MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Search Based Software Engineering: Foundations, Challenges and Recent Advances

Marouane Kessentini
marouane@umich.edu

SBSE Research Lab, CIS Department,
College of Engineering and Computer Science,
University of Michigan, Dearborn, USA

IEEE **WCCI 2016** | 2016 IEEE World Congress on Computational Intelligence.

Acknowledgments

- Many thanks to Prof. Mark Harman (Founder of Search-Based Software Engineering) for the help to prepare part of this tutorial from the following source:

  – Mark Harman, UCL, UK
  Search Based Software Engineering: Automating Software Engineering, FSE2011, Technical Briefings.

Outline

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent Advances
  - Bi-Level SBSE for Design Defects Detection
  - Interactive Multi-Objective SBSE for Refactoring
  - Many-Objective SBSE for Software Remodularization
- Challenges and Future Research Directions

Outline

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent Advances
  - Bi-Level SBSE for Design Defects Detection
  - Interactive Multi-Objective SBSE for Refactoring
  - Many-Objective SBSE for Software Remodularization
- Challenges and Future Research Directions

Outline

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent Advances
  - Bi-Level SBSE for Design Defects Detection
  - Interactive Multi-Objective SBSE for Refactoring
  - Many-Objective SBSE for Software Remodularization
- Challenges and Future Research Directions

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Scientists' and Engineers' Viewpoints

Scientist:

What is true

Correctness

Model the world to understand

Engineer:

What is possible

Within tolerance

Model the world to manipulate

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Scientists' and Engineers' Viewpoints

Computer scientist:

What is true about computation

Proof correctness
Make it perfect

Software engineer:

What is possible with software

Test for imperfection
find where to improve

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Combining Science and Engineering

prove correctness
make it perfect

where possible ...

... and where impossible ...

test for imperfection
find where to improve

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Engineering Words

tolerance

With acceptable bounds

optimise

Improve performance

Reduce cost

Optimize

Within constraints

Optimization: so good they named it twice!

First in English ...

Then in American

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

What is SBSE?

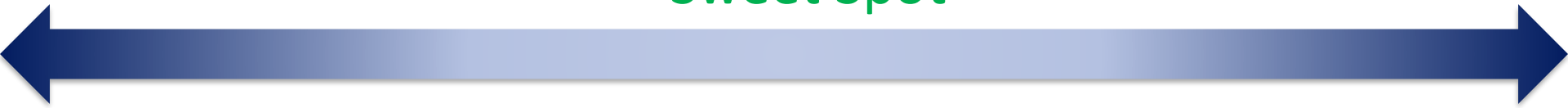
- In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

like google search?

like code search?

like breadth first search?

Exhaustive Sweet Spot *Random*



What is SBSE?

- In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

Genetic Programming

Ant Colonies

Hill Climbing

Harmony Search

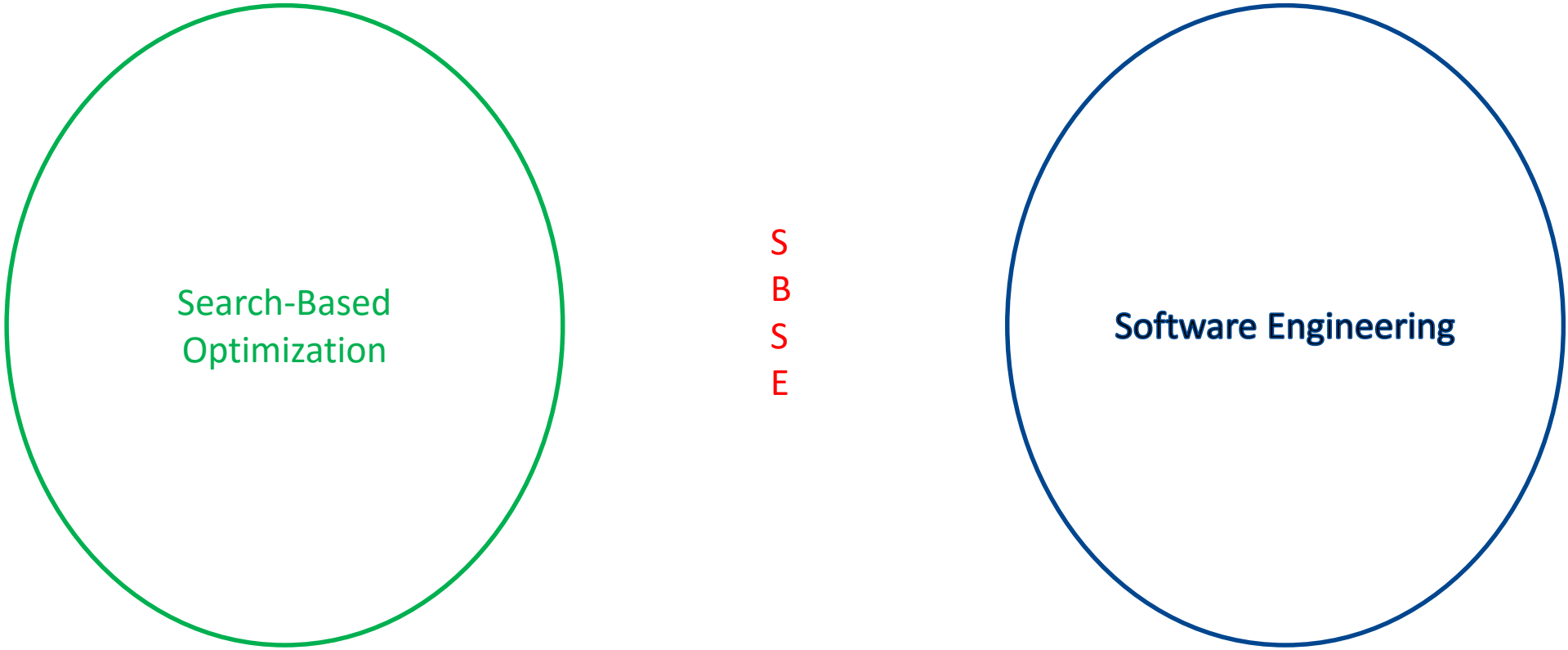
Tabu Search

Particle Swarm Optimization

Simulated Annealing

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

What is SBSE?



Checking vs Generating

- Search Based Software Engineering
  - Write a method to determine which is the better of two solutions
- Conventional Software Engineering
  - Write a method to construct a perfect solution

Checking vs Generating

- Search Based Software Engineering
  - Write a fitness function to guide automated search
- Conventional Software Engineering
  - Write a method to construct a perfect solution

…but…

why is Software Engineering different?

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

In situ fitness test

Physical Engineering

Cost: 20,000$

Virtual Engineering

Cost: 0 $

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Spot the Difference

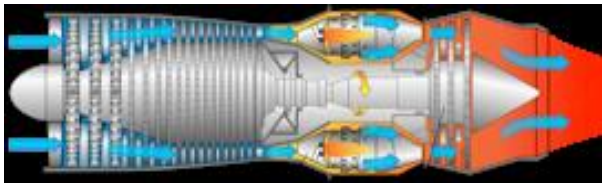
Traditional Engineering Artifact

Optimization goals

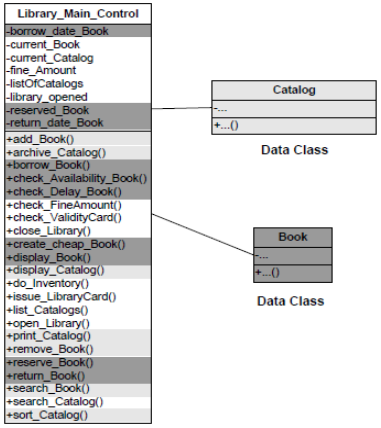
Maximize compression

Minimize fuel consumption

Fitness computed on a representation



Traditional Engineering Artifact

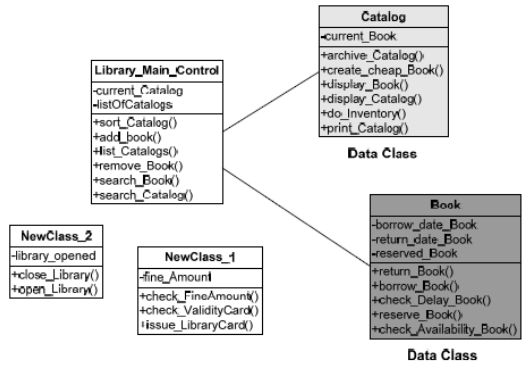


Optimization goals

Maximize cohesion

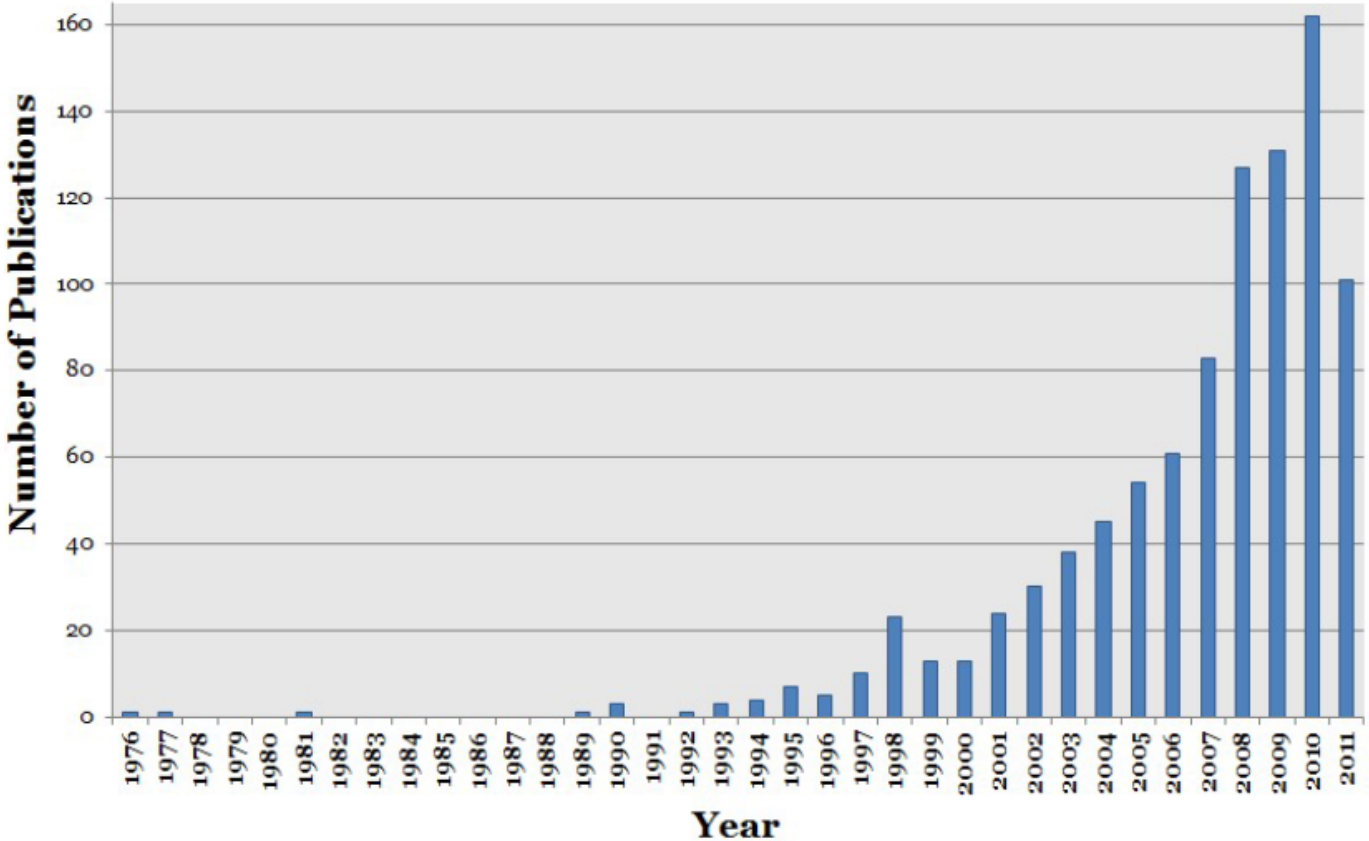
Minimize coupling

Fitness computed on a representation



…but…

why is SBSE growing very fast?



Software Engineers …

let’s listen to software engineers ...

... what sort of things do they say?

Software Engineers Say…

Requirements:	We need to satisfy business and technical concerns
Management:	We need to reduce risk while maintaining completion time
Design:	We need increased cohesion and decreased coupling
Testing:	We need fewer tests that find more nasty bugs
Refactoring:	We need to optimize for all metrics M1,..., Mn

All have been addressed in the SBSE literature

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Software Engineers Say…

Capture requirements | **Generate tests** | **Model Transformation** | **Refactoring**

Minimize

- Cost
- Development time

Maximize

- Satisfaction
- Fairness

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Software Engineers Say…

Capture requirements | **Generate tests** | **Model Transformation** | **Refactoring**

Minimize

- Number of test
- Execution time

Maximize

- Code coverage
- Fault coverage

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Software Engineers Say…

Capture requirements	Generate tests	Model Transformation	Refactoring
Minimize		• Rules correctness	
Maximize		• Rules complexity • Models quality	

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Software Engineers Say…

Capture requirements | **Generate tests** | **Model Transformation** | **Refactoring**

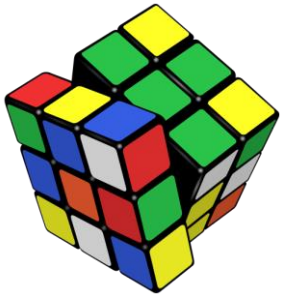
Minimize

- Number of refactorings

Maximize

- Quality factors
- Semantics preservation

The Advantages of SBSE



Generic

Scalable

Robust

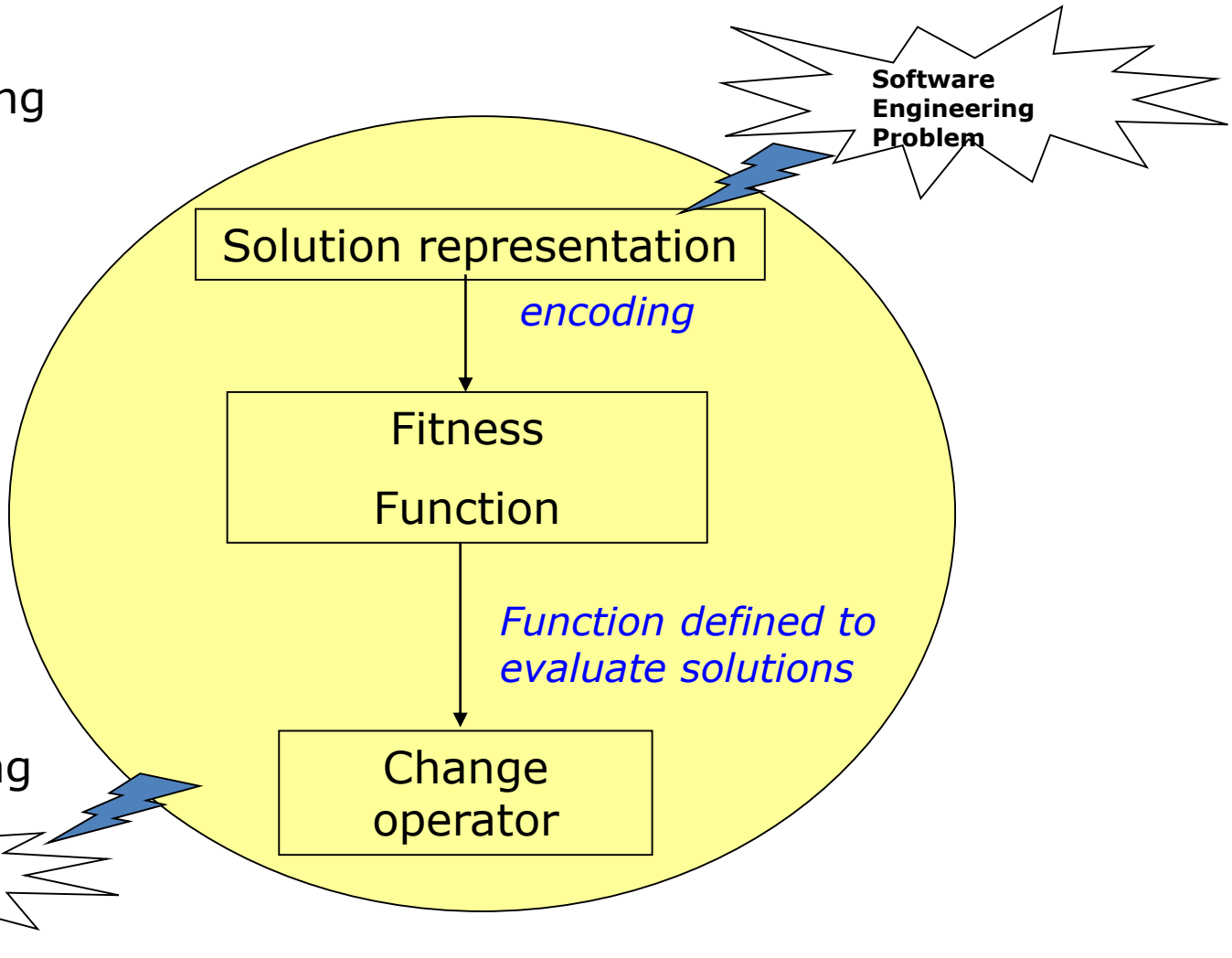
Realistic

SBSE is so generic…

Software Engineering

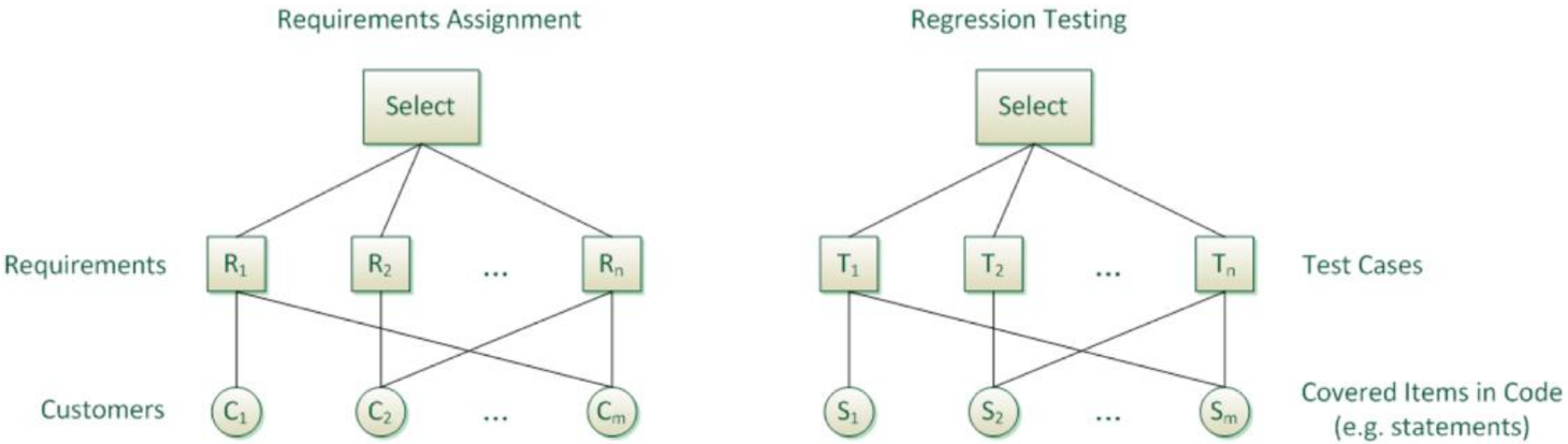
Optimization Techniques

Search Based Software Engineering

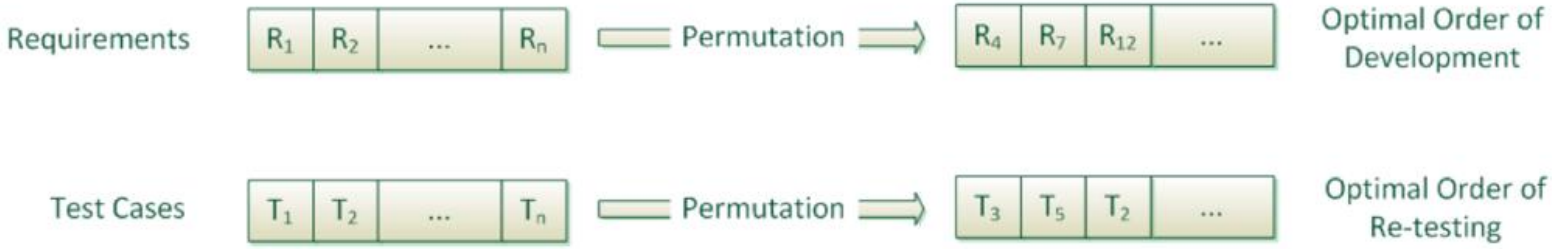


SBSE is so generic…

Selection Problems



Prioritization Problems



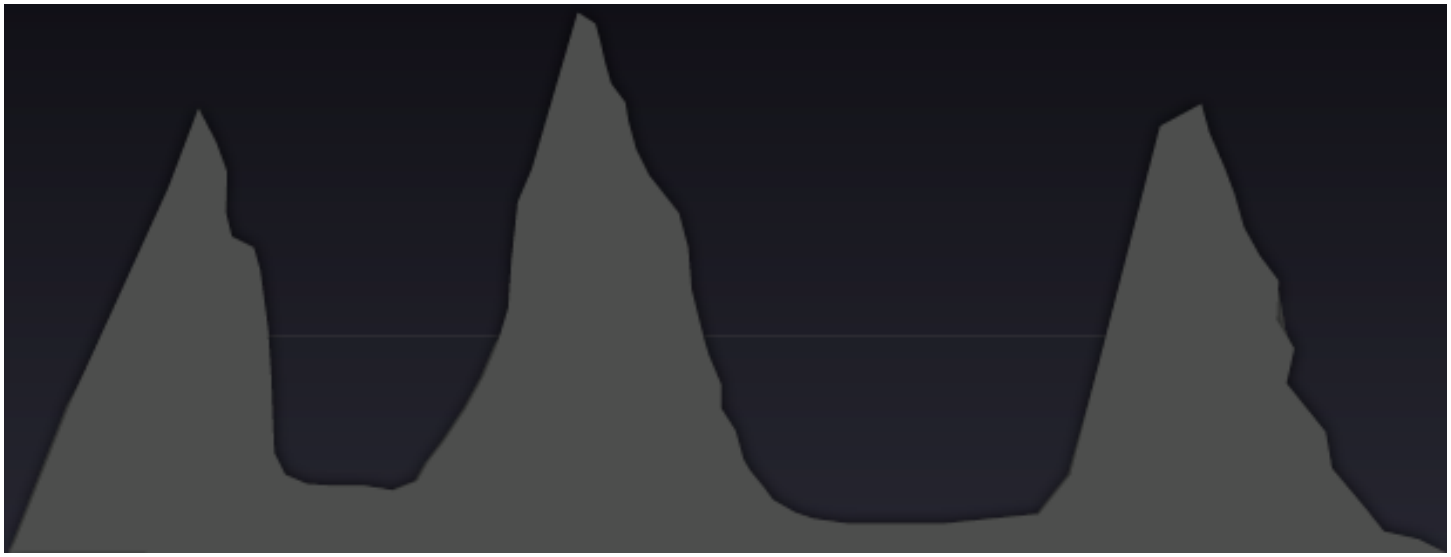
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Requirements and regression testing: Really different?

Alone

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

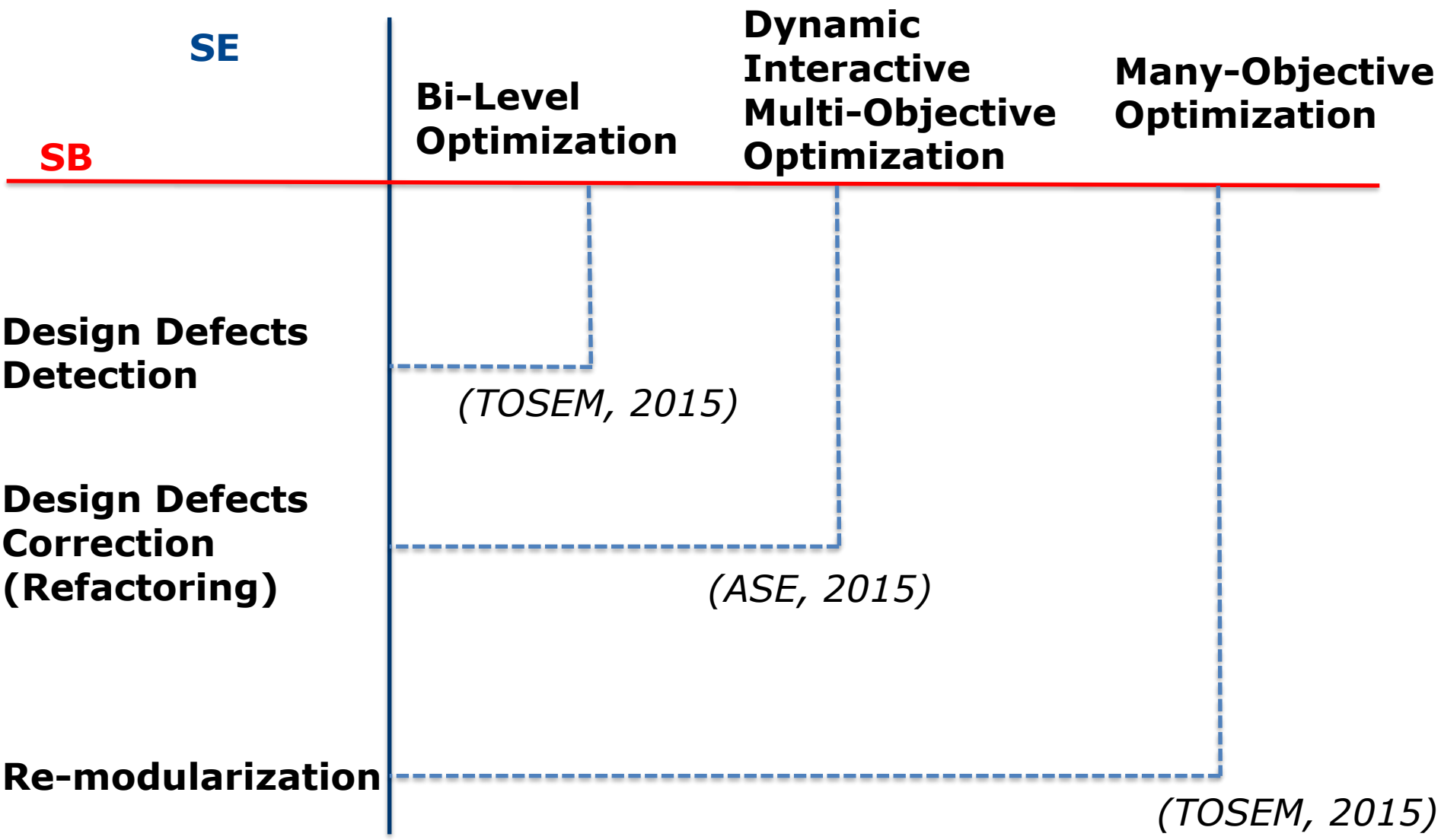
Requirements and regression testing: Really different?



All one

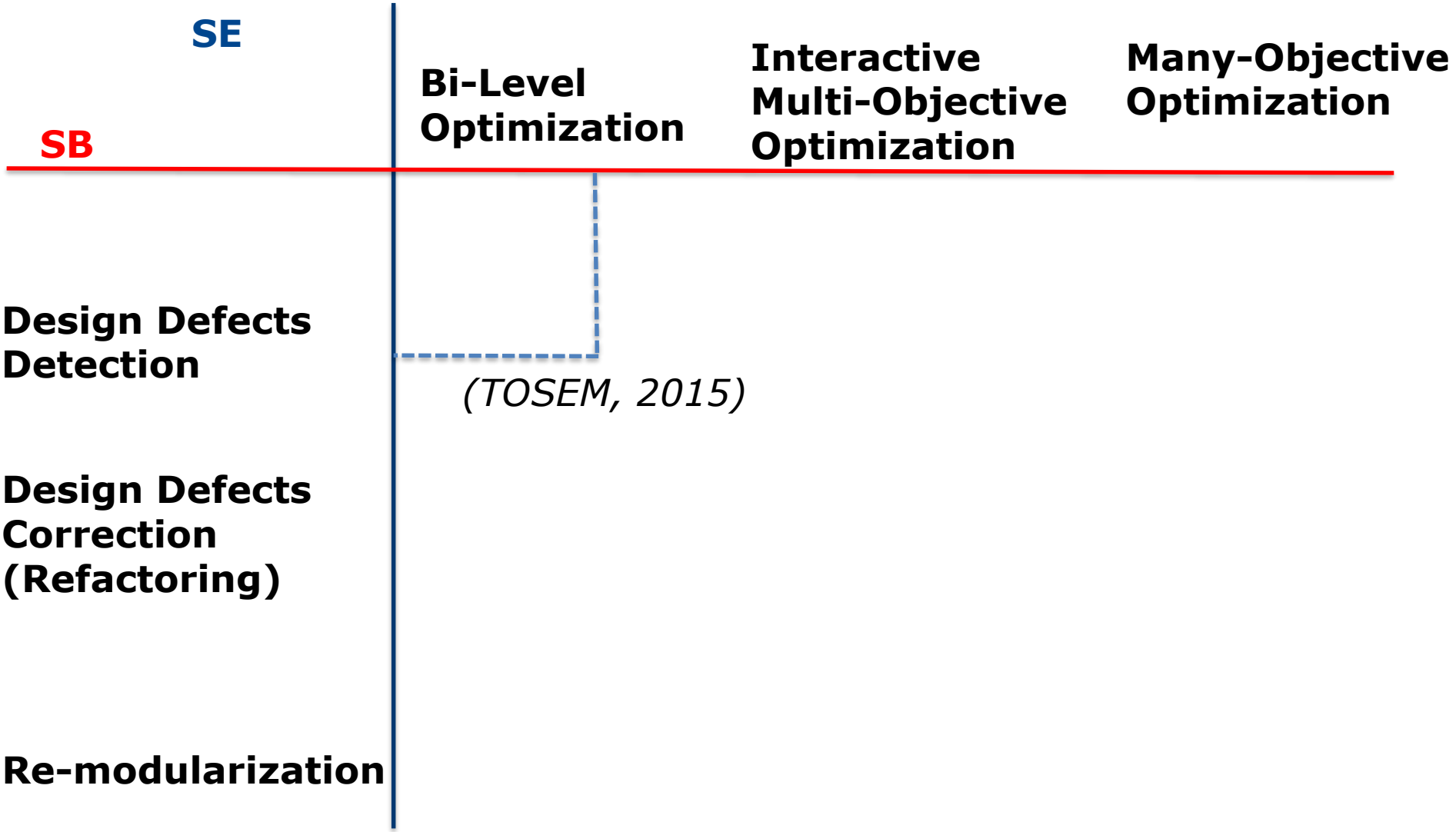
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Our Recent Advances in SBSE



MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Our Recent Advances in SBSE



Software Refactoring

- Software changes frequently
  - Add new functionalities
  - Correcting bugs
  - Adaptation to environment changes
  - Software engineers spend 60% of their time in understanding the code
- Easiness to accommodate changes depends on the software quality
  - Refactoring

Software Refactoring

- Refactoring
  - The process of improving a code after it has been written by changing its internal structure without changing the external behavior (Fowler et al., '99)
  - Examples: *Move method, extract class, move attribute, ...*

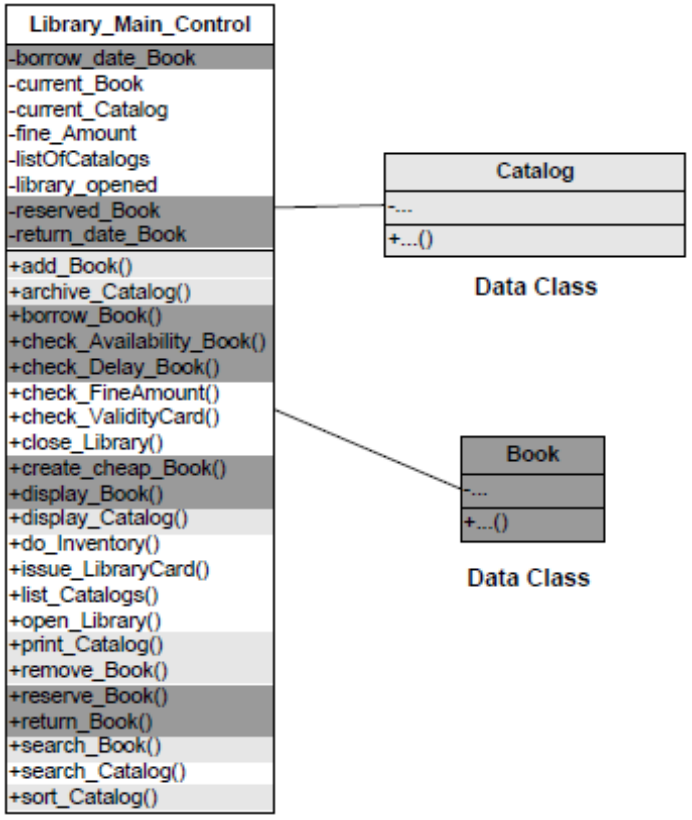
- Two main refactoring steps
  1. detection of code fragments to improve (e.g., design defects)
  2. identification of refactoring solutions

Step 1: Design defects detection

- Design defect introduced during the initial design or during evolution
  - Anomalies, anti-patterns, bad smells…
  - Design situations that adversely affect the development of a software (not bugs)
  - Examples: *Blob, spaghetti code, functional decomposition*, ...

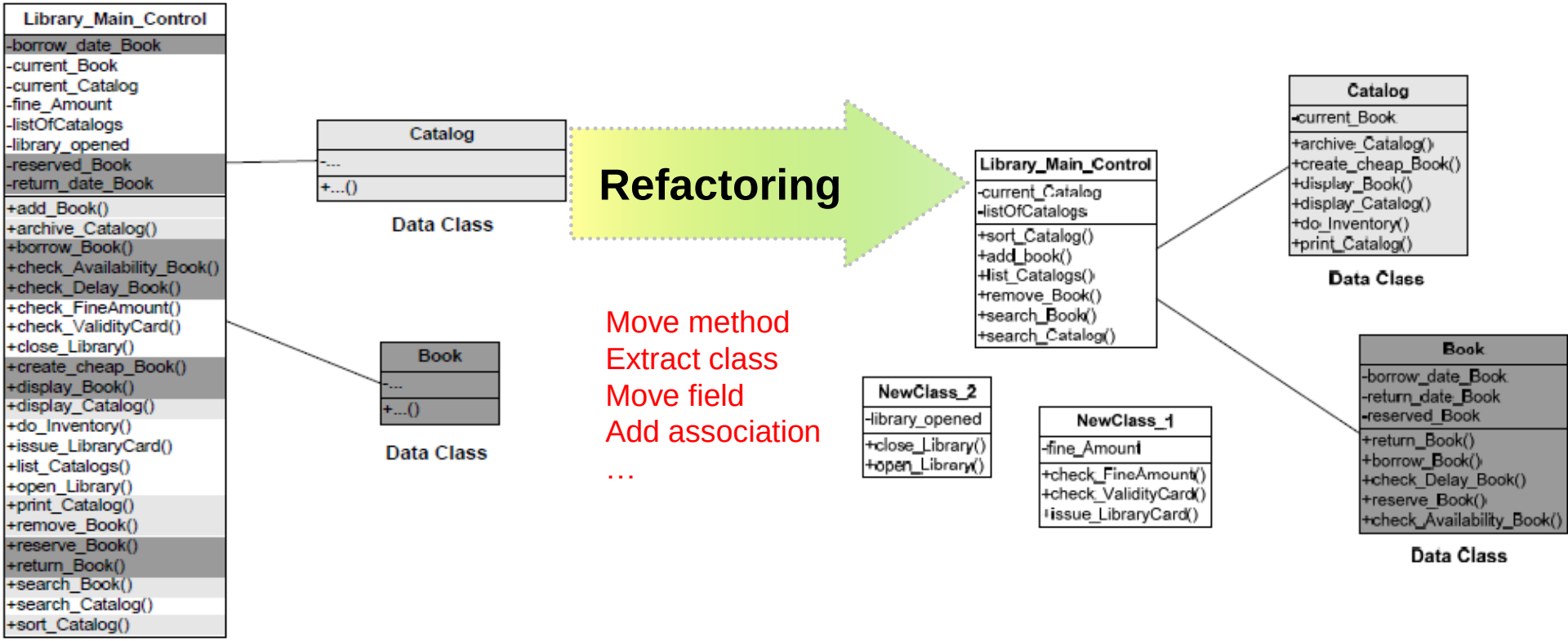
The Blob Example

- Definition
  - Procedural-style design leads to one object with numerous responsibilities and most other objects only holding data or executing simple processes.
- Symptoms
  - A Blob is a *controller* class, *abnormally large*, with *almost* no parents and no children. It *mainly uses* data classes, i.e. *very small* classes with *almost* no parents and no children (Brown et al. '98).



MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Step 2: Refactoring



Blob

Existing Work

- Design defects detection
  - Manual (Brown et al. ‘98, Fowler and Beck ‘99)
  - Metrics-based (Marinescu et al. ’04, Salehie et al. ’06, Maiga et al.‘12)
  - Visual (Dhambri et al. ’08, Langelier et al. ’05)
  - Symptoms-based (Moha et al. ’08, Murno et al. ‘08)

Definition ➔ symptoms ➔ detection algorithm

Existing Work

- Detection issues
  - No consensual definition of symptoms
  - The same symptom could be associated to many defect types
  - Difficulty to automate symptom's evaluation
  - Require an expert to manually write and validate detection rules

Limitations ...

- Detection rules heavily depend on the base of examples (coverage, quality, etc.)
- To generate good detection rules, we need to have examples from similar contexts

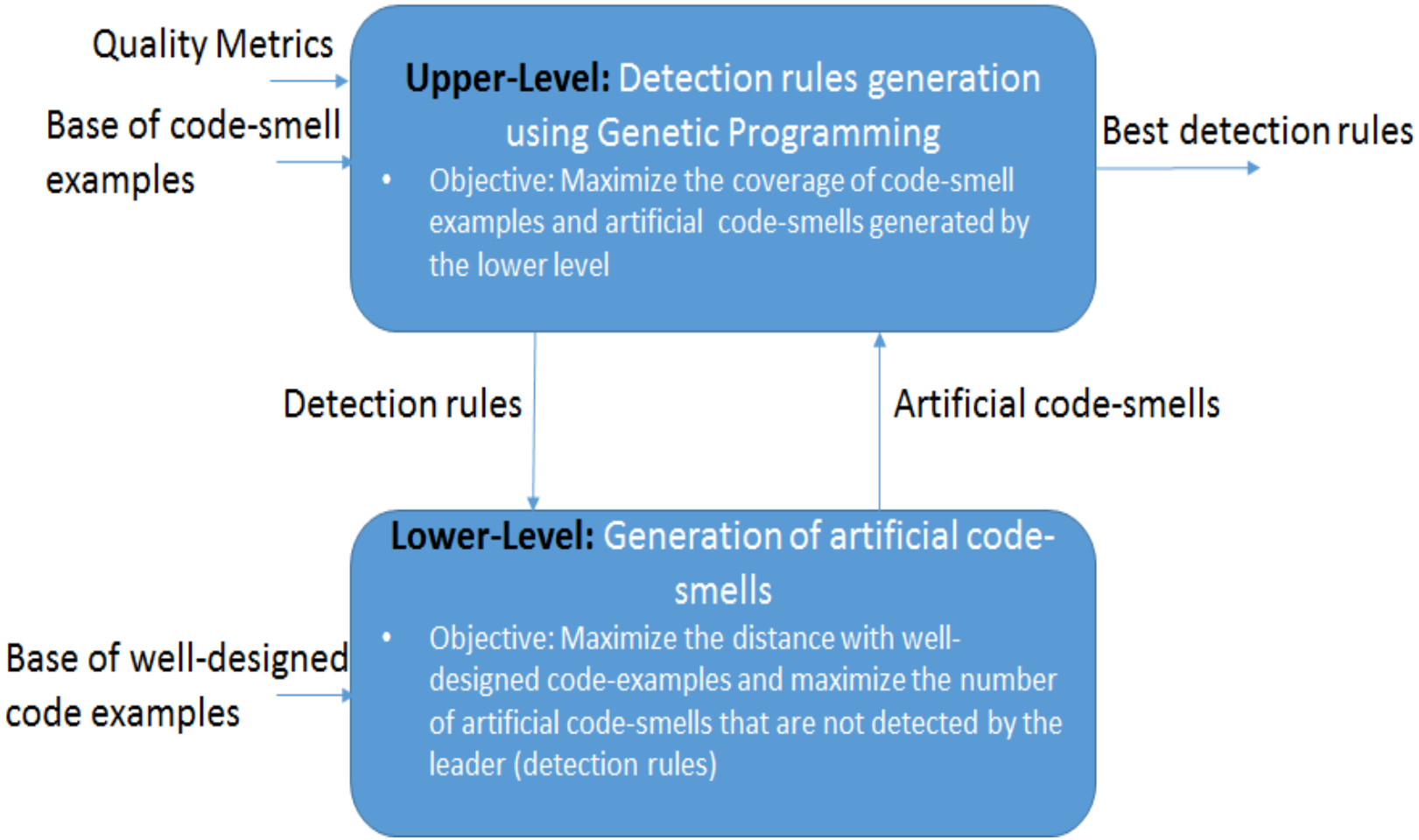
Problem

- Large exhaustive list of quality metrics
- Large number of possible threshold values

Search problem to explore this huge space

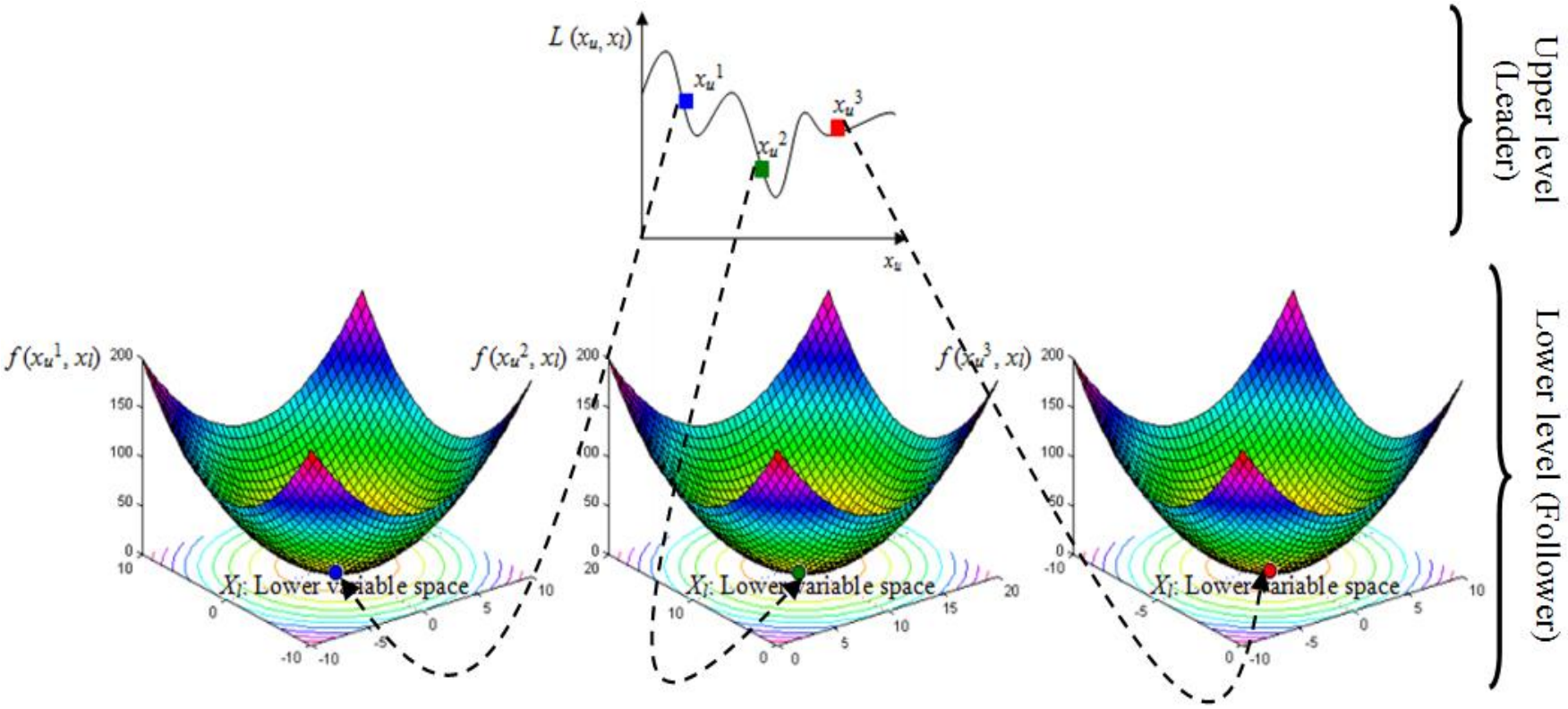
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Bi-Level Code-Smells Detection

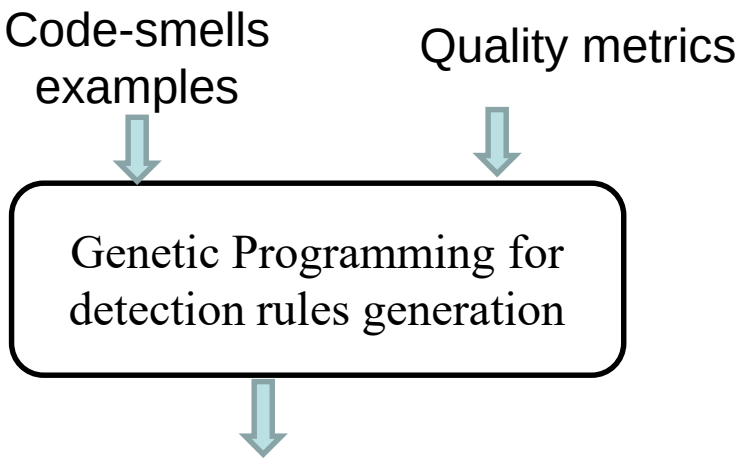


MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

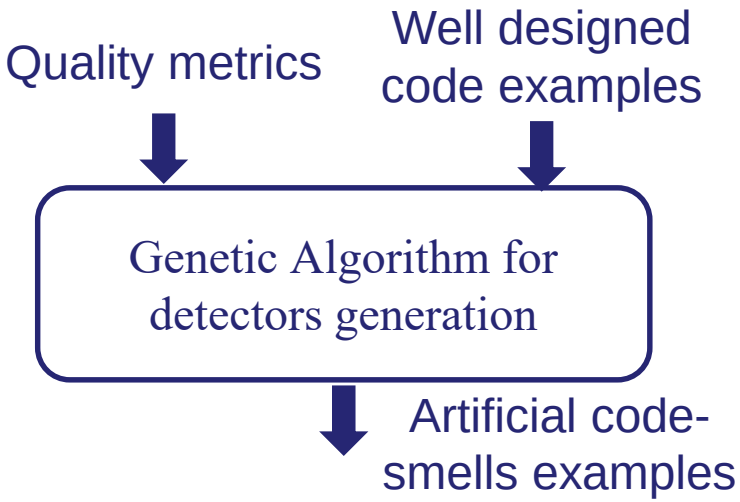
Bi-Level Optimization



Solution Representation

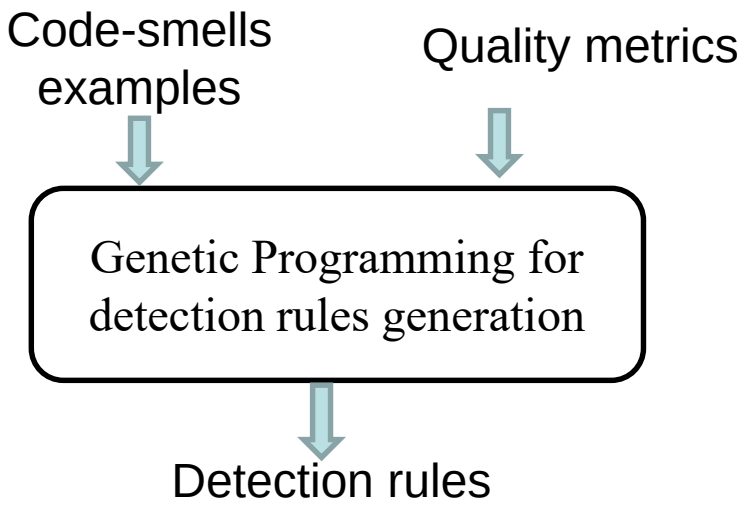


- **Tree :**
  - Leaf node (Terminal) : metrics and their thresholds
  - Internal node (Functions) : logic operators (AND;OR)

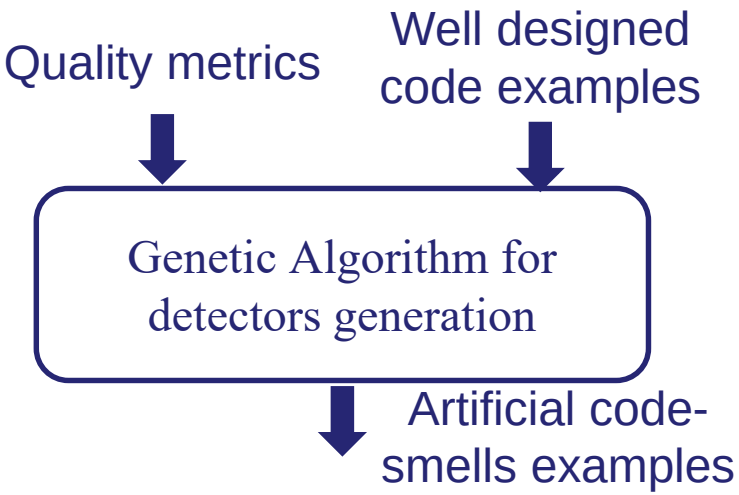


- **Vector :**
  - A set of detectors
  - A detector is composed of 7 metrics

Fitness Functions



- Evaluating detection-rules solutions :
  - Maximize the coverage of the base of code-smell examples
  - Maximize the number of covered “artificial” code-smells generated by the lower level solutions



- Evaluating the artificial code-smells examples :
  - Maximize the dissimilarity score between generated code-smells and reference code
  - Maximize the number of generated code-smell examples un-covered by the upper level solutions

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Validation: Research Questions

- **RQ1:** How does BLOP perform to detect different types of code-smells (Precision and Recall)?
- **RQ2:** How do BLOP perform compared to existing search-based code-smells detection algorithms?
- **RQ3:** How does BLOP perform compared to the existing code-smells detection approaches not based on the use of metaheuristic search?
- **RQ4:** How does our bi-level formulation scale?

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Validation: Studied Systems

Systems	*Release*	*#Classes*	*#Smells*	*KLOC*
JFreeChart	v1.0.9	521	82	170
GanttProject	v1.10.2	245	67	41
ApacheAnt	v1.5.2	1024	163	255
ApacheAnt	v1.7.0	1839	159	327
Nutch	v1.1	207	72	39
Log4J	v1.2.1	189	64	31
Lucene	v1.4.3	154	37	33
Xerces-J	V2.7.0	991	106	238
Rhino	v1.7R1	305	78	57

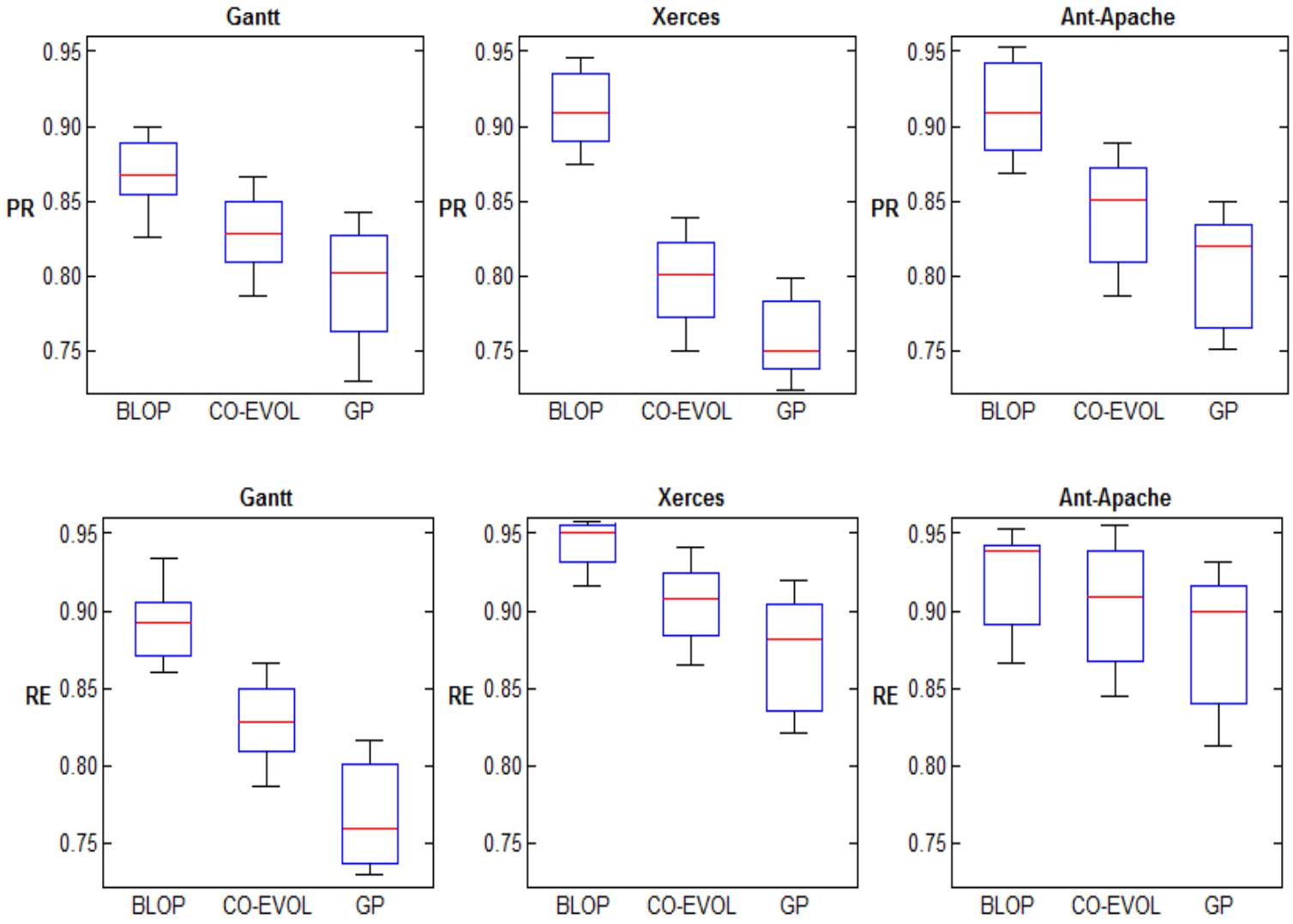
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Results

System	*PR-BLOP*	*PR-GP*	*PR-Co-Evol*	*PR-RS*	*RE-BLOP*	*RE-GP*	*RE-Co-Evol*	*RE-RS*
JFreeChart	89% (77/86)	78% (71/92)	84% (74/89)	26% (34/129)	93% (77/82)	86% (71/82)	90% (74/82)	41% (34/82)
GanttProject	88% (62/71)	80% (57/73)	82% (58/71)	28% (29/106)	89% (62/67)	83% (57/67)	85% (58/67)	43% (29/67)
ApacheAnt v1.5.2	90% (152/169)	84% (146/174)	86% (148/171)	26% (51/189)	93% (152/163)	89% (146/163)	90% (148/163)	31% (51/163)
ApacheAnt v 1.7.0	91% (149/164)	82% (142/173)	85% (144/169)	28% (54/184)	94% (149/159)	90% (142/159)	91% (144/159)	33% (54/159)
Nutch	89% (67/76)	73% (64/88)	76% (65/86)	34% (37/131)	92% (67/72)	89% (64/72)	90% (65/72)	51% (37/72)
Log4J	89% (59/67)	71% (52/74)	77% (54/71)	32% (34/127)	91% (59/64)	81% (52/64)	85% (54/64)	53% (34/64)
Lucene	91% (35/39)	70% (31/42)	79% (33/42)	12% (11/88)	95% (35/37)	84% (31/37)	89% (33/37)	29% (11/37)
Xerces-J	91% (101/111)	75% (94/126)	80% (96/119)	17% (31/179)	95% (101/106)	88% (94/106)	91% (96/106)	29% (31/106)
Rhino	90% (75/84)	74% (69/93)	79% (71/89)	14% (23/167)	95% (75/78)	87% (69/78)	91% (71/78)	29% (23/78)

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

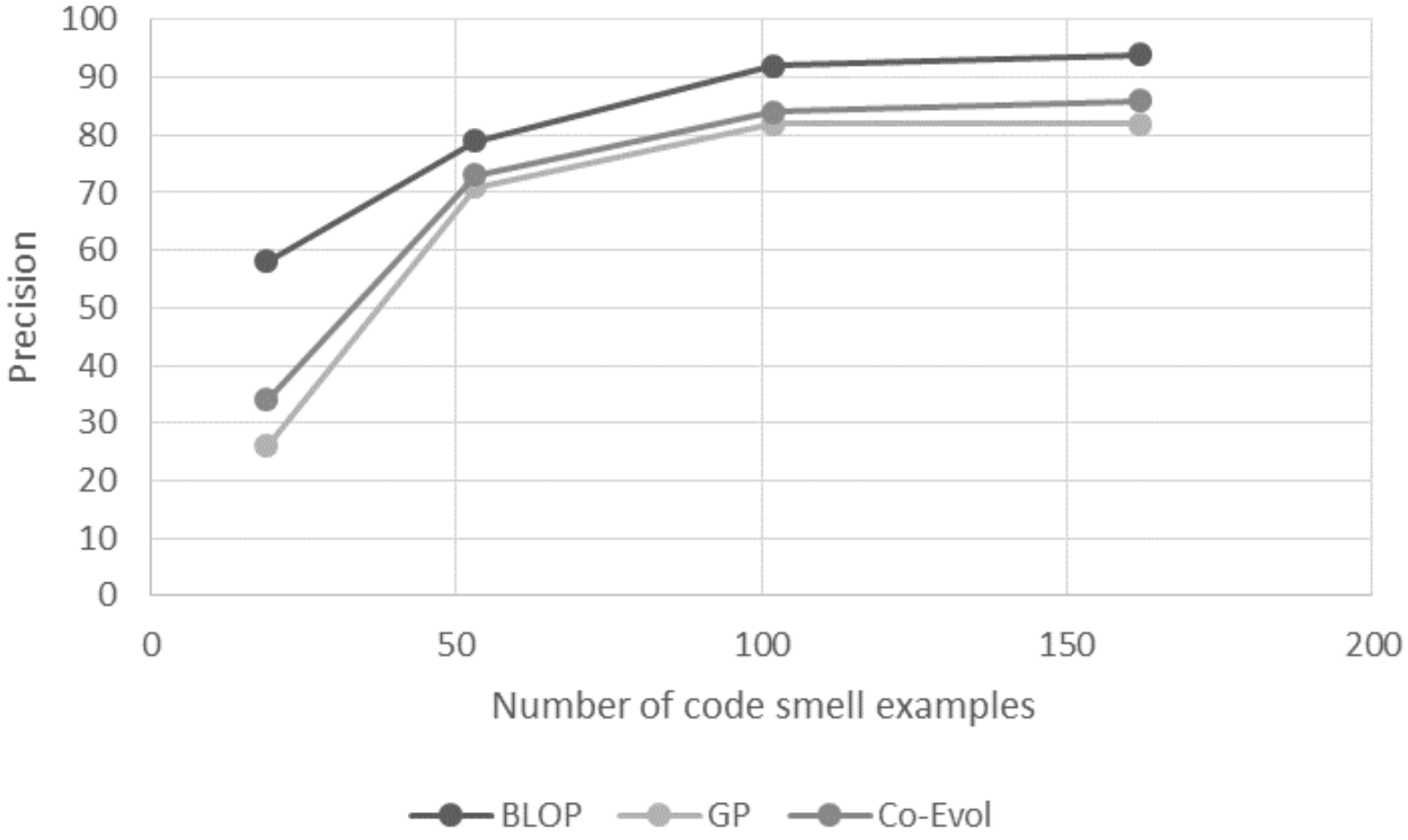
Results



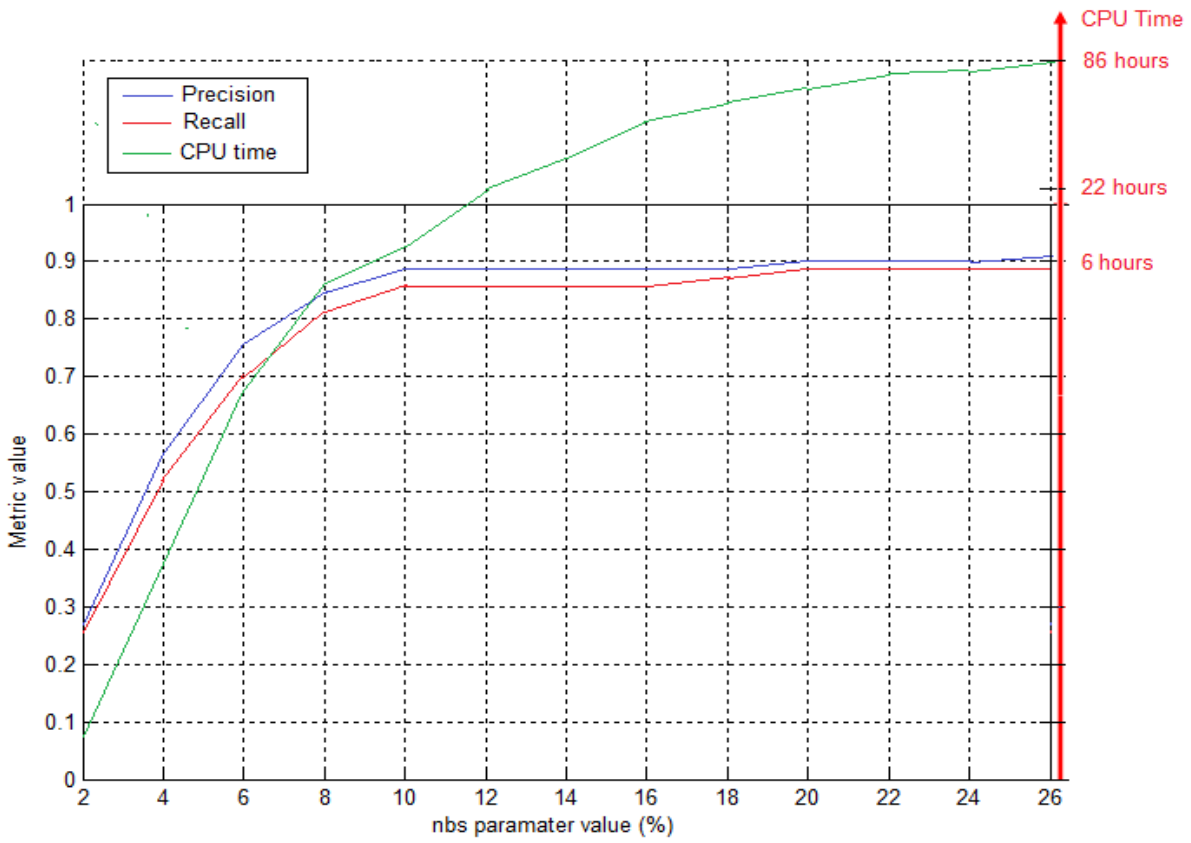
Box plots on three different systems of: (a) precision values, and (b) recall values

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Number of Defect Examples

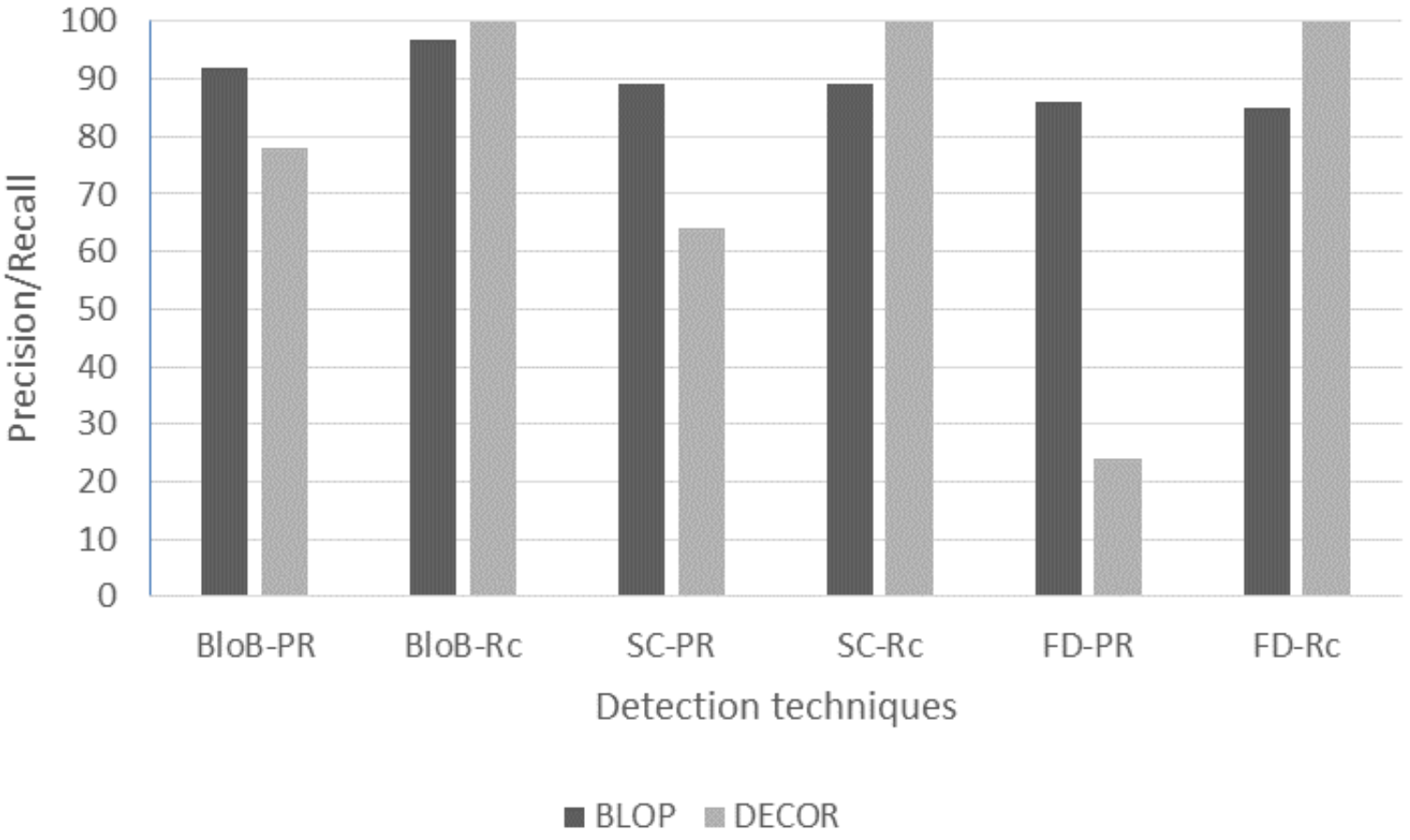


CPU Time


Precision
Recall
CPU time
CPU Time
86 hours
22 hours
6 hours
Metric value
nbs paramater value (%)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
4
6
8
10
12
14
16
18
20
22
24
26

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Comparison with non-search based approach



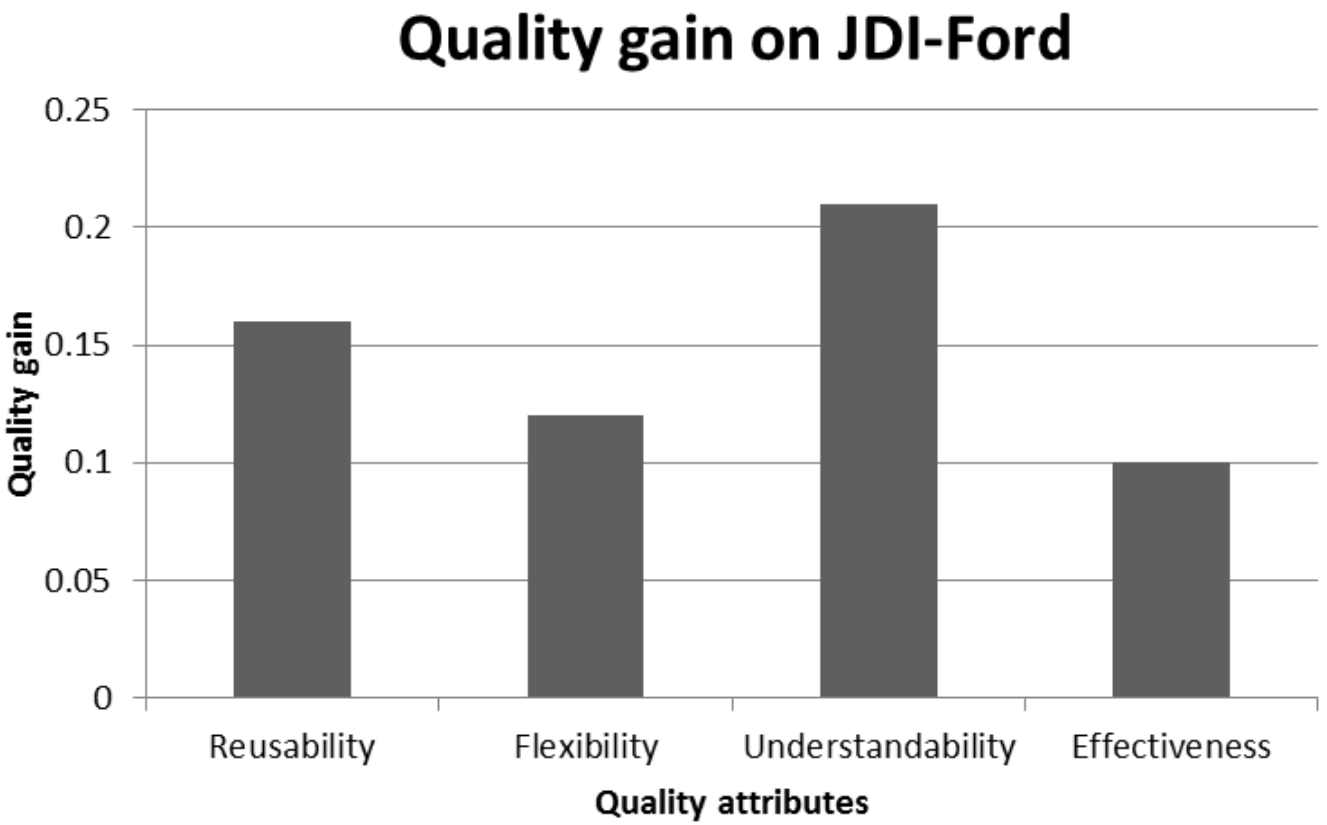
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Industrial Case Study: Ford Motor Company

- **8** software engineers from Ford evaluated the detected defects on the JDI System

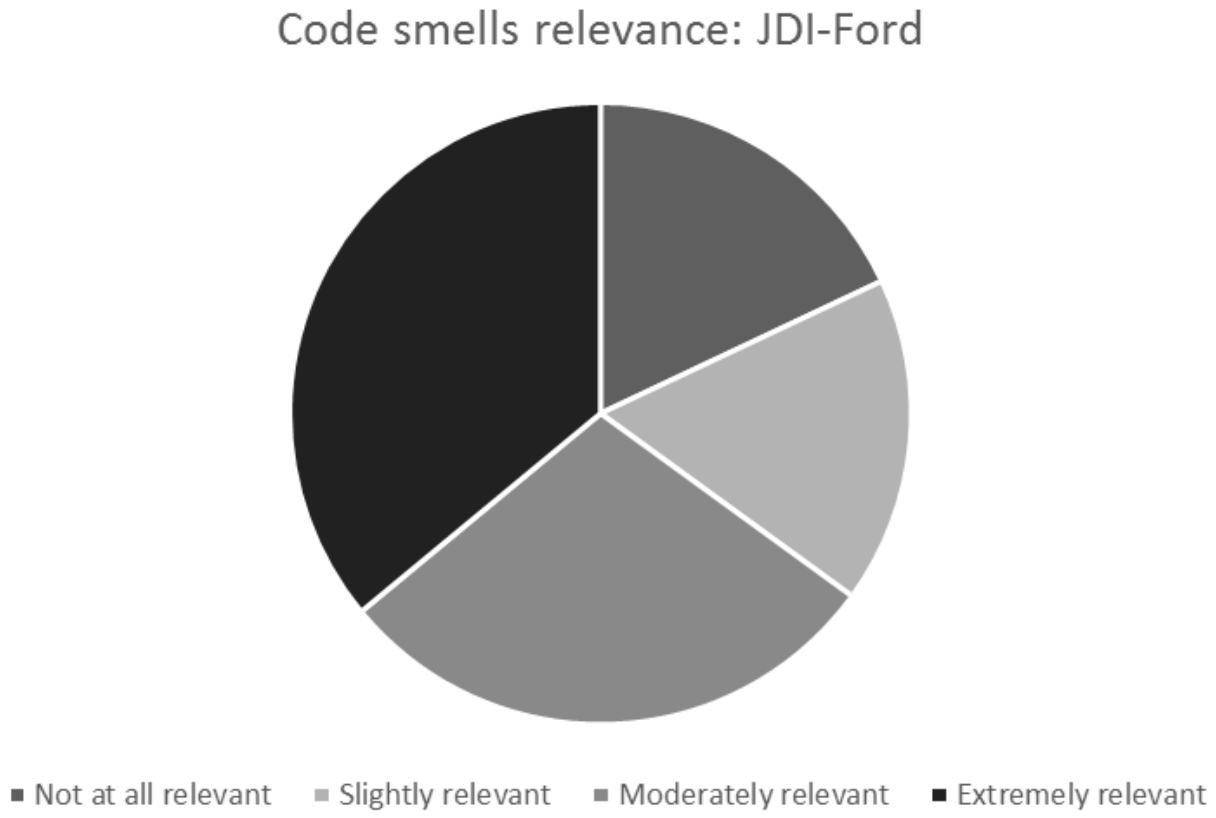
Systems	*Release*	*#Classes*	*#Smells*	*KLOC*
JDI-Ford	v5.8	638	88	247

Quality Gain



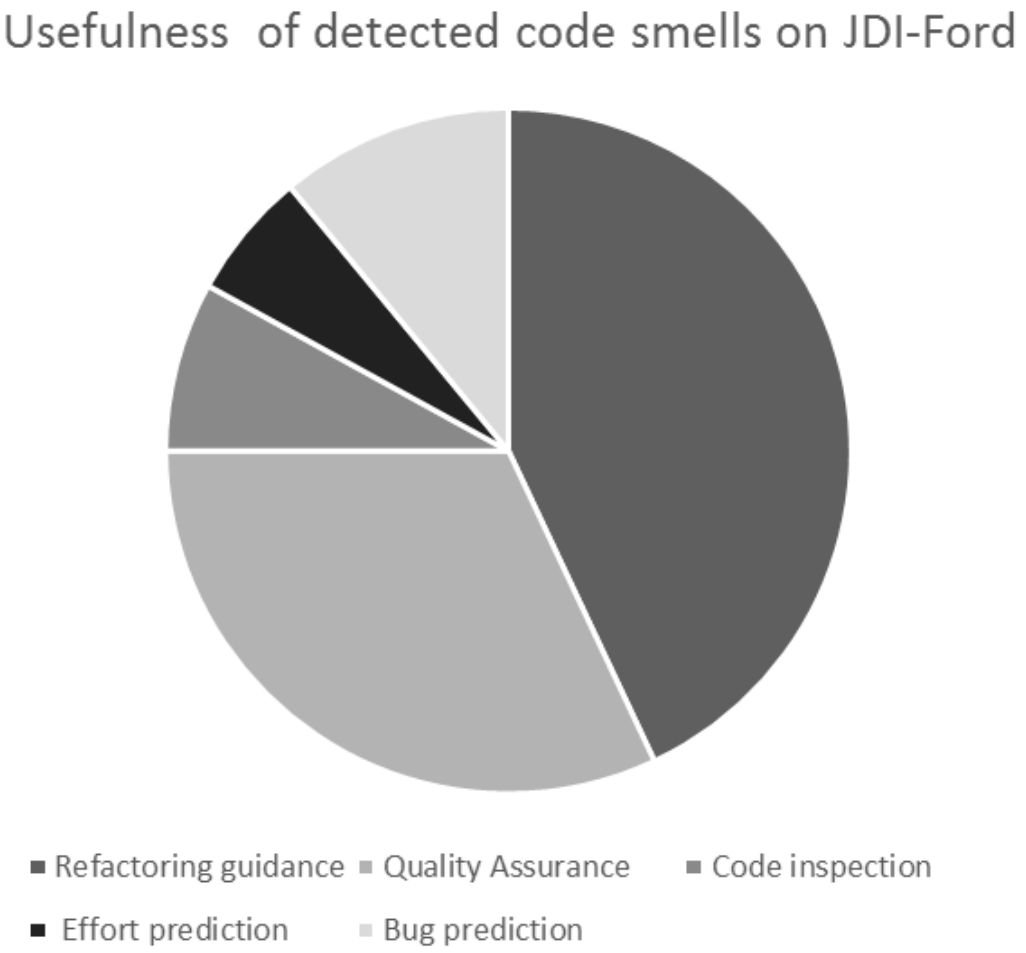
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Code Smells Relevance



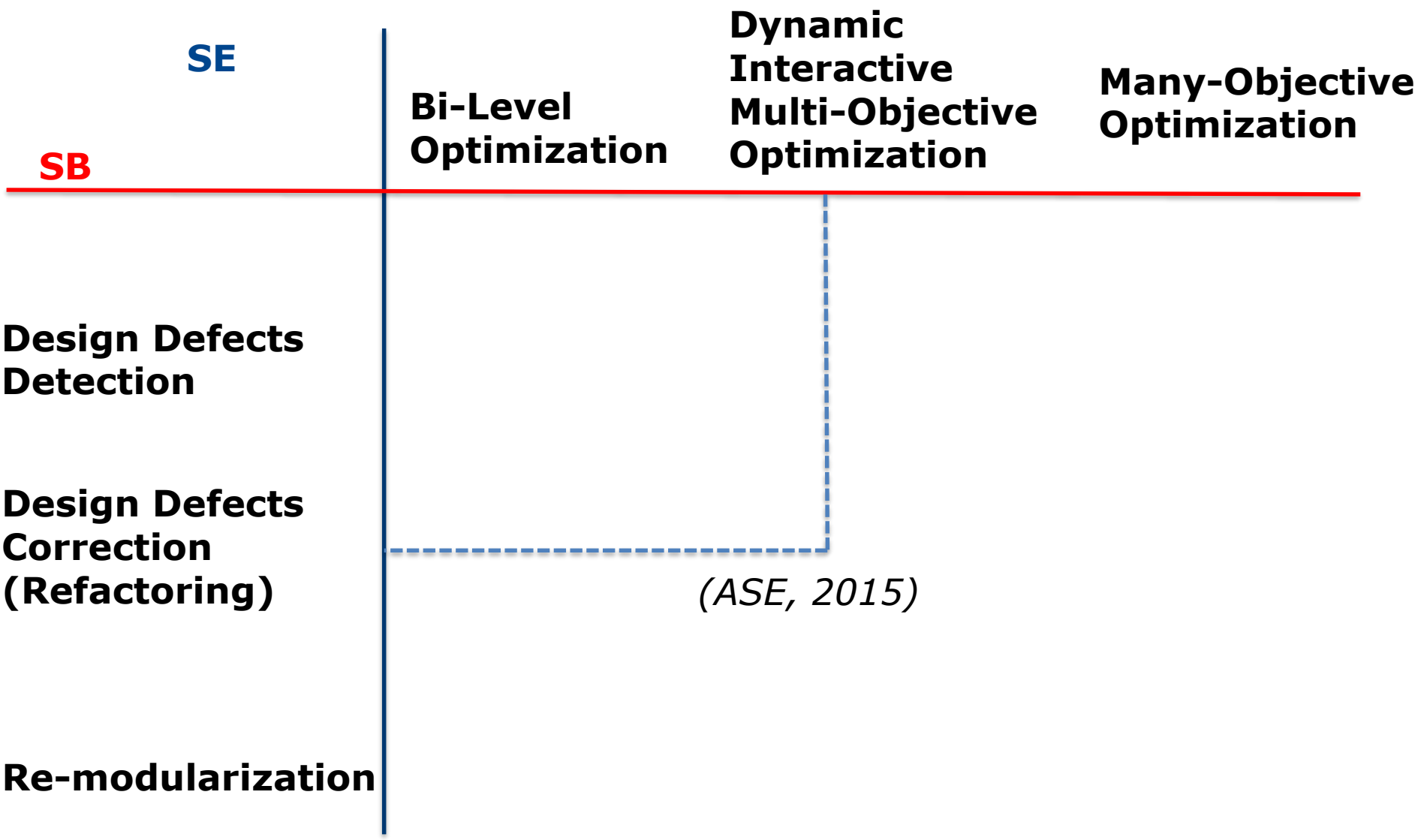
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Usefulness of Code Smells



MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

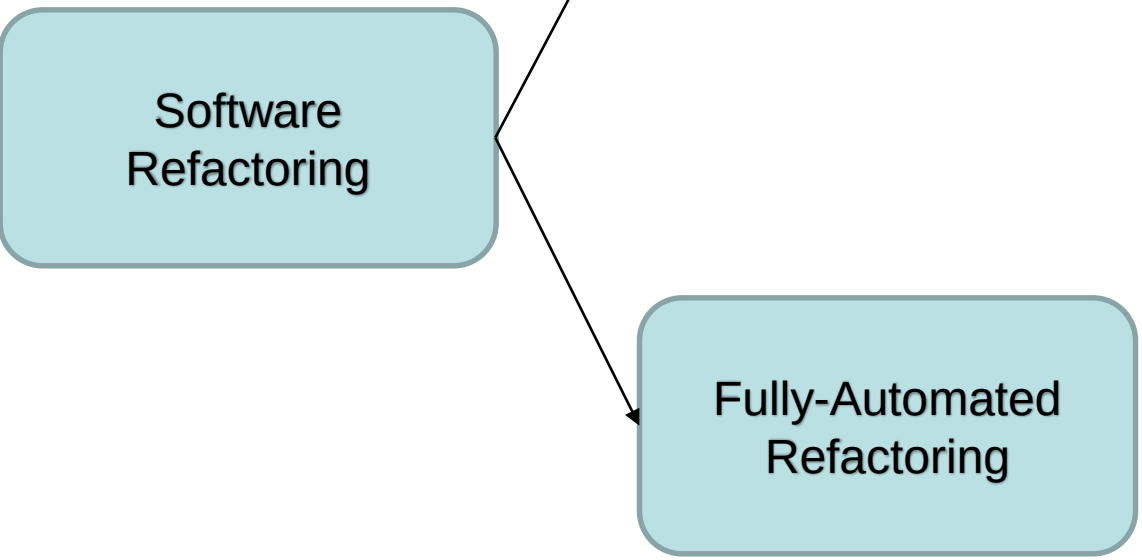
Our Recent Advances in SBSE



Existing Work

- Metric-based approaches
  - Search-based techniques
    - Find the best sequence of refactorings (Harman et al. '07, O'Keeffe et al. '08)
  - Analytic approaches
    - Study of relations between some quality metrics and refactoring changes (Sahraoui et al. '00, Du Bois et al. '04, Moha et al. '08)
- Graph-based approaches
  - Graph transformation
    - Software is represented as a graph
    - Refactorings activities as graph production rules (Kataoka et al, '01, Heckel et al. '95)

Refactoring Challenges



Manual refactoring is

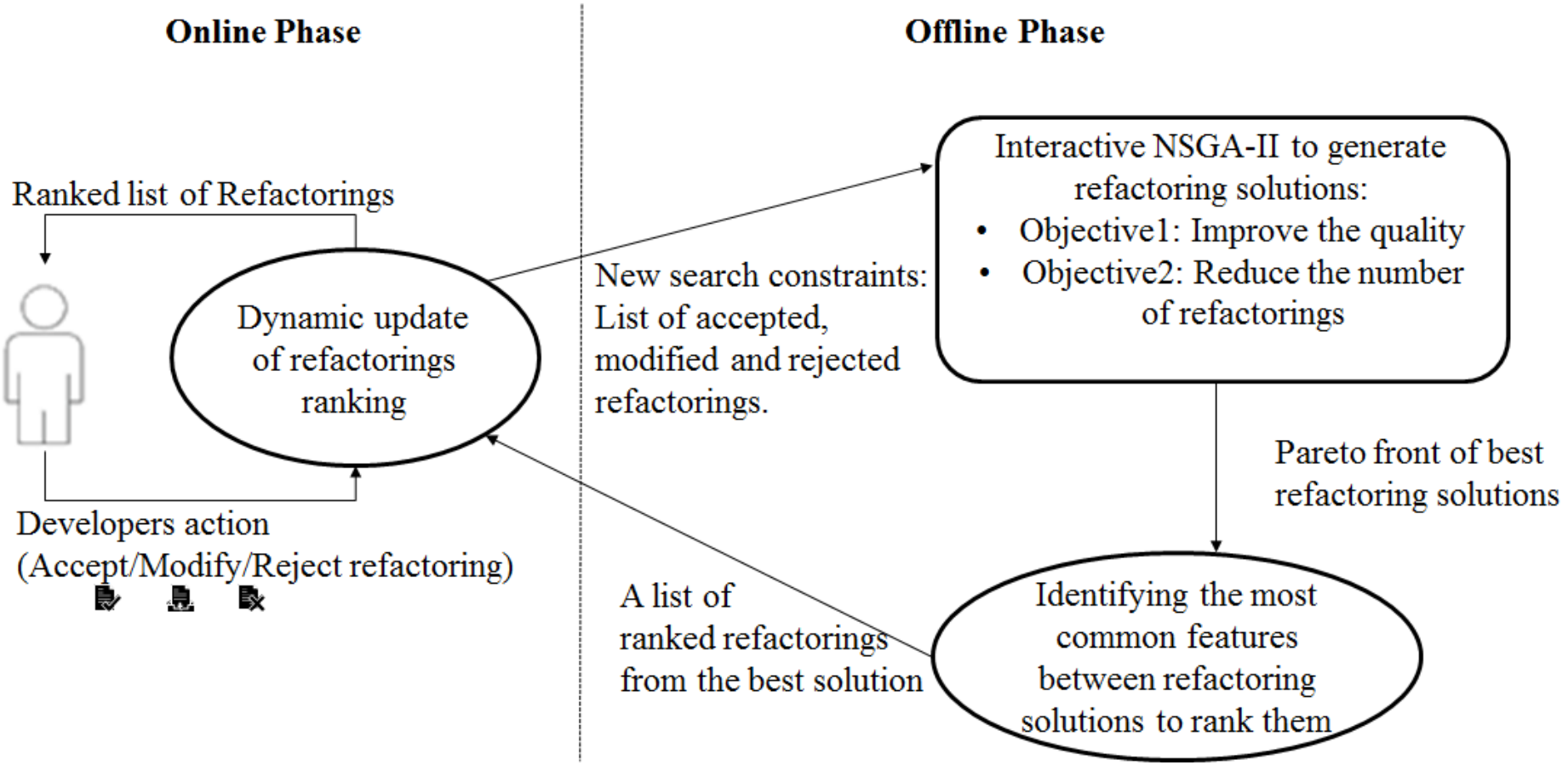
- ***error-prone,***
- ***time consuming,***
- ***not scalable***
- ***not useful for radical refactoring*** **(**extensive application of refactorings to correct unhealthy code.)

Fully-automated refactoring

- ***lacks flexibility*** (developers have to accept the entire refactoring solution),
- ***fails to consider developer perspective and feed-back,***
- ***proposes a long static list of refactorings to be applied but developers do not have enough time to apply all of them***

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

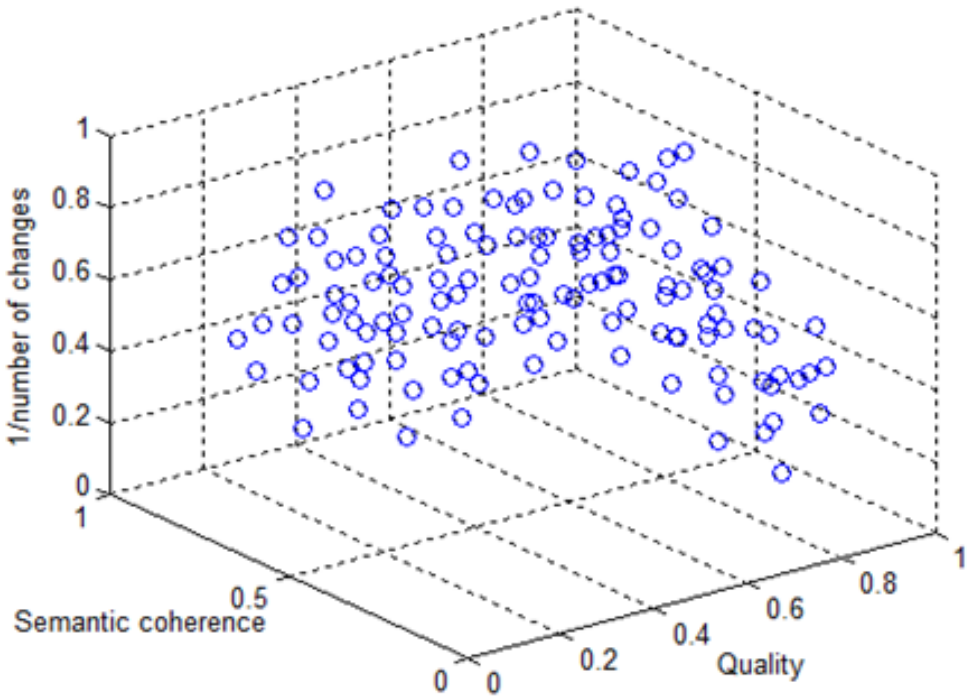
DINAR: Dynamic Interactive Multi-objective refactoring



MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

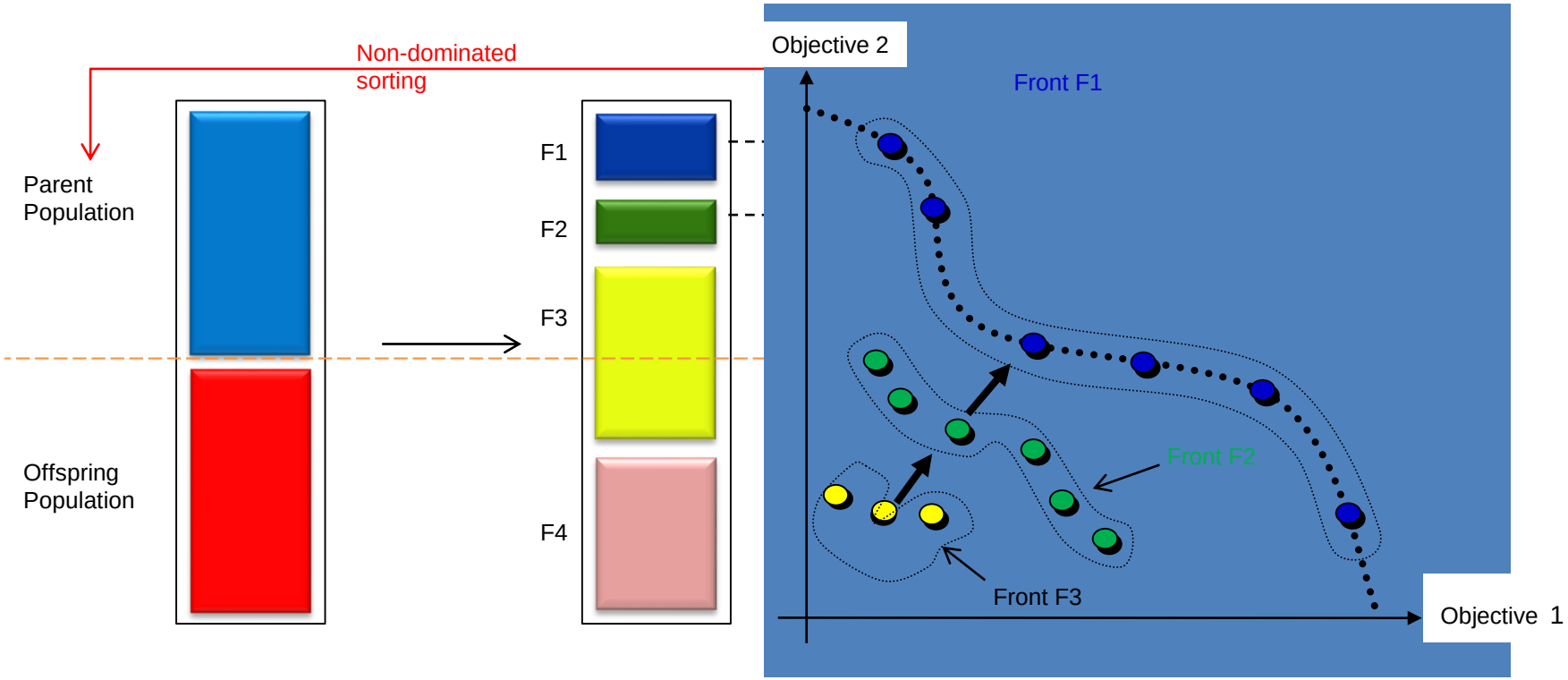
The Three Components of DINAR

1. Upfront generation of refactoring solutions using NSGA-II



NSGA-II Overview

- NSGA-II: Non-dominated Sorting Genetic Algorithm (K. Deb et al., '02)



Representation of Individuals

- Individual = Refactoring solution
- Sequence of refactoring operations

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass
RO8	moveMethod

Representation of Individuals

- Specify the controlling parameters

Refactorings	Controlling parameters
move method	(sourceClass, targetClass, method)
move field	(sourceClass, targetClass, field)
pull up field	(sourceClass, targetClass, field)
pull up method	(sourceClass, targetClass, method)
push down field	(sourceClass, targetClass, field)
push down method	(sourceClass, targetClass, method)
inline class	(sourceClass, targetClass)
extract class	(sourceClass, newClass)

Population of Solutions

- Population: set of refactoring solutions

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass
RO8	pullUpAttribute
RO9	extractClass
RO10	moveMethod

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass
RO8	moveMethod

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass
RO8	moveMethod

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass
RO8	moveMethod

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	extractClass
RO8	moveMethod

RO1	moveMethod
RO2	pullUpAttribute
RO3	extractClass
RO4	inlineClass
RO5	extractSuperClass
RO6	inlineMethod
RO7	inlineClass
RO8	inlineMethod
RO9	extractSuperClass

The Three Components of DINAR

2.a Ranking of refectoring solutions

- Counts the number of occurrence of the refactoring operation $R_{i,j}$ among all the Pareto front solutions.
- Search for common principles among the refactoring solutions.

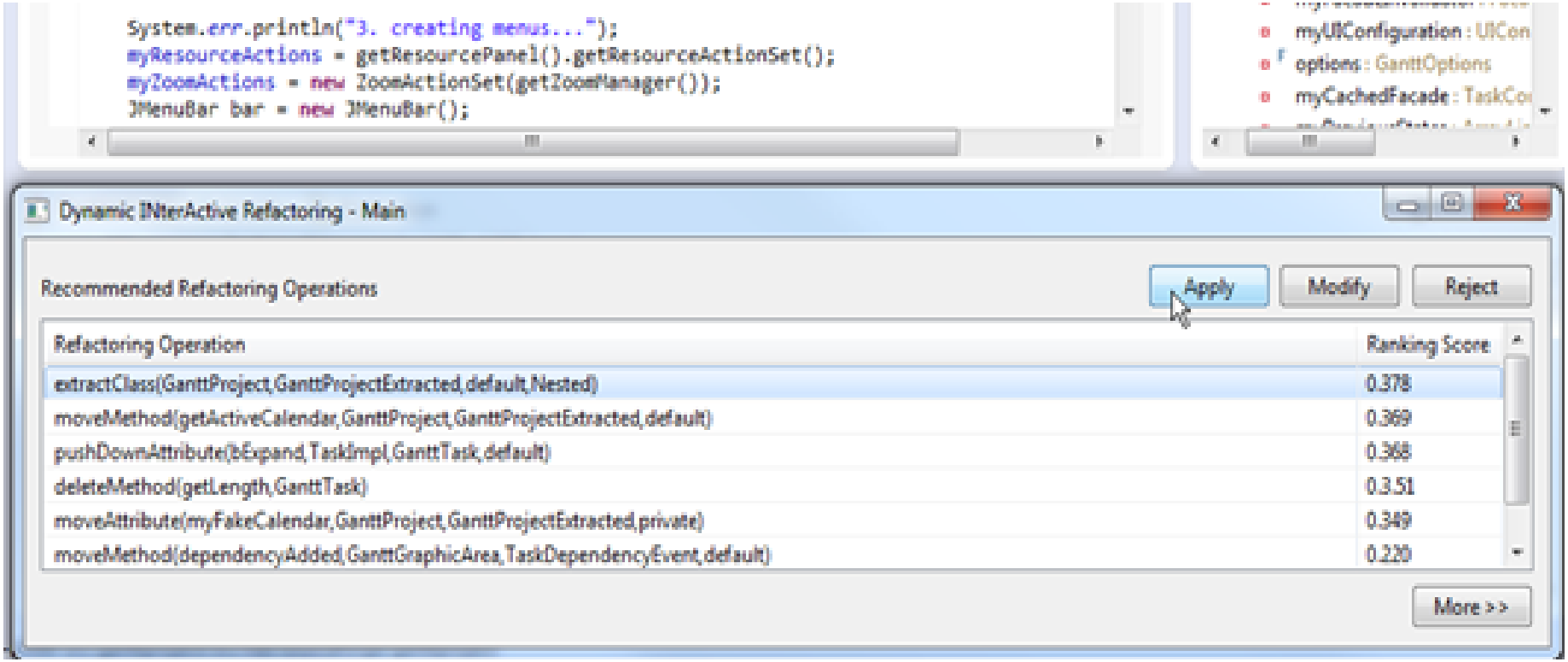
$$Rank(R_{i,j}) = \frac{number\_occurence}{\max\_number\_occurence} + \frac{\sum_{k=1}^{i} Sim(R_{k,j}, \text{recommended\_ref})}{\#recommended\_ref}$$

- The ranking of refactorings is updated automatically after every feed-back from the developer.

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

The Three Components of DINAR

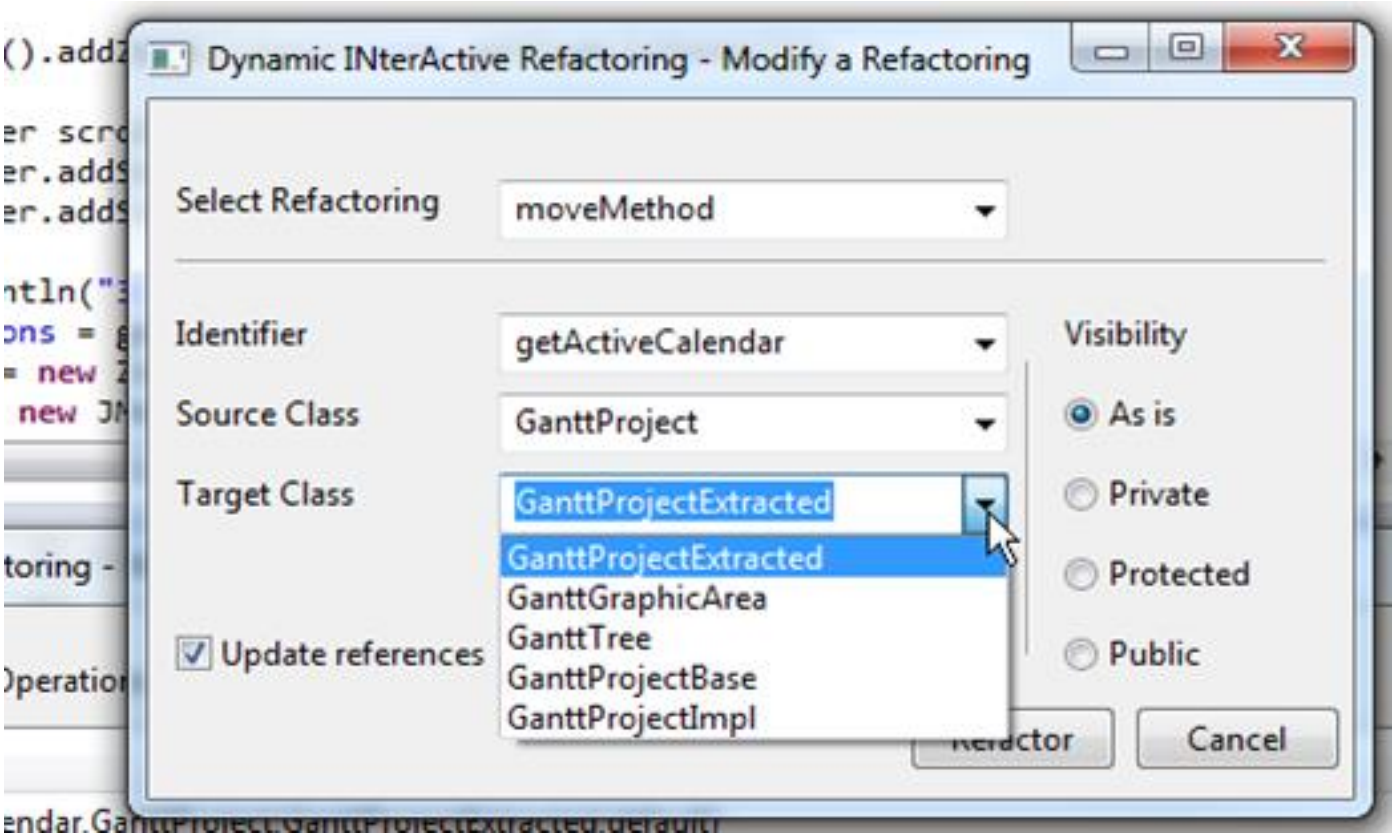
2.b Interactive recommendation of refactorings



The list of ranked Refactorings recommended by DINAR

The Three Components of DINAR

2.b Interactive recommendation of refactorings

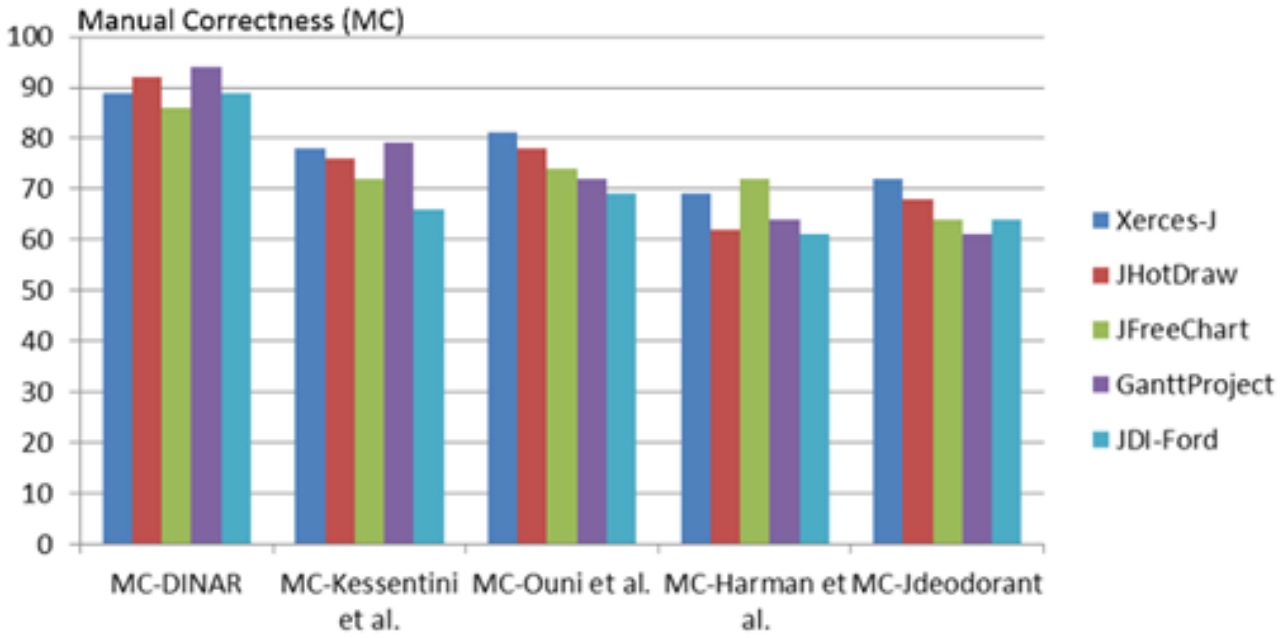


The user can modify the suggested refactoring

The Three Components of DINAR

3. Dynamic update of recommended refactorings
   - The input of this component is the new system after major changes are performed by developers on the original one and the latest set of good refactoring solutions.
   - The output is a new updated set of non-dominated refactoring solutions that are adapted to the new system using an indicator-based local search

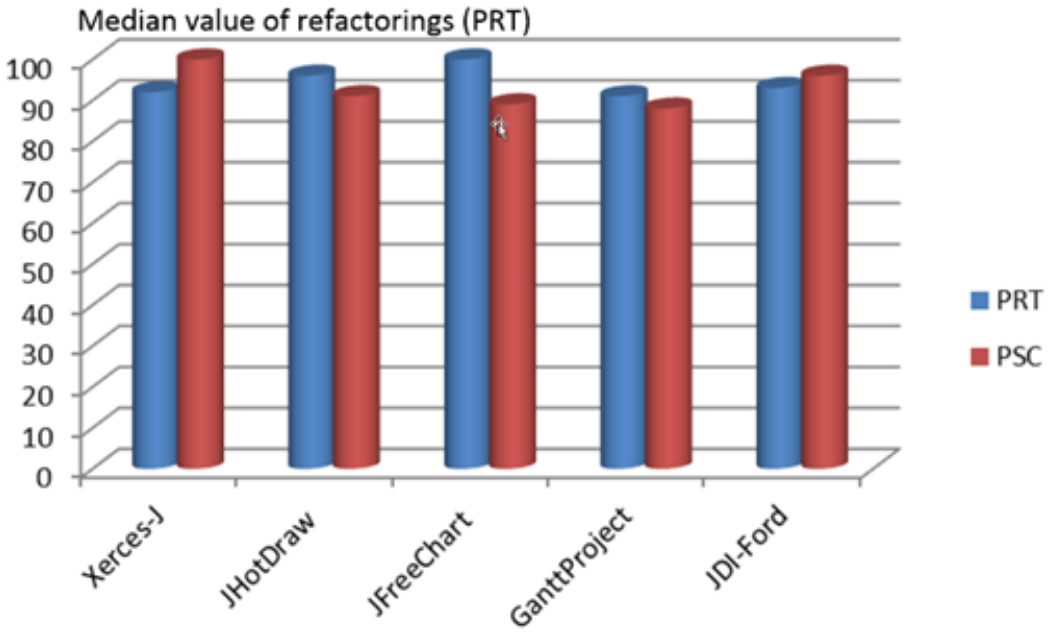
Results: Manual Correctness



Median manual correctness (MC) value over 31 runs on all the five systems using the different refactoring techniques with a 99% confidence level ($\alpha < 1\%$).

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

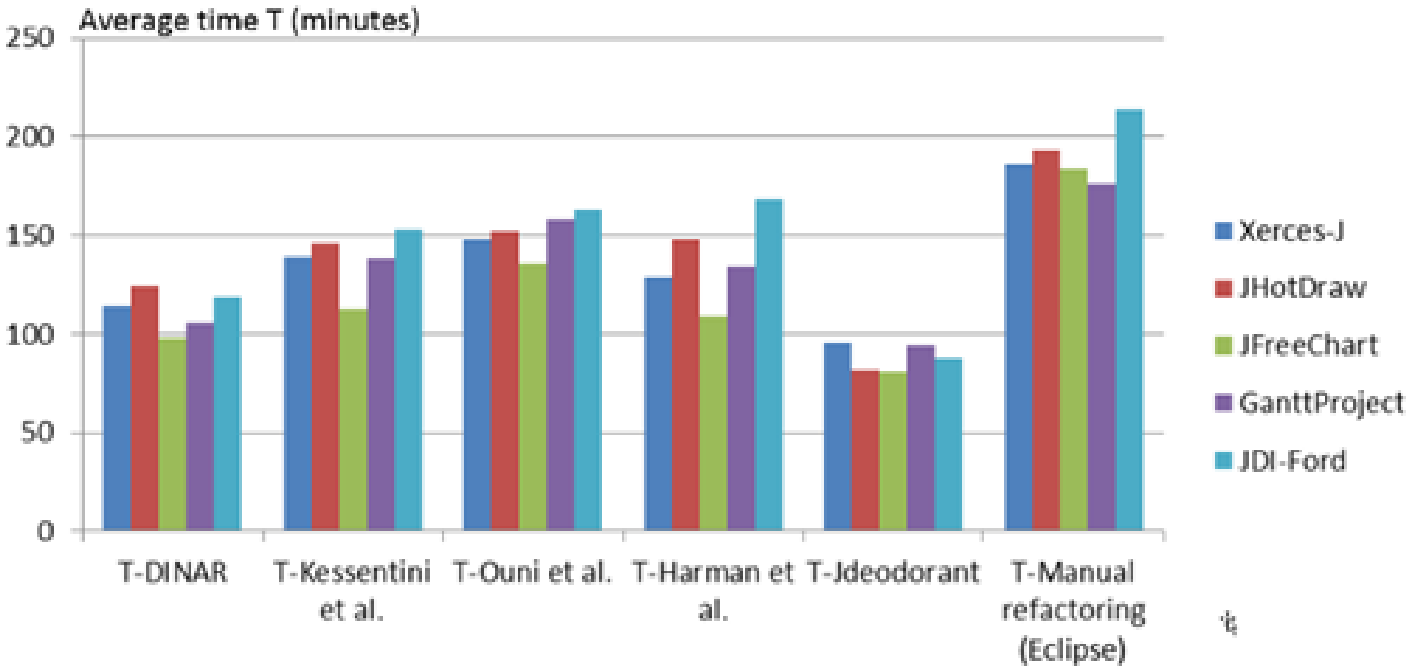
Results: Refactoring Recommendation



Median value of refactorings (PRT) and code elements selected from the top5 on all the five systems.

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Results: Refactoring Recommendation



Average time T (minutes) required by developers to finalize a refactoring session.

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

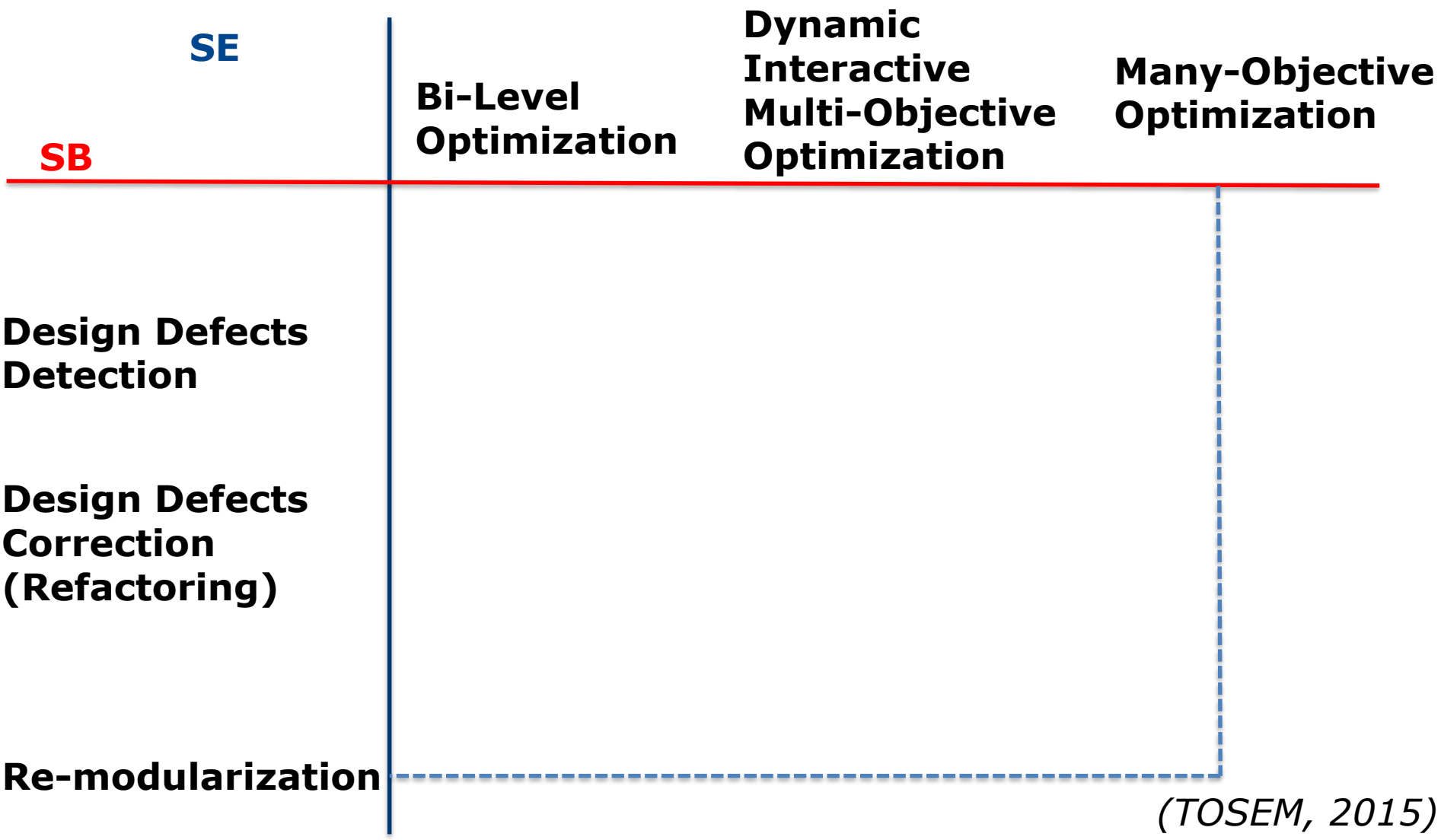
Results: Feed-back from Developers

- The participants agreed that
  - the interactive dynamic refactoring recommendations are a desirable feature in IDEs
  - the interactive manner of recommending refactorings by DINAR is a useful and flexible way to refactor systems comparing to fully-automated or manual refactorings

- The participants found DINAR helpful
  - for both **floss refactoring** and **root canal refactoring**
  - to modify the source code (to add new functionality) while doing refactoring
  - to take the advantages of using multi-objective optimization for software refactoring without the explicit exploration of the set of non-dominated solutions

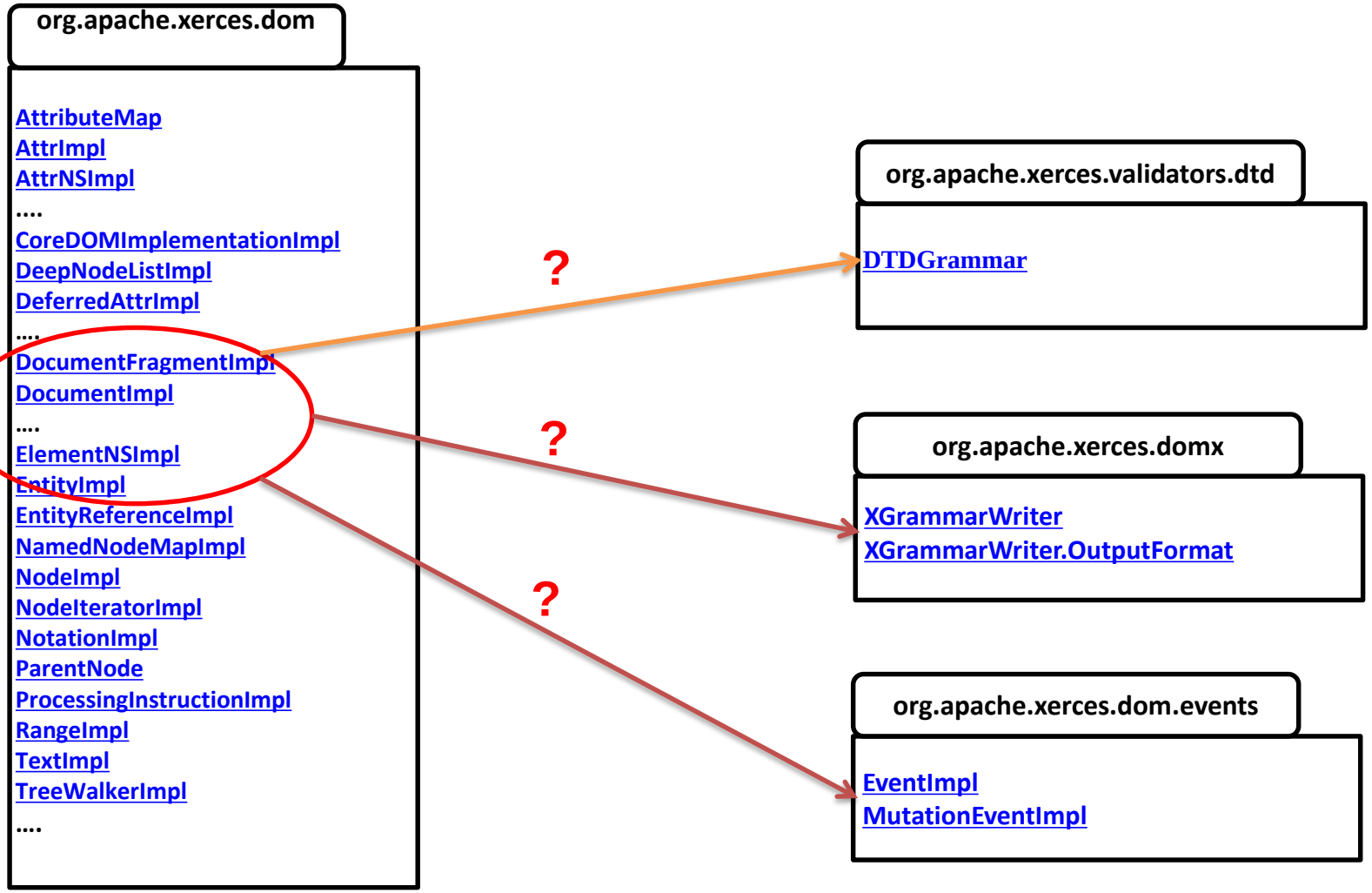
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Our Recent Advances in SBSE

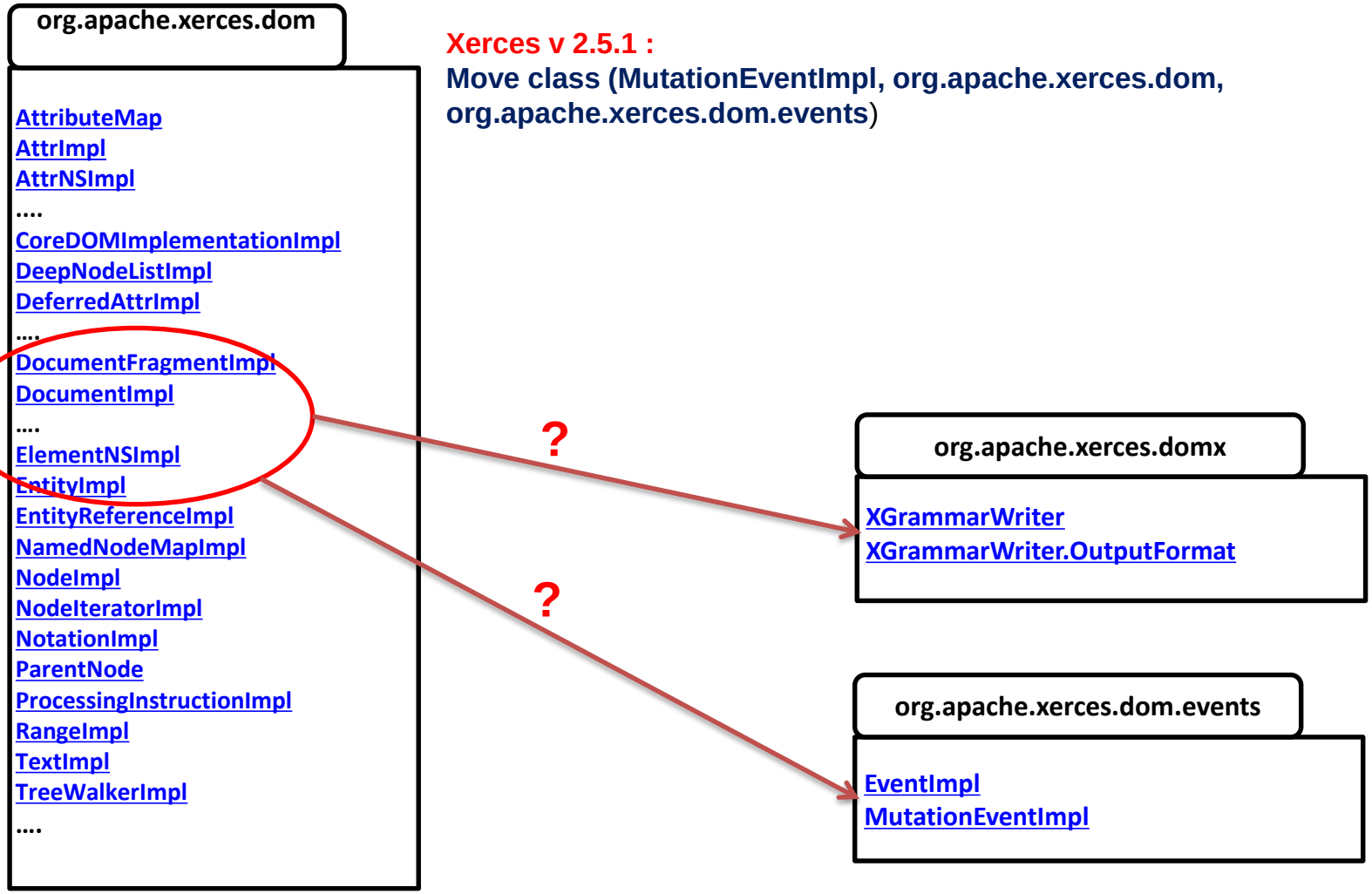


(TOSEM, 2015)

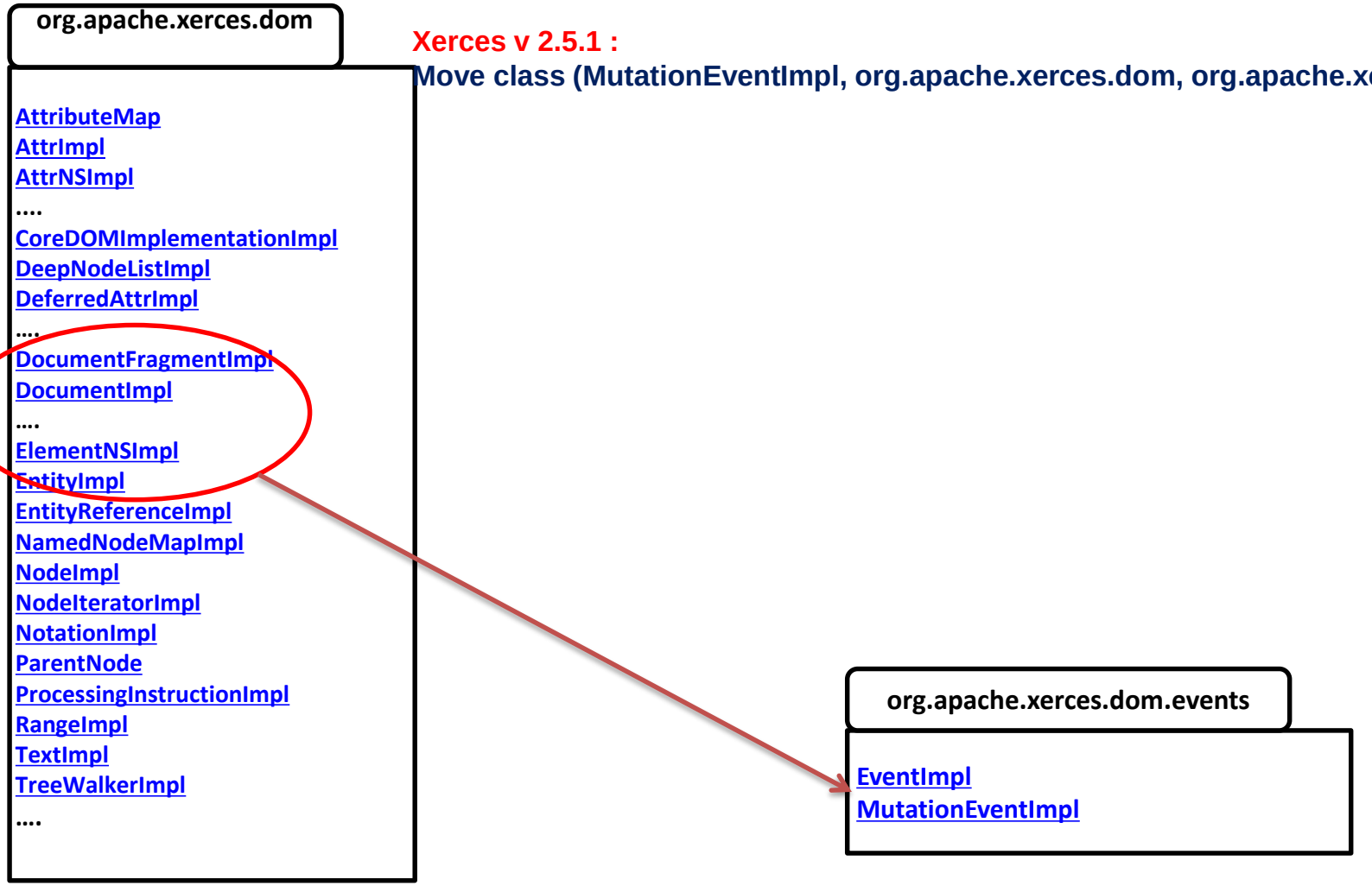
Motivating Example



Motivating Example

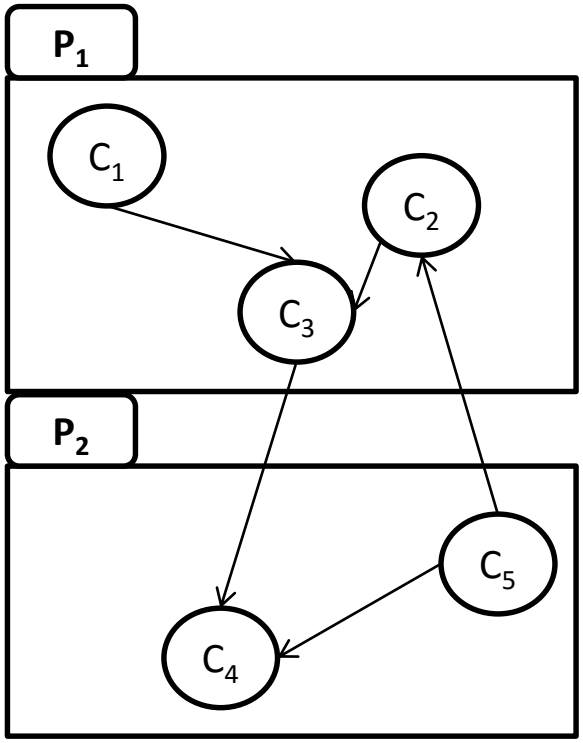


Motivating Example



Remodularization

- Software remodularization consists of automatically reorganizing software packages to improve the overall program structure



Cohesion: number of intra-edges

Coupling: number of inter-edges

Related Work

- Bavota *et al* : Software Re-Modularization based on Structural and Semantic Metrics, 2010
  - Proposed an automated mono-objective where semantic and structural metric are combined in one objective value.

- Bavota *et al*: Putting the Developer in-the-Loop: An Interactive GA for Software Remodularization,2012.
  - Propose an extension of its work with a Mono and Multi- Objective using an Interactive GA where the developer give their feedback to proposed remodularization solution.

Related Work

- Harman *et al*: Software Module Clustering as a Multi-Objective Search Problem,2011.
  - Use genetic algorithm with three objectives: Coupling, Cohesion and Complexity.
- Abdeen *et al*: Towards automatically improving package structure while respecting original design decisions, 2013.
  - Proposed a re-modularization task as a multi-objective optimization problem to improve existing packages structure while minimizing the modification amount on the original design.

Limitations

- Focus only on improving structural measures (cohesion, coupling, etc.)
- Violate the domain semantics
- Do not consider the number of code-changes (deviation from initial design) and development/maintenance history.
- Limited to only 2 types of changes
  - Move class
  - Split package

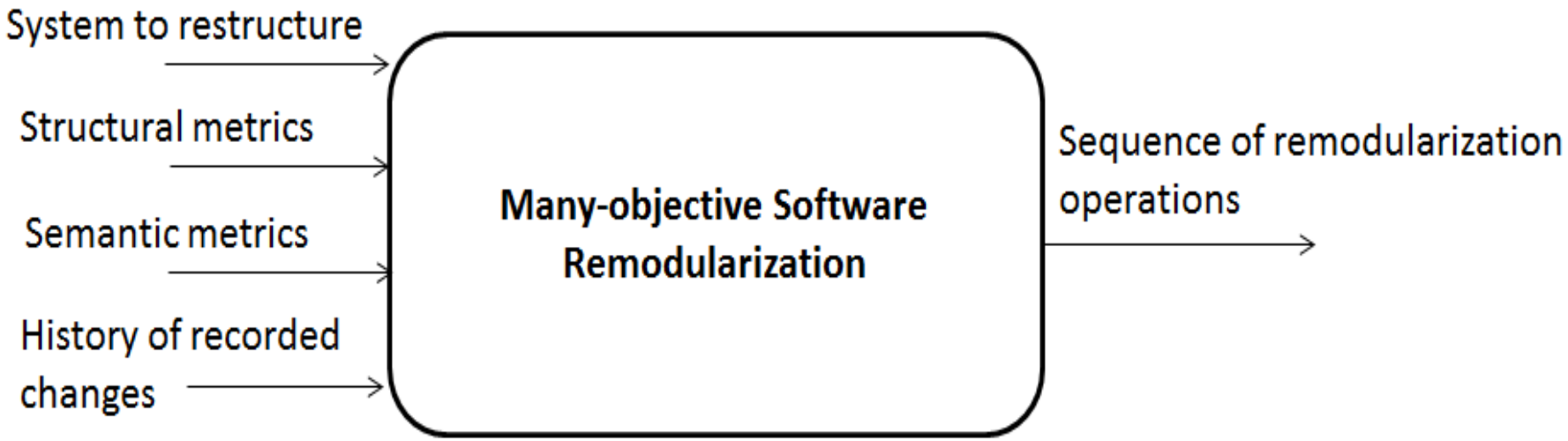
Proposal

- Software remodularization as a **many-objective search problem**
  - 4 Structural measures (number of packages, number of classes per package, cohesion and coupling)
  - Semantic coherence (cosine similarity and Call graphs)
  - Number of operations per solution
  - Consistency with the history of changes

- New operation types
  - Move method
  - Extract class
  - Merge packages
  - Move class
  - Extract package

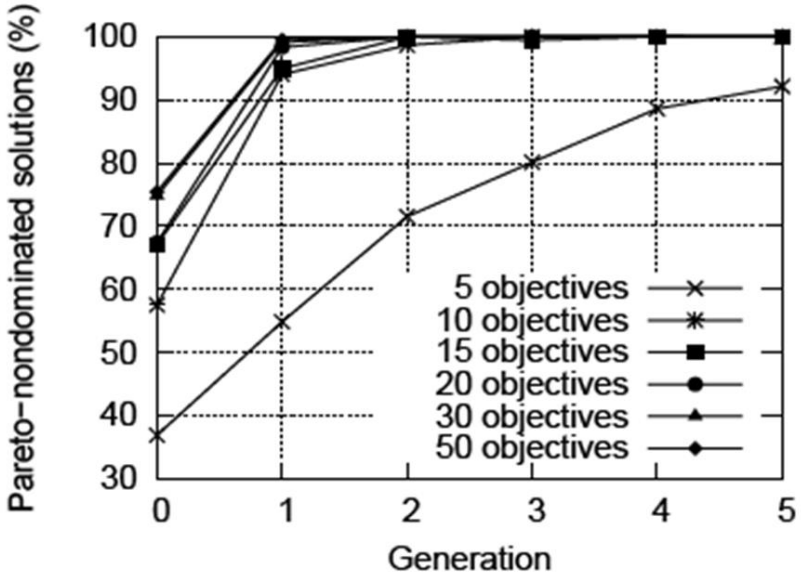
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Approach Overview



Multi-Objective issues

	Points
2	50
4	62 500
5	1 953 125
7	1 708 984 375

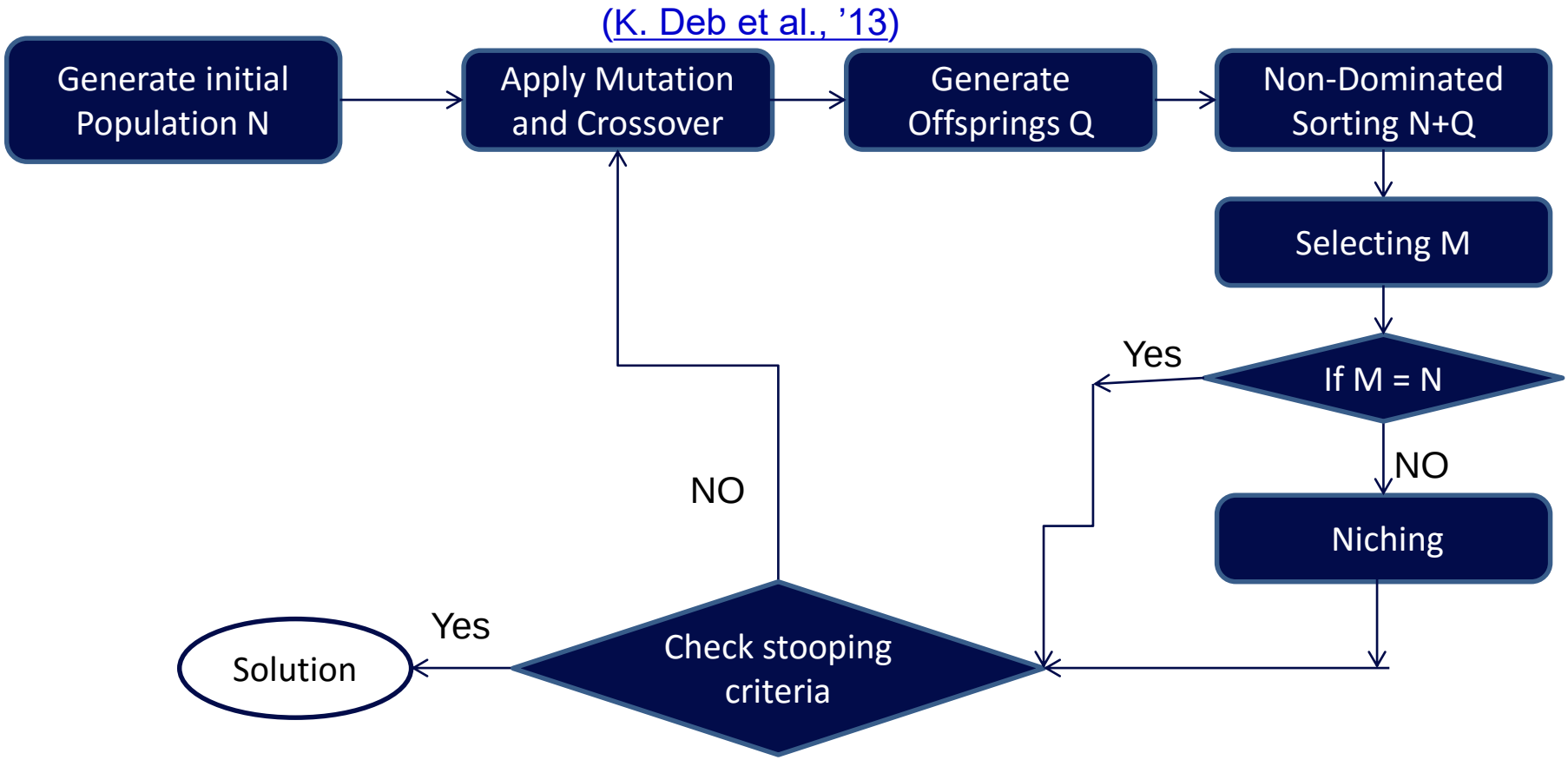


➔ **The algorithm becomes unable to distinguish between solutions**

➔ **Random search behavior**

➔ **Require an additional selection process for convergence.**

Multi-Objective issues



MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Solution Representation

- Each solution is represented as a vector of multiple refactorings
- Each refactoring is generated randomly.

List of possible operations

1	Create Package(p)
2	Merge Package(p1,p2)
3	Split Package(p3, p4)
4	Move Class(C1, P1)
5	Merge Package(p5,p6)

Example of a remodularization solution

Move Class(AttrNSImpl, org.apache.xerces.dom, org.apache.xerces.validators.dtd)
Extract Class(XGrammarWriter, XGrammarInput, parseInt())
Move Method(normalize(), XGrammarWriter, DTDGrammar)
Extract Package(org.apache.xerces.dom, org.apache.xerces.dtl, CharacterDataImpl, ChildNode)

Fitness Functions

- Structural Metrics:
  1. *number of classes per package*
  2. *number of packages in the system*
  3. *Coupling*
  4. Cohesion

Fitness Functions

- Semantic Metrics:

5. Vocabulary-based similarity:

$$Sim(c1,c2) = \cos(\vec{c1},\vec{c2}) = \frac{\vec{c1}.\vec{c2}}{\|\vec{c1}\| * \|\vec{c2}\|} \in [0,1]$$

6. Dependency-based similarity:

$$\text{sharedCallOut}(c1, c2) = \frac{|\text{callOut}(c1) \cap \text{callOut}(c2)|}{|\text{callOut}(c1) \cup \text{callOut}(c2)|} \in [0,1]$$

$$\text{sharedCallIn}(c1, c2) = \frac{|\text{callIn}(c1) \cap \text{callIn}(c2)|}{|\text{callIn}(c1) \cup \text{callIn}(c2)|} \in [0,1]$$

Fitness Functions

- Other Metrics:

7. Number of code changes : Sum of Operations

8. Similarity with history of code changes

$$Sim\_history(RO) = \sum_{j=1}^{n} w_j$$

Experiments

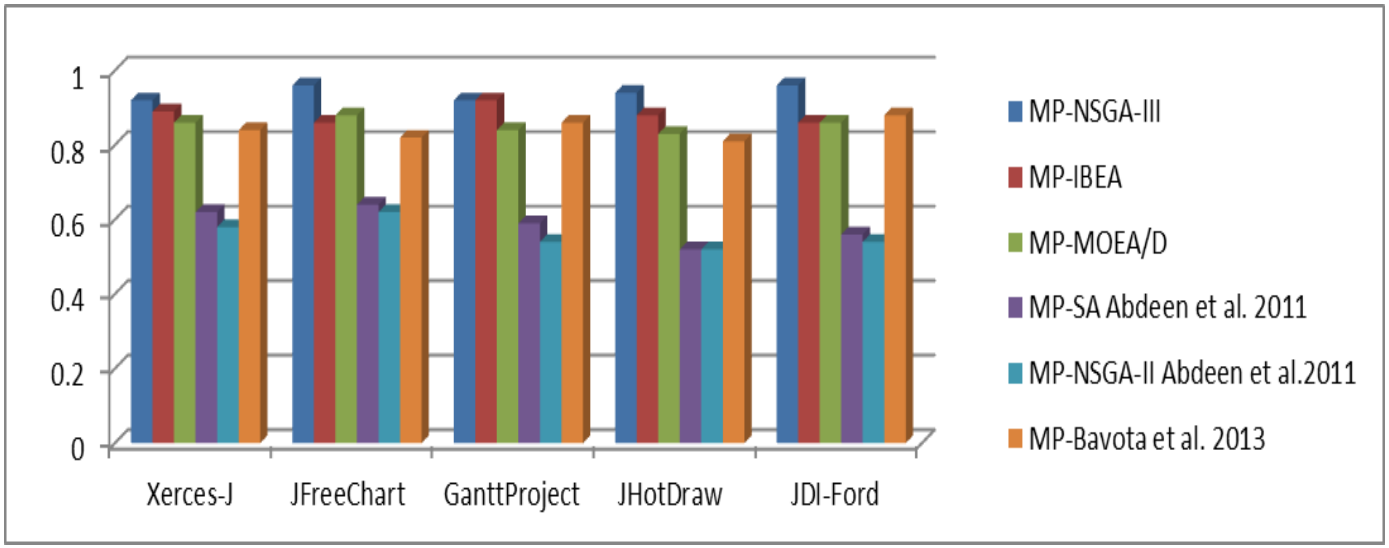
- 4 medium and large scale open systems and 1 industrial system provided by Ford Motor Company.
- Each experiment is repeated 32 times.

Systems	Release	# classes	KLOC
Xerces-J	v2.7.0	991	240
JHotDraw	v6.1	585	21
JFreeChart	v1.0.9	521	170
GanttProject	v1.10.2	245	41
JDI-Ford	v5.8	638	247

Results

- Manual Precision:

$$MP = \frac{\#coherent\ operations}{\#proposed\ operations} \in [0,1]$$

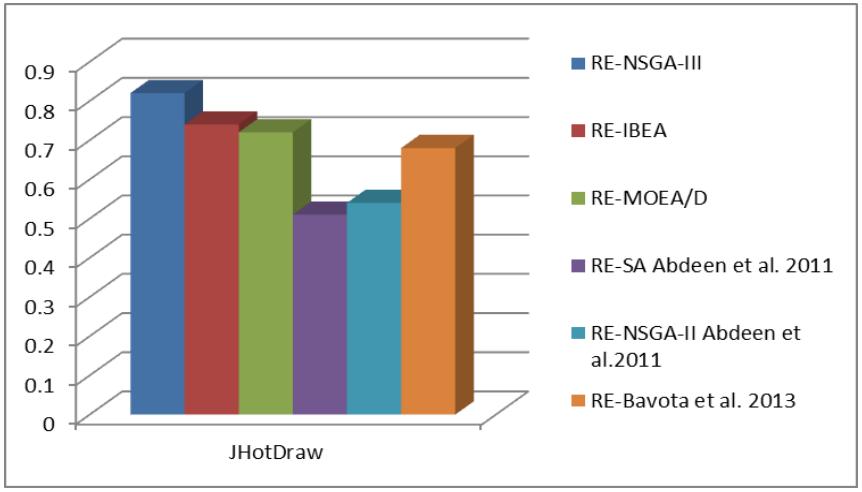


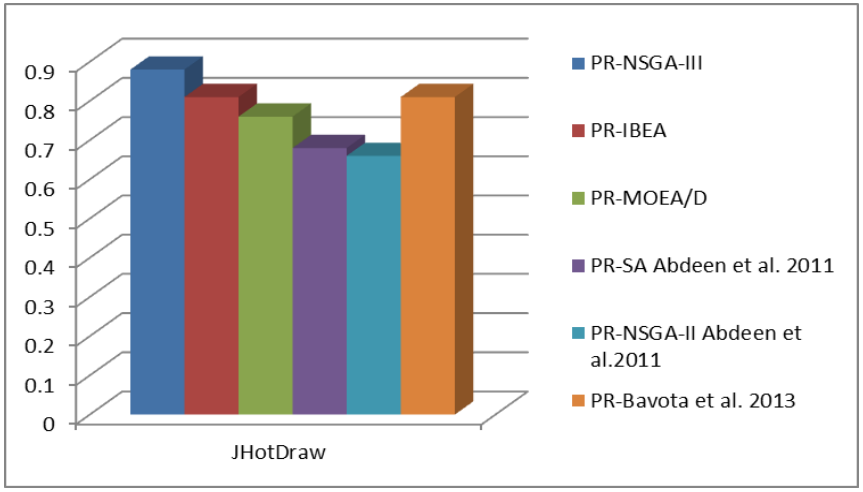
Results

- Automatic Validation

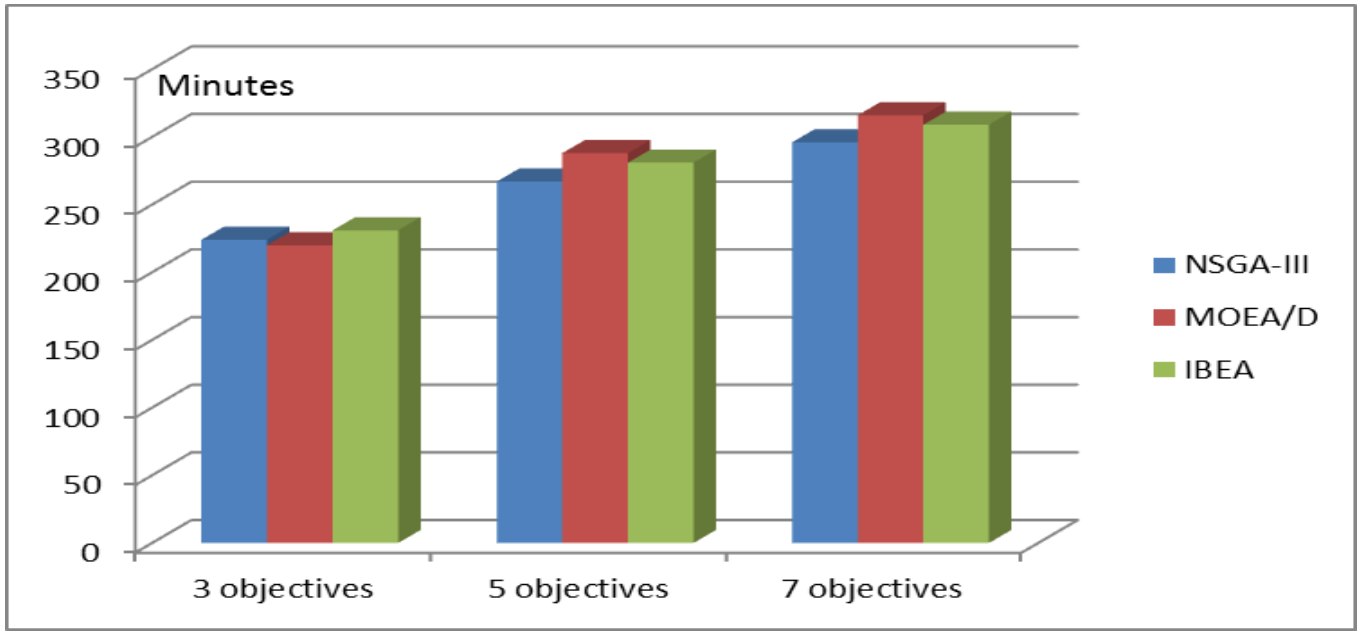
$$RE_{recall} = \frac{|\text{suggested operations}| \cap |\text{expected operations}|}{|\text{expected operations}|} \in [0,1]$$

$$PR_{precesion} = \frac{|\text{suggested operations}| \cap |\text{expected operations}|}{|\text{suggested operations}|} \in [0,1]$$



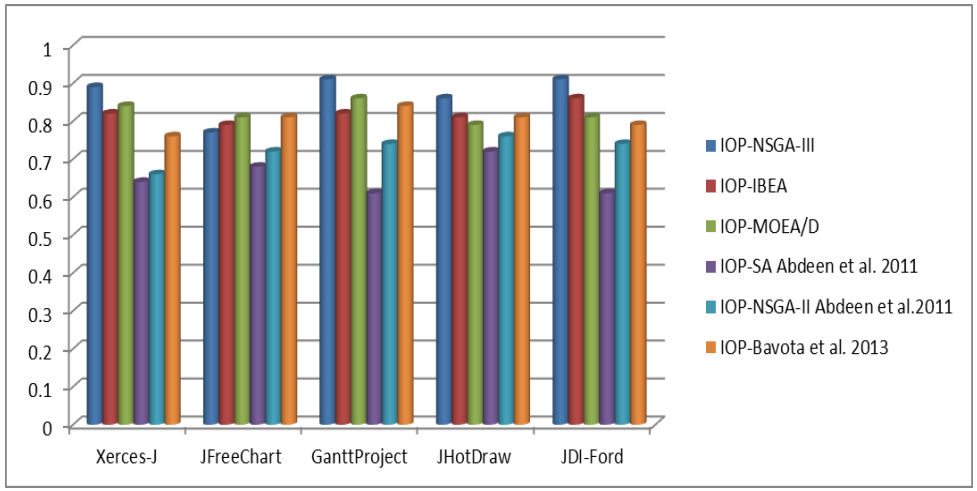


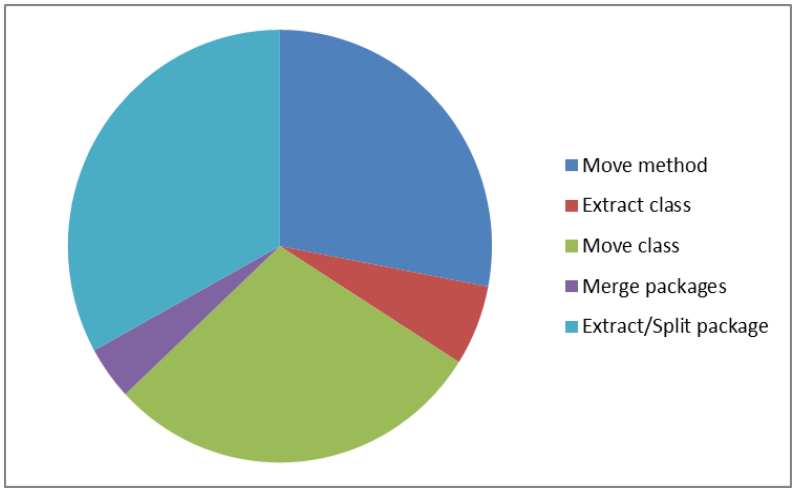
Results


Minutes
350
300
250
200
150
100
50
0
3 objectives
5 objectives
7 objectives
NSGA-III
MOEA/D
IBEA

Results

- How can our approach be useful for software engineers in real-world setting?





MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Challenges and Open Research Directions

- *Why do we currently need to **design special algorithms for each software engineering problem** instance?*
  - This is unrealistic: Science is about generality. Several software engineering activities have a lot of common patterns and similarities

- *Why do we currently **address silos of software engineering activity**?*
  - This is unrealistic: engineering decision making needs to take account of requirements, designs, test cases and implementation details *simultaneously*.

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN

Challenges and Open Research Directions

- ***Automation level:*** *How best do we draw the dividing line between adaptive automation for small changes and human intervention to invoke more fundamental adaption and to provide oversight and decision making?*

- **Surrogate metrics:** Any approach that seeks dynamic adaptivity must necessarily compute many fitness evaluations between adaptations surrogate fitness computation will need to be fast.

- **Dynamic Adaptativity**

Take a Problem and "SBSE" it !

Thank You

Questions?