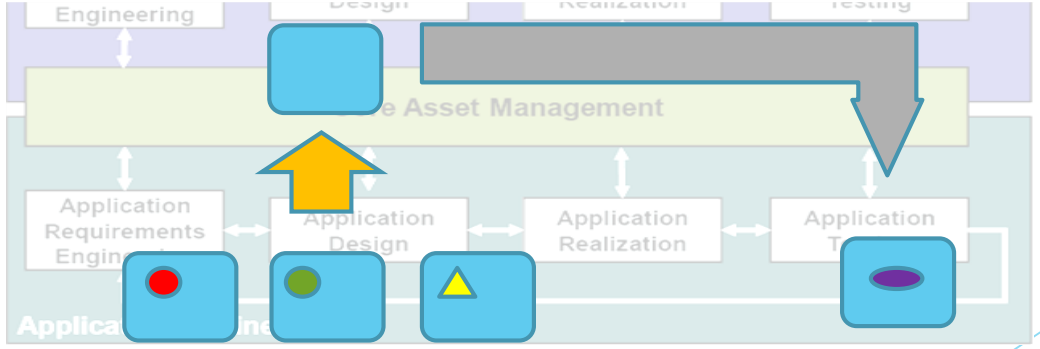
https://sites.google.com/is.haifa.ac.il/varmer/

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

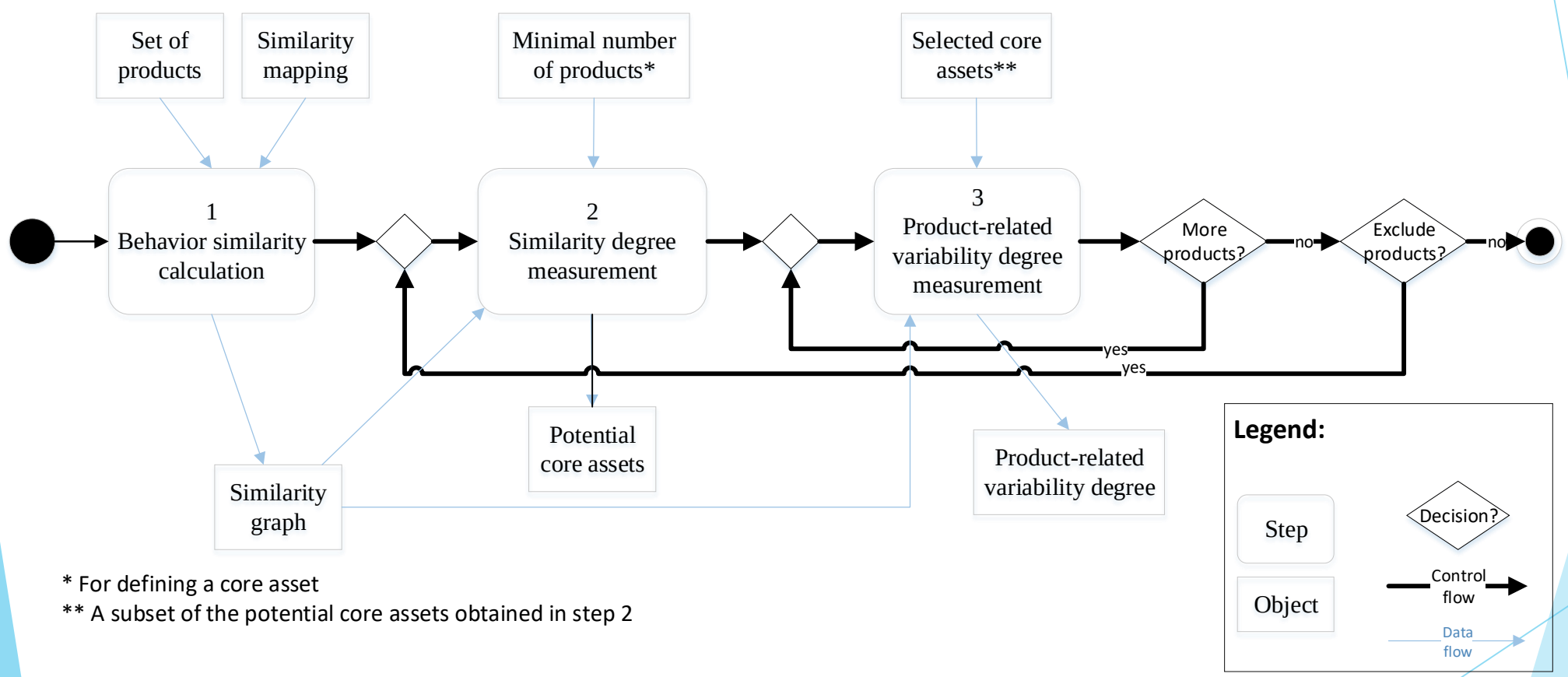
- Suggested metrics assume high similarity of representations, mainly implementations or architecture models (Berger et al. 2014)

Name	Formula	Description
Size of commonality (SoC)	SoC = $\lvert \bigcap_{i=1..n} C_{p_i,r} \rvert + \lvert \bigcap_{i=1..n} C_{p_i,o} \rvert$	Number of identical components among $p_1,\ldots,p_n$
Product related reusability (PrR)	$PrR_i = \frac{SoC}{\lvert C_{p_i,r} \cup C_{p_i,o} \rvert}$	Ratio relating the size of commonality for a specific product p_i

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

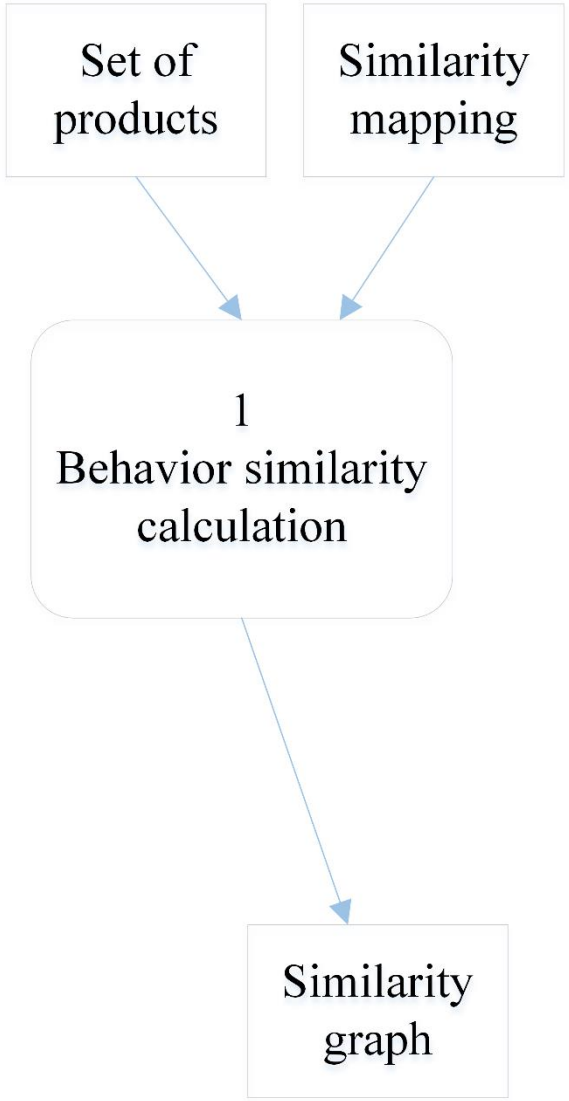
Product line ability decision support

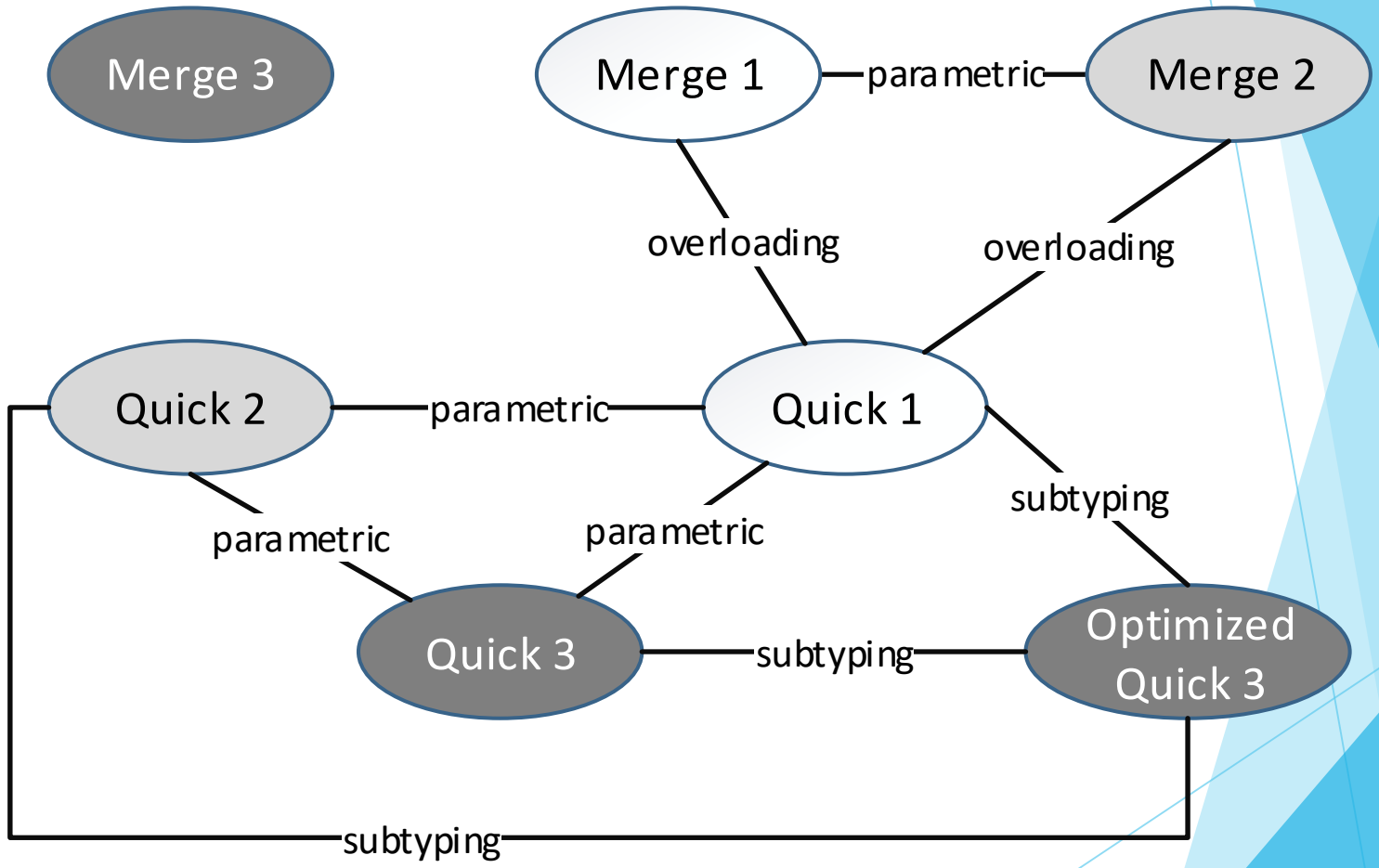


* For defining a core asset

** A subset of the potential core assets obtained in step 2

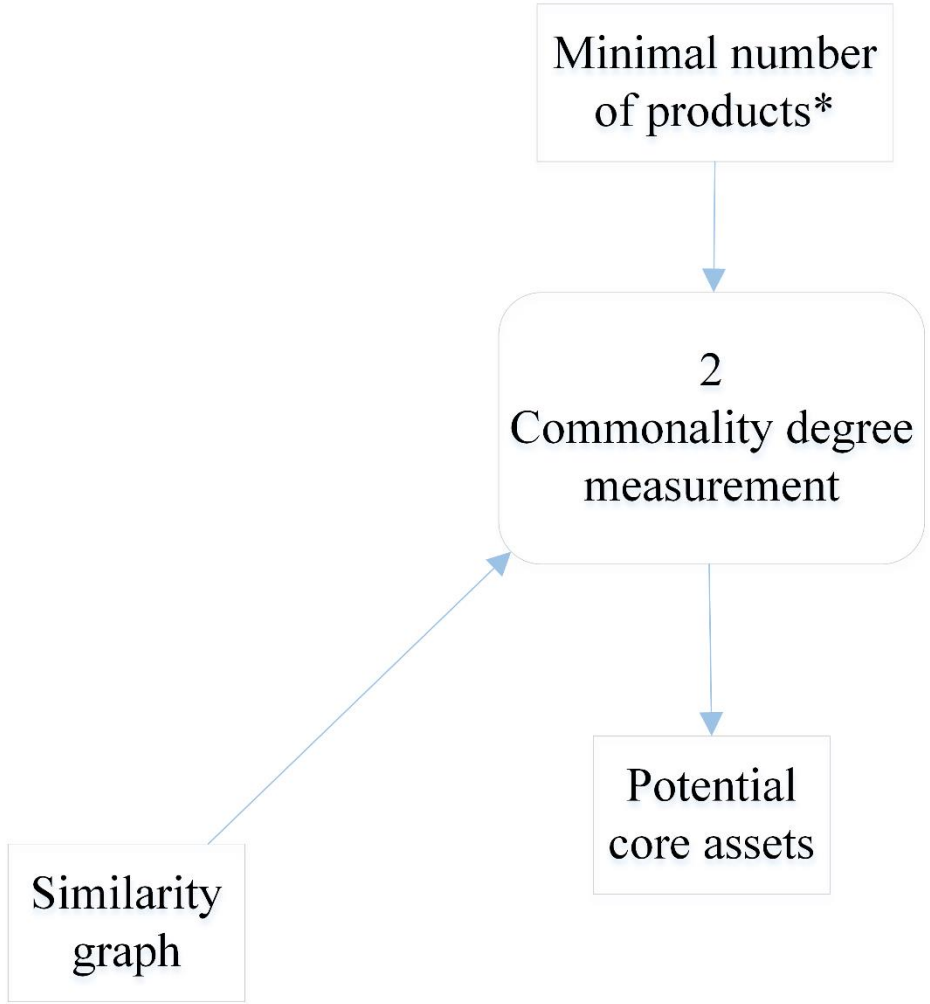
Product line ability decision support



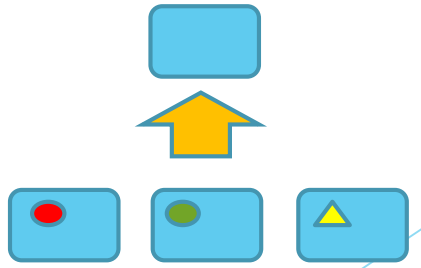


* Node's Color represents the product

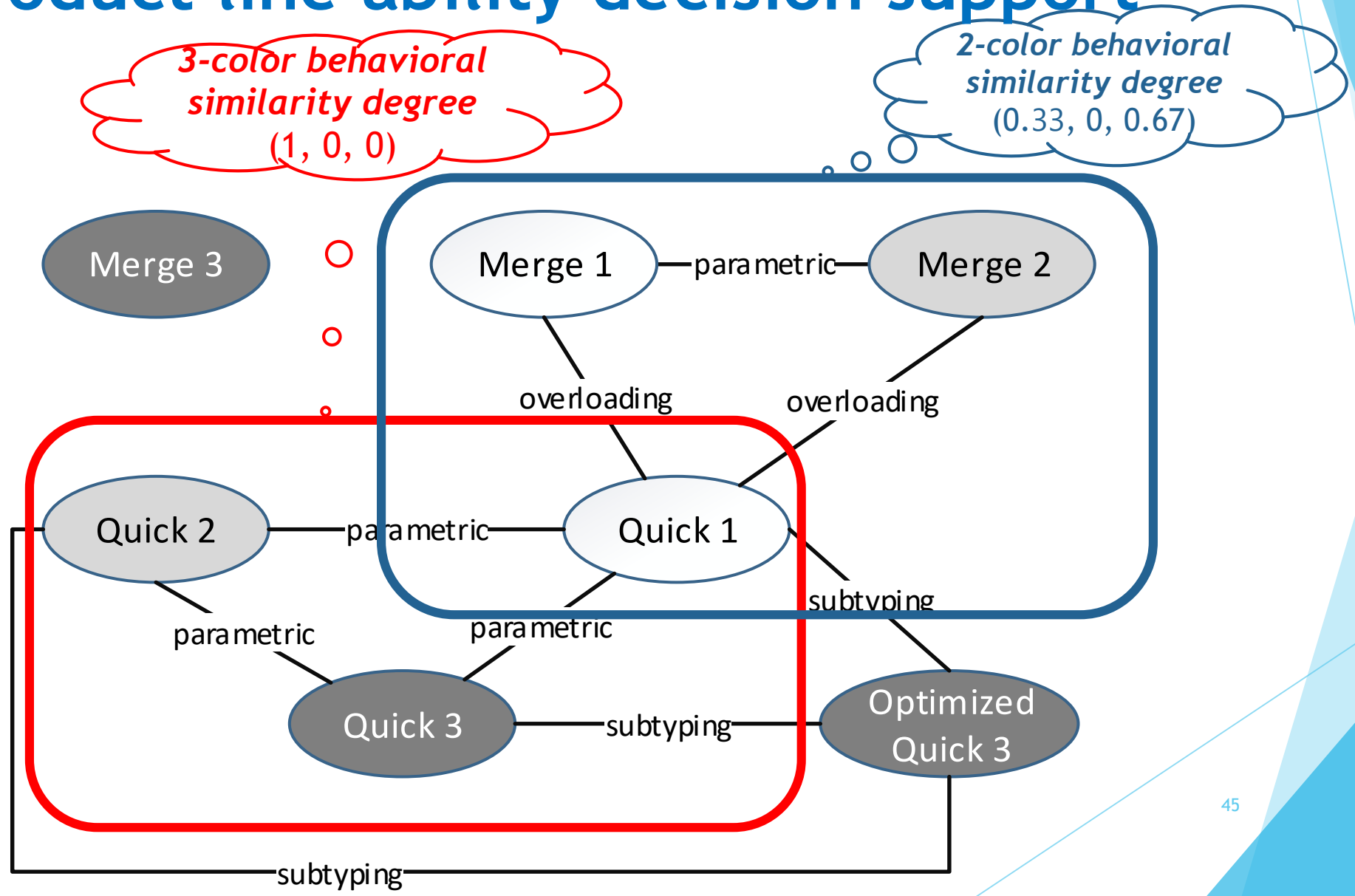
Product line ability decision support



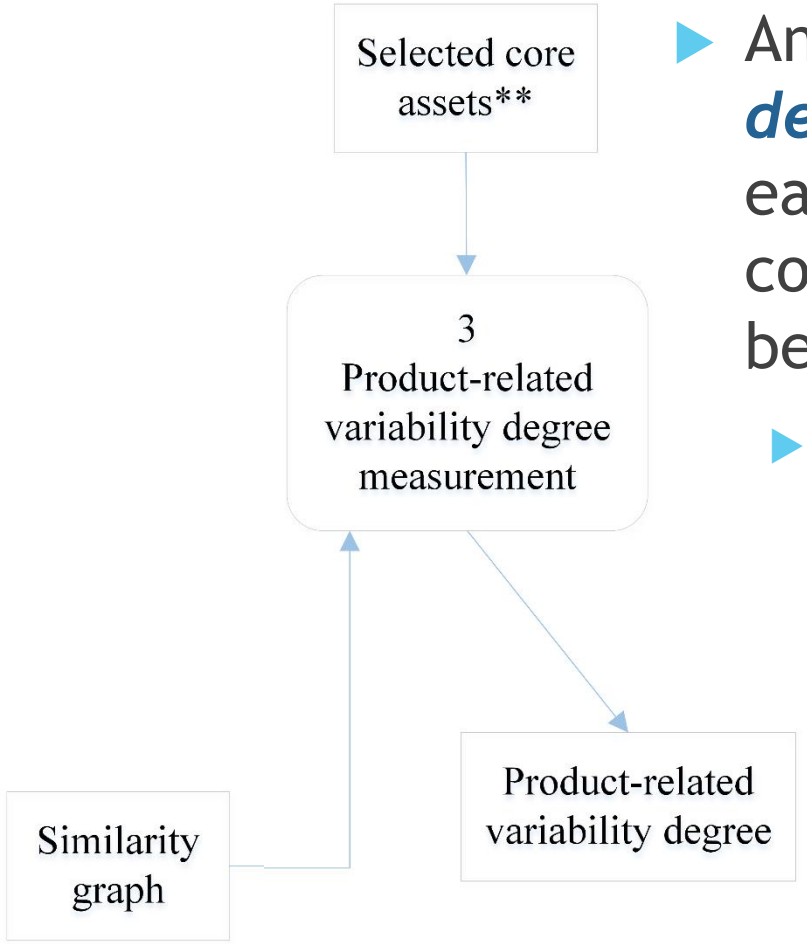
- An ***m-colored parametric asset*** is a subgraph of a similarity graph representing at least m products (colors) where each two nodes are connected with a parametric edge
- An ***m-color behavioral similarity degree*** measures how "close" a given similarity m-colored sub-graph is to being an m-colored parametric asset.



Product line ability decision support



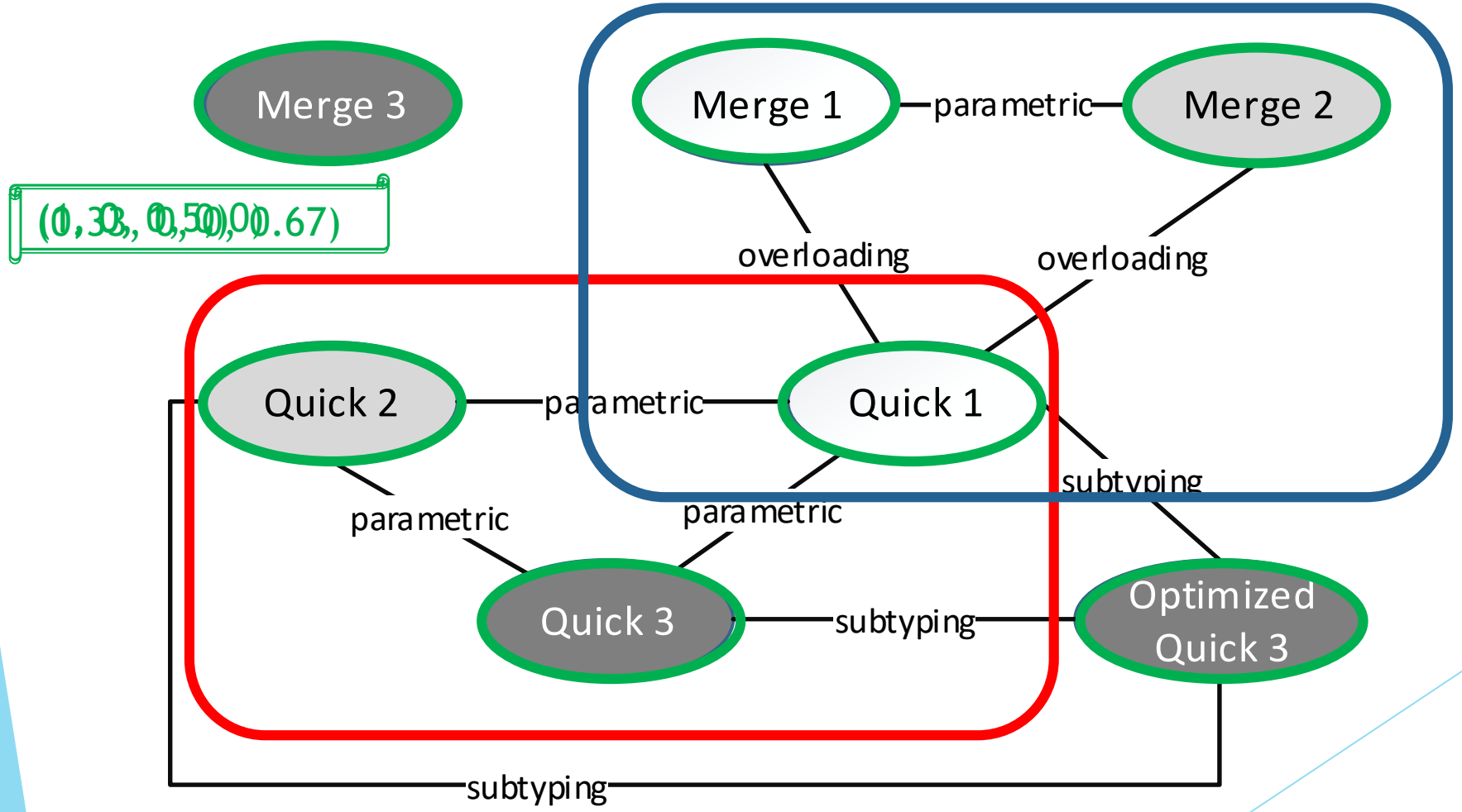
Product line ability decision support



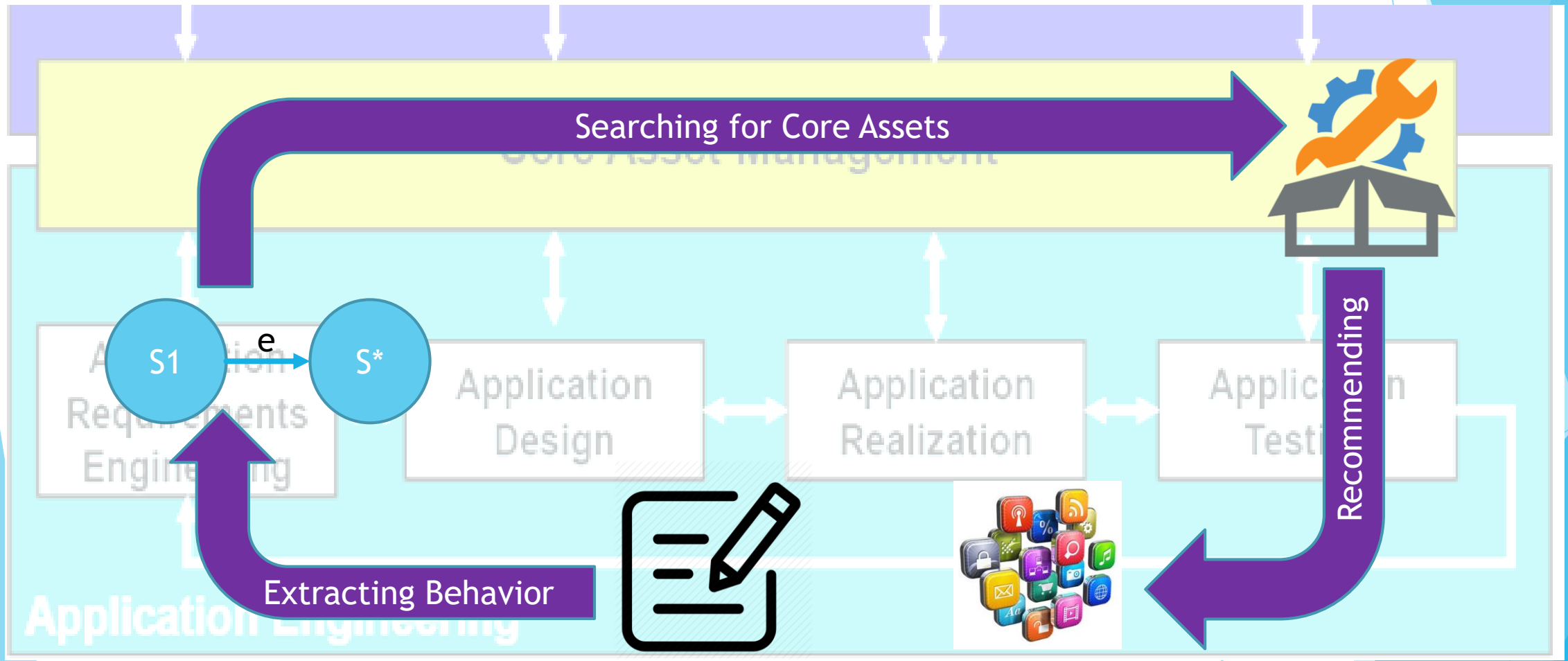
- An ***m-color product-related variability degree*** measures the difference between each product and the potential m-color core assets, as captured by the m-color behavioral similarity degree
  - Intuitively, greater coverage of vertices indicates higher product line-ability.

Product line ability decision support

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



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

- I. Reinhartz-Berger, A. Zamansky. A Behavior-based Framework for Assessing Product Line-Ability. CAiSE 2018: 571-586.
- I. Reinhartz-Berger, Anna Zamansky. VarMeR - A Variability Mechanisms Recommender for Software Artifacts. CAiSE-Forum-DC2017, 57-64.
- A. Zamansky, I. Reinhartz-Berger. Visualizing Code Variabilities for Supporting Reuse Decisions. SCME 2017.
- I. Reinhartz-Berger, A. Zamansky, Y. Wand, An Ontological Approach for Identifying Variants: The Cases of Specialization and Template Instantiation, ER'16.
- I. Reinhartz-Berger, A. Zamansky and Y. Wand. Taming Software Variability: Ontological Foundations of Variability Mechanisms, ER'15.
- I. Reinhartz-Berger, A. Zamansky and M. Kemelman. Analyzing Variability of Cloned Artifacts: Formal Framework and its Application to Requirements. EMMSAD'15.

Additional References mentioned in the presentation

- Clone detection:
  - Rattan, D., Bhatia, R., & Singh, M. (2013). Software clone detection: A systematic review. Information and Software Technology, 55(7), 1165-1199.
- Variability mechanisms:
  - Bachmann, F., & Clements, P. C. (2005). Variability in software product lines (No. CMU/SEI-2005-TR-012). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.
  - Gacek, C., & Anastasopoules, M. (2001, May). Implementing product line variabilities. In ACM SIGSOFT Software Engineering Notes (Vol. 26, No. 3, pp. 109-117). ACM.
  - I. Jacobson, M. Griss, P. Jonsson. (1997). Software reuse: architecture process and organization for business success. 1. ed. Boston: Addison-Wesley, p. 528.
  - Svahnberg, M., Van Gurp, J., & Bosch, J. (2005). A taxonomy of variability realization techniques. Software: Practice and experience, 35(8), 705-754.
  - vom Brocke, J. (2007). Design principles for reference modeling: reusing information models by means of aggregation, specialisation, instantiation, and analogy. In Reference modeling for business systems analysis (pp. 47-76). IGI Global.

Additional References mentioned in the presentation

- Variability mechanisms (cont.):
  - Becker, J., Delfmann, P., & Knackstedt, R. (2007). Adaptive reference modeling: Integrating configurative and generic adaptation techniques for information models. In Reference modeling (pp. 27-58). Physica-Verlag HD.
  - Bachmann, F., & Clements, P. C. (2005). Variability in software product lines (No. CMU/SEI-2005-TR-012). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.
  - Muthig, D., & Patzke, T. (2002, October). Generic implementation of product line components. In Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (pp. 313-329). Springer, Berlin, Heidelberg.
- But4Reuse
  - Martinez, J., Ziadi, T., Bissyandé, T. F., Klein, J., & Le Traon, Y. (2015, July). Bottom-up adoption of software product lines: a generic and extensible approach. In Proceedings of the 19th International Conference on Software Product Line (pp. 101-110). ACM.