Identification and Selection of Refactorings for Improving Maintainability of Object-Oriented Software

Ah-Rim Han

Dept. of Computer Science
KAIST

2013. 5. 8

Contents

- Introduction
- Main Approach
  - Refactoring Candidate Identification
    - Extracting with Dynamic Information Based Rules
    - RER-aware Grouping Entities into Maximal Independent Sets (MISs)
  - Refactoring Selection
    - Selecting Multiple Elementary Refactorings
- Evaluation
- Related Work
- Conclusion and Future Work

Introduction

Software Changes and Need of Refactoring

- Object-oriented software undergoes continuous changes with various maintenance activities
  - Ex) addition of new functionalities and correction of bugs

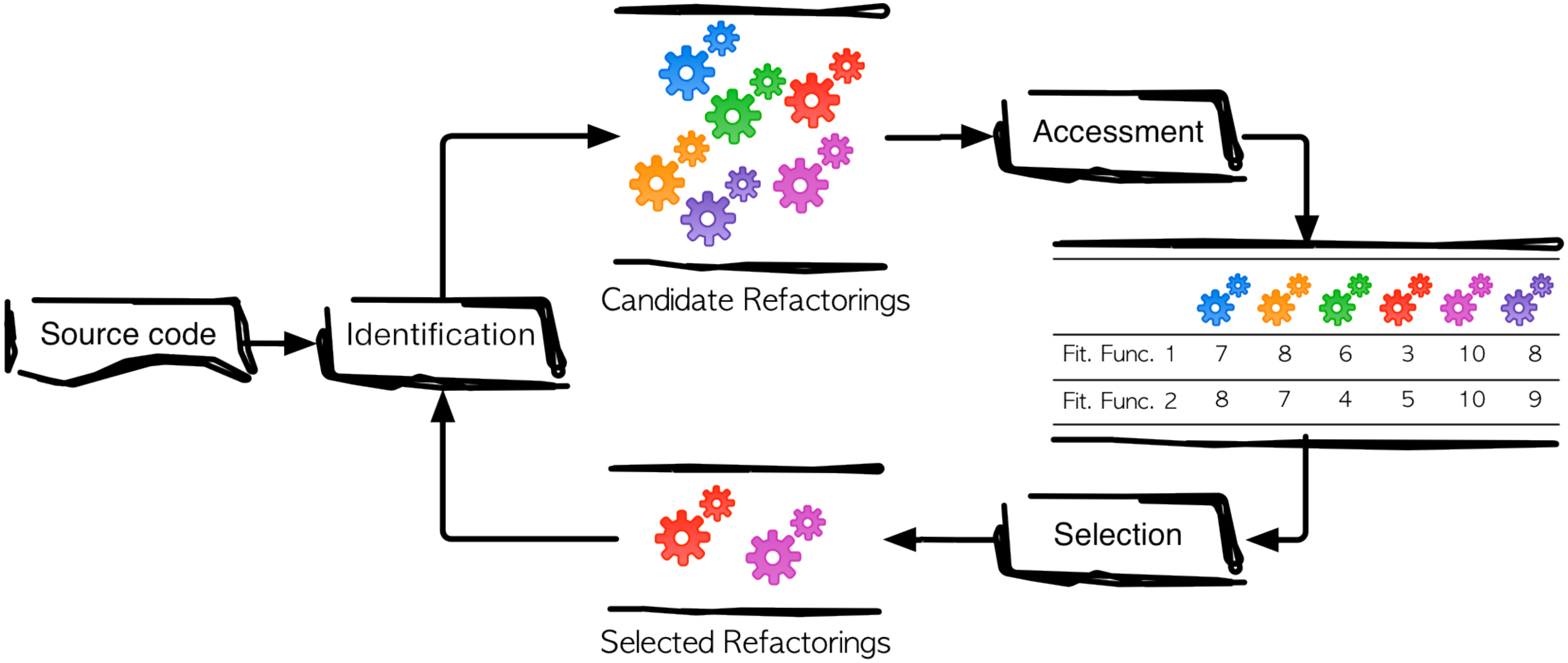
- Since the changes often take place without consideration of the design rationale due to time constraints
  - The design quality of the software may degrade overtime

> *"Refactoring can serve to restructure the design of object-oriented software without altering its external behavior to improve maintainability" [Fowler'1999]*

→ In this thesis, by refactoring, we aim to make software for accommodating changes more easily

Systematic Refactoring Identification Process

- Activities for systematic refactoring identification process



Fit. Func. 1	7	8	6	3	10	8
Fit. Func. 2	8	7	4	5	10	9

Motivation and Research Goal (1/2)

- Refactoring identification using only static information (captured by static source code analysis)
  - Refactorings candidates may be suggested on the pieces of code
    - Never used and never changes having occurred

→ **When establishing refactoring candidate extraction rules, we use dynamic information**

- **Motivated by the previous study [Han'2010] that the data capturing how the system is utilized (i.e., dynamic information) is an important factor for estimating changes**
- **Investing efforts on the refactorings involving such codes may effectively reduce maintenance cost**

Motivation and Research Goal (2/2)

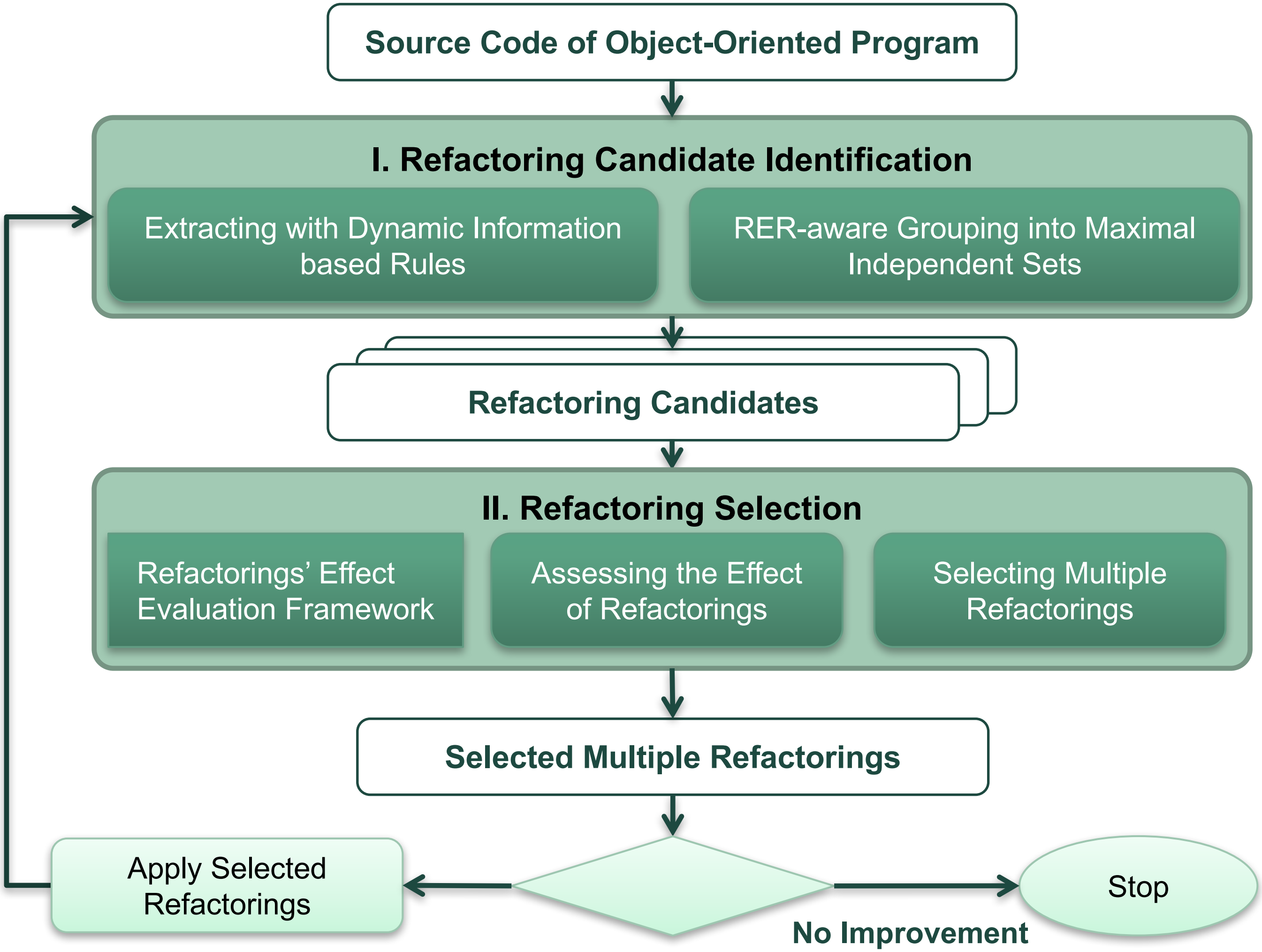
- Determining refactoring sequences to be applied
  - The best refactoring selection in a greedy way
    - ***Inefficient*** to select just one best refactoring for the iteration of refactoring identification process

→ **For each iteration of refactoring identification process, we select the group of elementary refactorings (multiple refactorings) that can be applied at a same time**

- **When grouping elementary refactorings, we consider refactorings' effect relevance (RER) on maintainability**

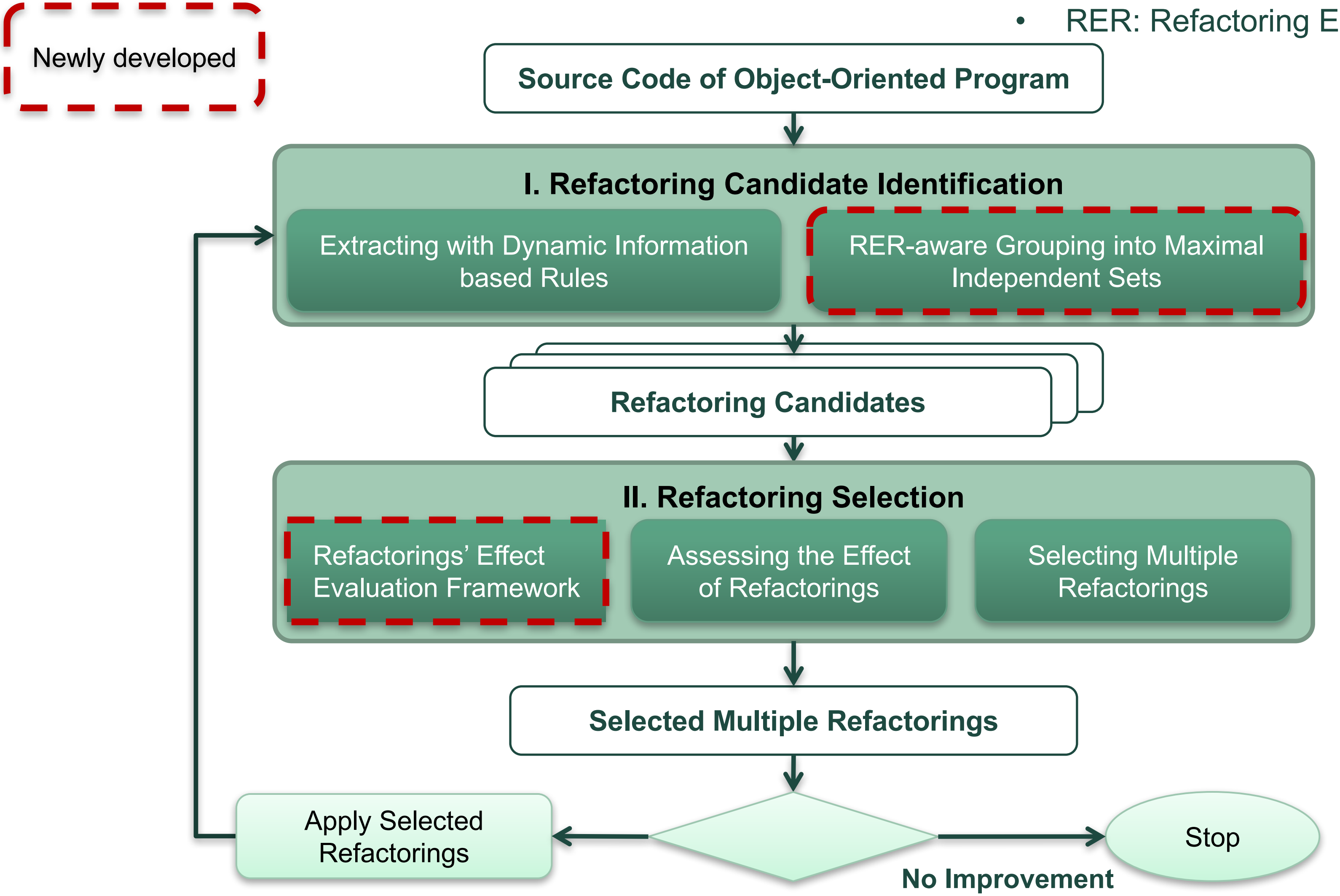
Thesis Overview

- RER: Refactoring Effect Relevance


Source Code of Object-Oriented Program
I. Refactoring Candidate Identification
Extracting with Dynamic Information based Rules
RER-aware Grouping into Maximal Independent Sets
Refactoring Candidates
II. Refactoring Selection
Refactorings' Effect Evaluation Framework
Assessing the Effect of Refactorings
Selecting Multiple Refactorings
Selected Multiple Refactorings
Apply Selected Refactorings
Stop
No Improvement

What Have Been Improved from Proposal

- RER: Refactoring Effect Relevance


Newly developed
Source Code of Object-Oriented Program
I. Refactoring Candidate Identification
Extracting with Dynamic Information based Rules
RER-aware Grouping into Maximal Independent Sets
Refactoring Candidates
II. Refactoring Selection
Refactorings' Effect Evaluation Framework
Assessing the Effect of Refactorings
Selecting Multiple Refactorings
Selected Multiple Refactorings
Apply Selected Refactorings
Stop
No Improvement

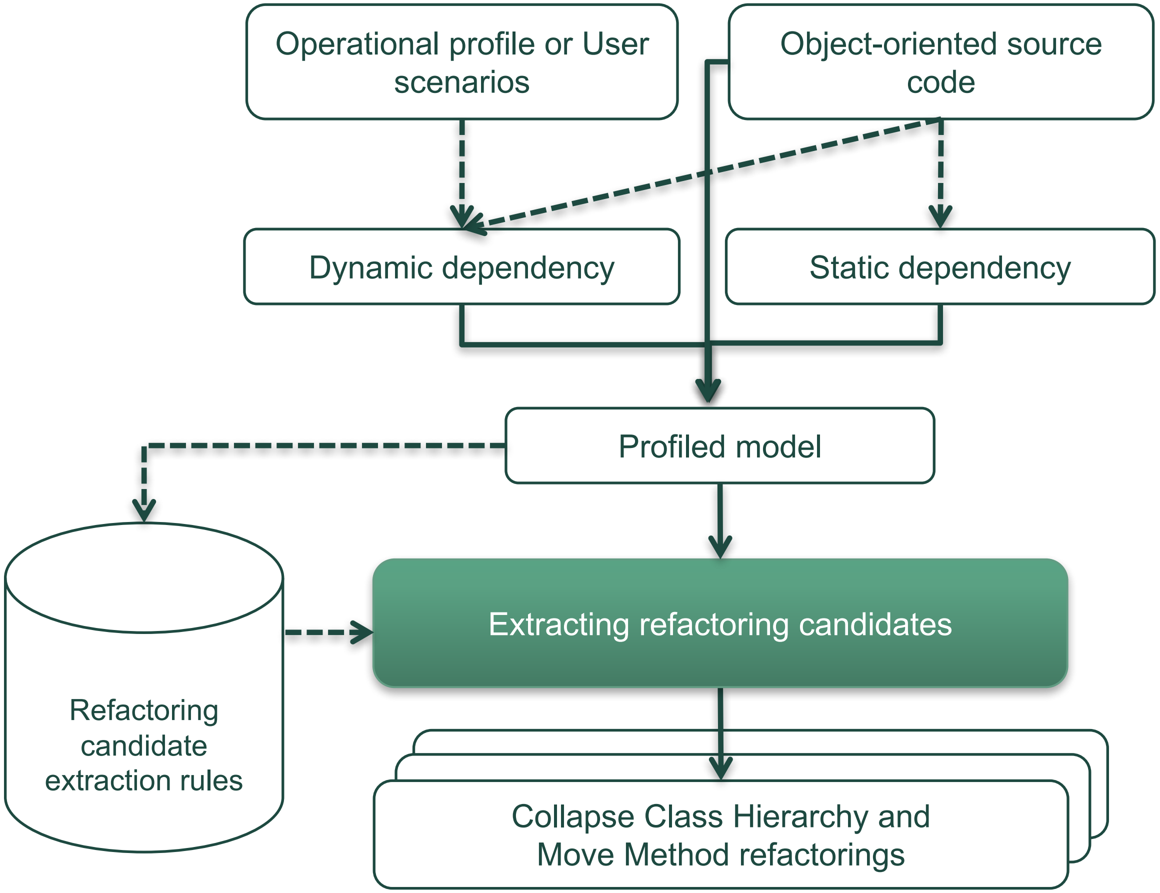
Refactoring Candidate Identification:

Extracting with Dynamic Information Based Rules

"Dynamic profiling-based approach to identifying cost-effective refactorings", Information and Software Technology (IST), Vol. 55, No. 6, pp. 966-985, Jun. 2013.

Extracting with Dynamic Information Based Rules

- Overview



Operational profile or User scenarios

Object-oriented source code

Dynamic dependency

Static dependency

Profiled model

Refactoring candidate extraction rules

Extracting refactoring candidates

Collapse Class Hierarchy and Move Method refactorings

Design Problems and Resolving Refactoring

- Change Preventing Related Design Problems [Fowler'1999]
  - Many classes are modified when making a single change to a system (e.g., Shotgun Surgery)
  - A single class is modified by many different types of changes (e.g., Divergent Change)

- Resolving Refactorings
  - Refactorings should be applied in a way that reduces dependencies of entities (i.e., methods and classes)
    - Collapse Class Hierarchy and Move Method refactorings

Use of Dynamic Dependency

- Dynamic dependency enables to find
  - Entities being really in use
  - Frequency of the relations for those entities

- Dynamic dependencies (DMC)
  - Obtained using dynamic profiling by executing programs
    - Based on dynamic method calls

Refactoring Candidate Extraction Rules

- Rules are defined for reducing dynamic dependencies for identifying refactoring candidates
  - Total of 18 rules (6 types of heuristic design strategies x 3 types of refactorings)
    - When the called methods are implemented in the N (N = 2, 3, 4, 5, 6) different classes (*N*Diff)
      - $^{\forall}(c_i, c_j) \in$ *N*Diff_C → Collapse Class Hierarchy (c_i, c_j)
      - $^{\forall}(m_i, m_j) \in$ *N*Diff_M → Move Method (m_i.class, m_j)
      - $^{\forall}(m_i, m_j) \in$ *N*Diff_M → Move Method (m_j.class, m_i)
    - When the two methods have many interactions (Int)
      - $^{\forall}(c_i, c_j) \in$ Int_C → Collapse Class Hierarchy (c_i, c_j)
      - $^{\forall}(m_i, m_j) \in$ Int_M → Move Method (m_i.class, m_j)
      - $^{\forall}(m_i, m_j) \in$ Int_M → Move Method (m_j.class, m_i)

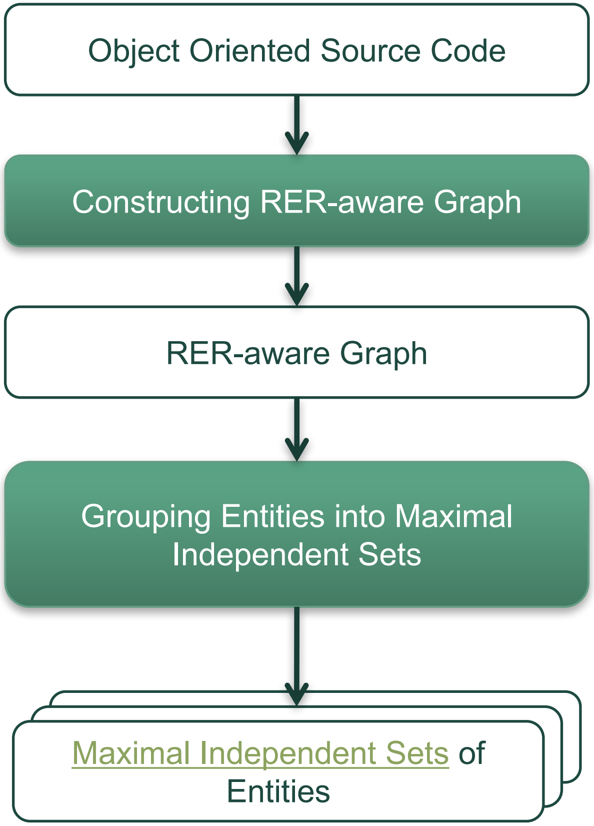
- **c_i (m_i) : class (method) entity in a system**
- **x_C (x_M) : pairs of classes (methods) extracted as refactoring candidates**

Refactoring Candidate Identification:

RER-aware Grouping Entities into Maximal Independent Sets (MISs)

RER-aware Grouping Entities into MISs

Overview

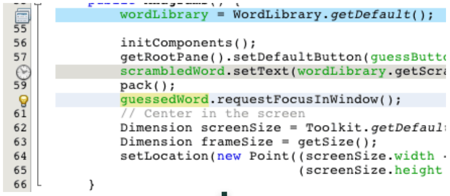

Object Oriented Source Code

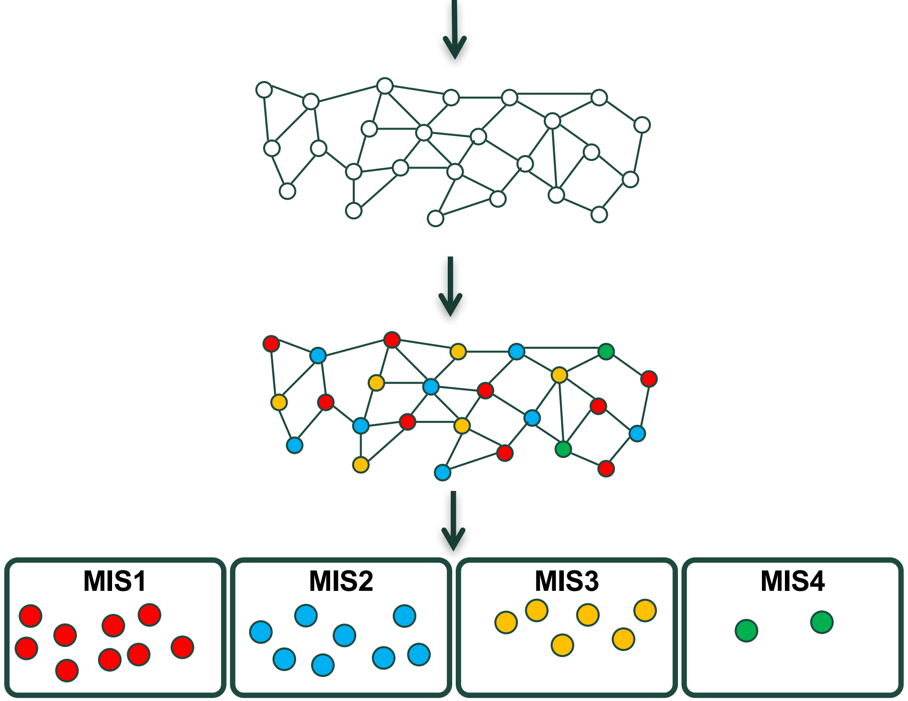
Constructing RER-aware Graph

RER-aware Graph

Grouping Entities into Maximal Independent Sets

Maximal Independent Sets of Entities

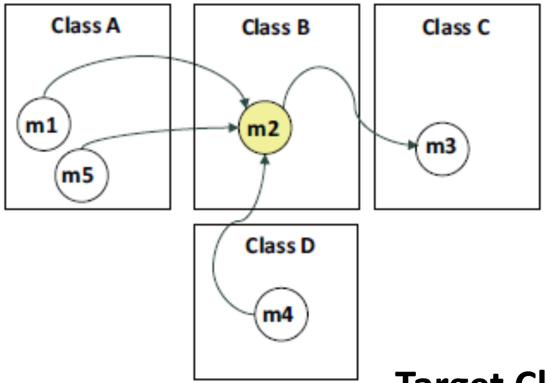

```
        wordLibrary = WordLibrary.getDefault();

        initComponents();
        getRootPane().setDefaultButton(guessButto
        scrambledWord.setText(wordLibrary.getScra
        pack();
        guessedWord.requestFocusInWindow();
        // Center in the screen
        Dimension screenSize = Toolkit.getDefaul
        Dimension frameSize = getSize();
        setLocation(new Point((screenSize.width -
                                  (screenSize.height
    }
```



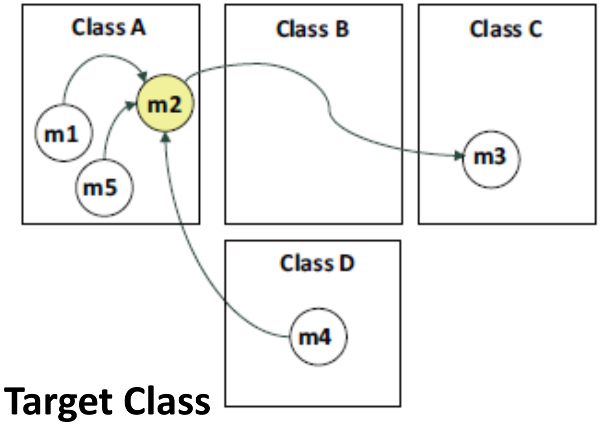
- RER: Refactoring Effect Relevance
- MIS: Maximal Independent Set

Refactorings' Effect Relevance (RER)

- Motivating example



After moving method m2 to class A



Target Class

Moving Method

	A	B	C	D
m1	-	-1	0	0
m2	-2	-	-1	-1
m3	0	-1	-	-1
m4	0	-1	0	-
m5	-	-1	0	0

Target Class

Moving Method

	A	B	C	D
m1	-	1	1	1
m2	-	2	1	1
m3	-1	0	-	0
m4	-1	0	0	-
m5	-	1	1	1

Delta of coupling for each of Move Method refactoring

Example : applying Move Method(method m2, class A) and Move Method(method m1, class B)	
Expected reduced coupling : -3	Actual reduced coupling: -1
Move Method(method m2, class A) = -2	Move Method(method m2, class A) = -2
Move Method(method m1, class B) = -1	Move Method(method m1, class B) = +1

RER-aware Graph

- G = (V, E) for the corresponding object-oriented program is constructed
  - Representing entities (V) and their associations (E)
    - V = {methods, attributes}
    - E = {method_calls (method m1, method m2), attribute_assesses$_1$ (method m1, attribute a1), attribute_assesses$_2$(method m1, method m2)}

- Associations:

1) a method calls the other method (method call)
2) a method assesses an attribute (attribute_assess$_1$)
3) two methods assess the same attribute (attribute_assess$_2$)

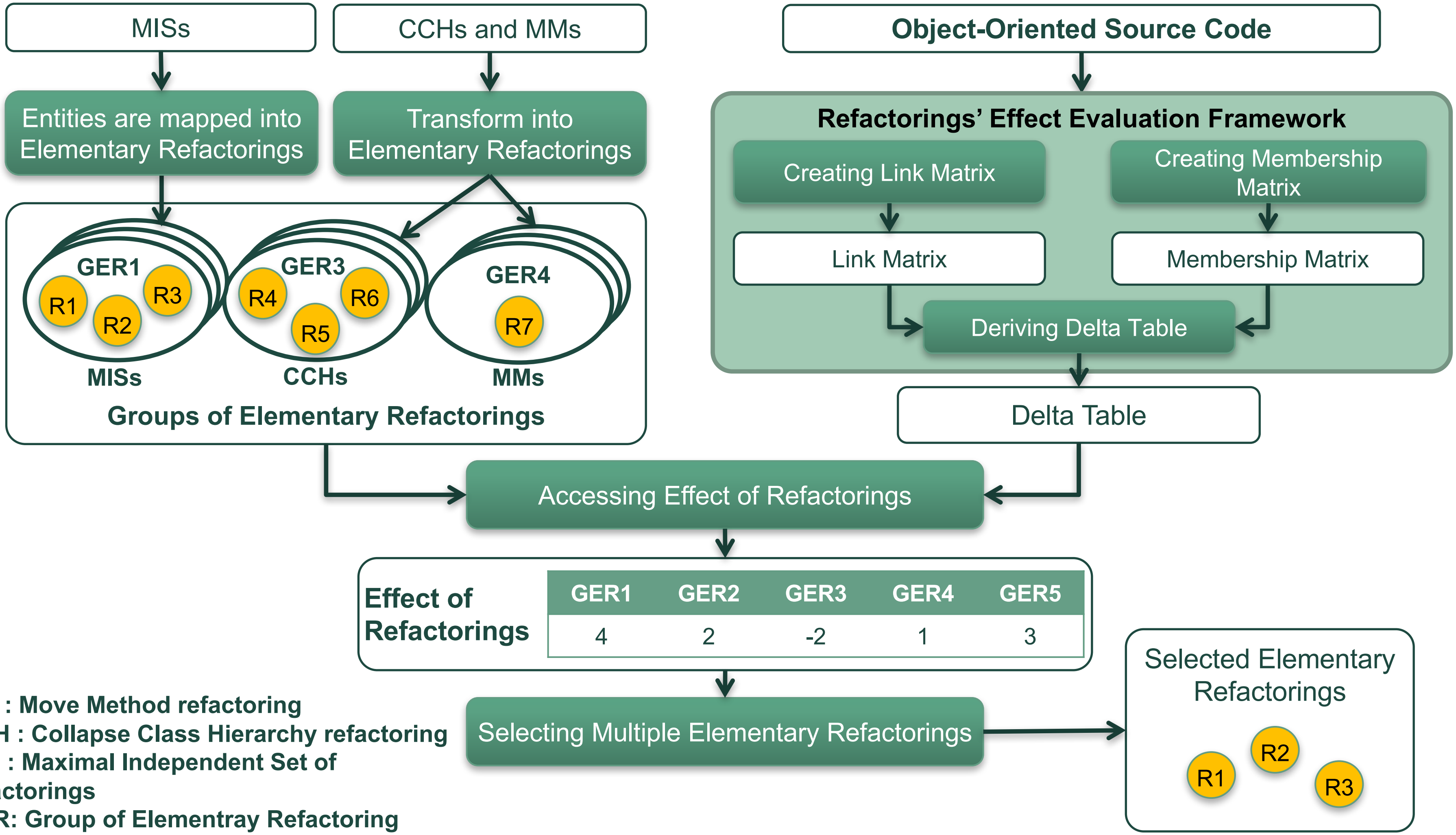
Grouping Entities into MISs

- Procedure
  - Based on G, intermediate groups of entities is obtained by grouping the entities using transitive independent relations
    - (u, v $\in$ V and (u, v) $\notin$ E)
  - Then, remaining entities are assigned on the intermediate groups of entities
    - Until no more entities can be added to any other groups of entities without violating the independence property
  - Finally, groups of entities (= MISs) are obtained; and attributes are excluded from MISs

Refactoring Selection

Selecting Multiple Elementary Refactorings

Overview


MISs
CCHs and MMs
Object-Oriented Source Code
Entities are mapped into Elementary Refactorings
Transform into Elementary Refactorings
Refactorings' Effect Evaluation Framework
Creating Link Matrix
Creating Membership Matrix
Link Matrix
Membership Matrix
Deriving Delta Table
GER1
R1
R2
R3
GER3
R4
R5
R6
GER4
R7
MISs
CCHs
MMs
Groups of Elementary Refactorings
Delta Table
Accessing Effect of Refactorings
Selecting Multiple Elementary Refactorings
Selected Elementary Refactorings
R1
R2
R3

Effect of Refactorings

GER1	GER2	GER3	GER4	GER5
4	2	-2	1	3

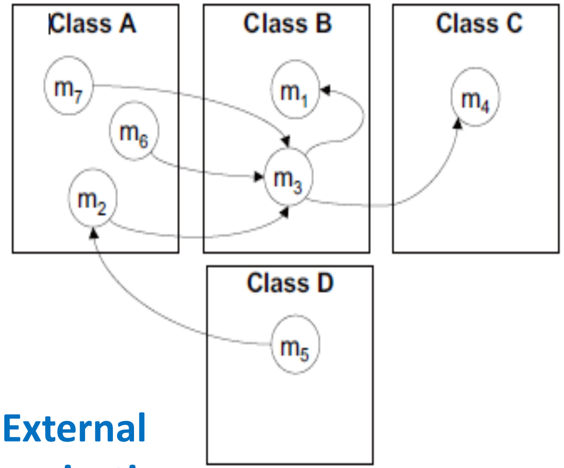
- **MM : Move Method refactoring**
- **CCH : Collapse Class Hierarchy refactoring**
- **MIS : Maximal Independent Set of refactorings**
- **GER: Group of Elementray Refactoring**

Refactorings' Effect Evaluation (1/2)

- Delta Table (D)
  - Provides the method for evaluating elementary refactorings' effect on maintainability
    - Each element indicates Δ *maintainability*
      - Maintainability variance after the application of the elementary refactoring on the current design configuration
    - Maintainability is assessed by the number of external links
      - This number of external links naturally represents *lack of cohesion* and, at the same time, *coupling*
      - As a result, by applying refactorings, we aim to *reduce this number for improving maintainability*
  - Computed by matrix computation (fast)

Refactorings’ Effect Evaluation (2/2)

- Delta Table derivation
  - Formulation
    - $L_{Int}\ \mathrm{X}\ M = P_{Int};\ L_{Ext}\ \mathrm{X}\ M = P_{Ext};\ Inv(P_{Int}) - P_{Ext} = D$
  - Example



Internal link matrix (L_{Int})

L_{Int}	m1	m2	m3	m4	m5	m6	m7
m1	0	0	1	0	0	0	0
m2	0	0	0	0	0	0	0
m3	1	0	0	0	0	0	0
m4	0	0	0	0	0	0	0
m5	0	0	0	0	0	0	0
m6	0	0	0	0	0	0	0
m7	0	0	0	0	0	0	0

X

Membership matrix (M)

M	A	B	C	D
m1	0	1	0	0
m2	1	0	0	0
m3	0	1	0	0
m4	0	0	1	0
m5	0	0	0	1
m6	1	0	0	0
m7	1	0	0	0

=

P_{Int}	A	B	C	D
m1	0	1	0	0
m2	0	0	0	0
m3	0	1	0	0
m4	0	0	0	0
m5	0	0	0	0
m6	0	0	0	0
m7	0	0	0	0

External link matrix (L_{Ext})

L_{Ext}	m1	m2	m3	m4	m5	m6	m7
m1	0	0	0	0	0	0	0
m2	0	0	1	0	1	0	0
m3	0	1	0	1	0	1	1
m4	0	0	1	0	0	0	0
m5	0	1	0	0	0	0	0
m6	0	0	1	0	0	0	0
m7	0	0	1	0	0	0	0

X

Membership matrix (M)

M	A	B	C	D
m1	0	1	0	0
m2	1	0	0	0
m3	0	1	0	0
m4	0	0	1	0
m5	0	0	0	1
m6	1	0	0	0
m7	1	0	0	0

=

P_{Ext}	A	B	C	D
m1	0	0	0	0
m2	0	1	0	1
m3	3	0	1	0
m4	0	1	0	0
m5	1	0	0	0
m6	0	1	0	0
m7	0	1	0	0

Inversed internal projection matrix $Inv(P_{Int})$

$Inv(P_{Int})$	A	B	C	D
m1	1	0	1	1
m2	0	0	0	0
m3	1	0	1	1
m4	0	0	0	0
m5	0	0	0	0
m6	0	0	0	0
m7	0	0	0	0

—

External projection matrix (P_{Ext})

P_{Ext}	A	B	C	D
m1	0	0	0	0
m2	0	1	0	1
m3	3	0	1	0
m4	0	1	0	0
m5	1	0	0	0
m6	0	1	0	0
m7	0	1	0	0

=

Delta Table (D)

D	A	B	C	D
m1	1	-	1	1
m2	-	-1	0	-1
m3	-2	-	0	1
m4	0	-1	-	0
m5	-1	0	0	-
m6	0	-1	0	0
m7	0	-1	0	0

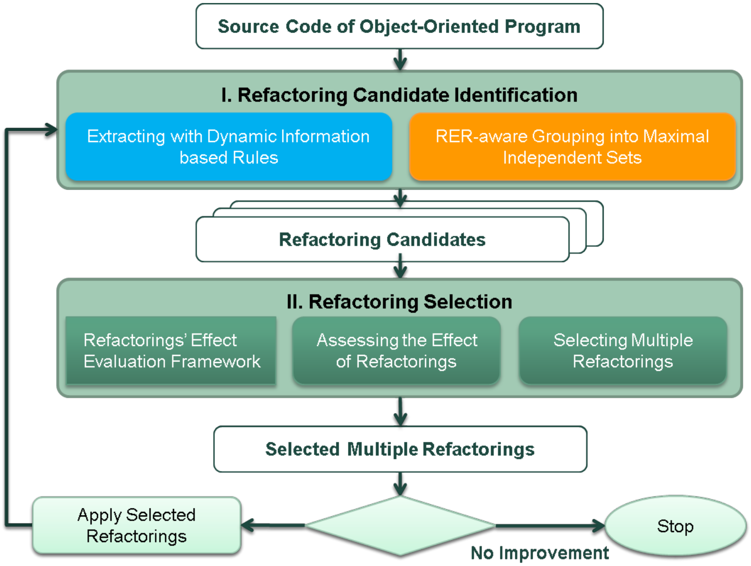
Evaluation

Research Questions

- [RQ 1.] Effect of dynamic information
  - Is the dynamic information helpful in identifying refactorings that effectively improve maintainability?
- [RQ 2.] Effect of multiple refactorings
  - Do the multiple refactorings help to improve maintainability and reduce search space exploration?
  - Is the RER an important when grouping entities into MISs?

RQ 1. Effect of dynamic information

RQ 2. Effect of multiple refactorings



Source Code of Object-Oriented Program

I. Refactoring Candidate Identification

Extracting with Dynamic Information based Rules

RER-aware Grouping into Maximal Independent Sets

Refactoring Candidates

II. Refactoring Selection

Refactorings' Effect Evaluation Framework

Assessing the Effect of Refactorings

Selecting Multiple Refactorings

Selected Multiple Refactorings

Apply Selected Refactorings

No Improvement

Stop

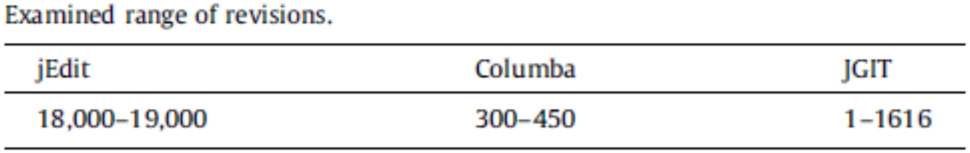
Experimental Subjects

- Characteristics and development history for each subject

Name **(Version)**	**jEdit** (jEdit-4.3)	**Columba** (Columba-1.4)	**jGit** (jGit-1.1.0)
Type	Text editor	Email clients	Distributed source version control system
Total # of revisions	19501	458	1616
Report period	2001-09 ~ 2011-09	2006-07 ~ 2011-07	2009-09 ~ 2011-09
Number of developers	25	9	9
Class #	952	1506	689
Method #	6487	8745	5334
Attribute #	3523	3967	2989

Effect of Dynamic Information

- Experimental design
  - To assess the capability of refactorings for maintainability improvement, we use the *change simulation*
    - Extract changes as input for change impact analysis
      - Changed methods that had occurred within the examined revisions of the development history



Examined range of revisions.

jEdit	Columba	JGIT
18,000–19,000	300–450	1–1616

    - Obtain *propagated changes* by performing change impact analysis
  - We compare the **reduced number of propagated changes**
    - approach using dynamic information only (dynamic)
    - approach using static information only (static)
    - combination of the two approaches (dynamic + static)

Effect of Dynamic Information

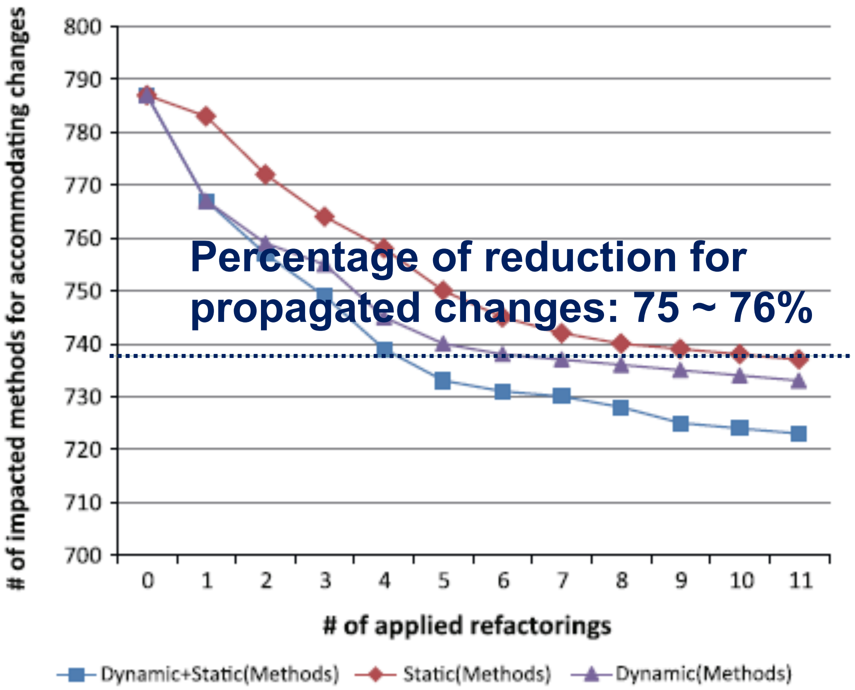
- Results
  - Ex) Columba

- **Average rate of reduction for propagated changes (%)**

Dynamic+Static	Static	Dynamic
9.09	7.10	7.67

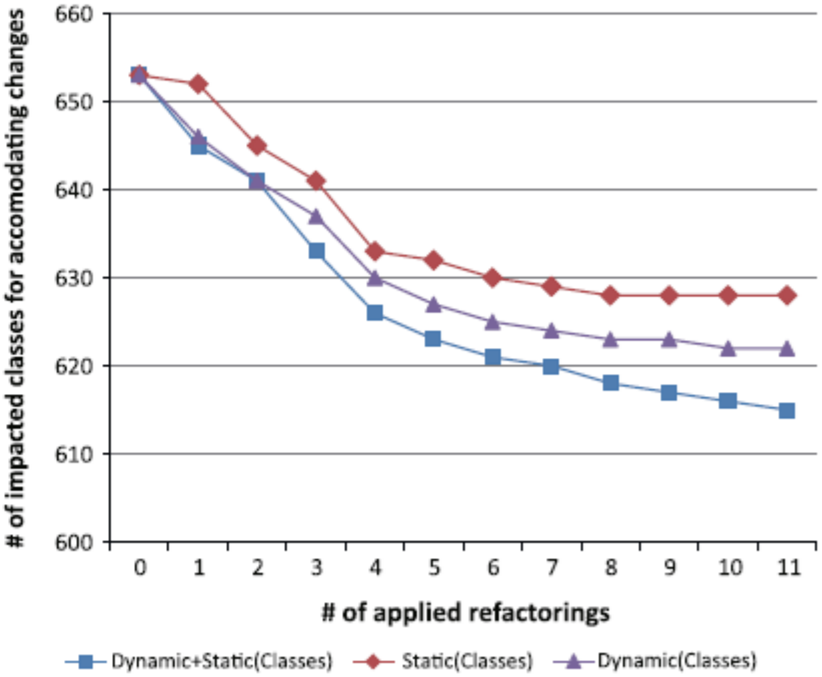
- **Percentage of reduction for propagated changes (%)**

Dynamic+Static	Static	Dynamic
100	78.1	84.4



Percentage of reduction for propagated changes: 75 ~ 76%

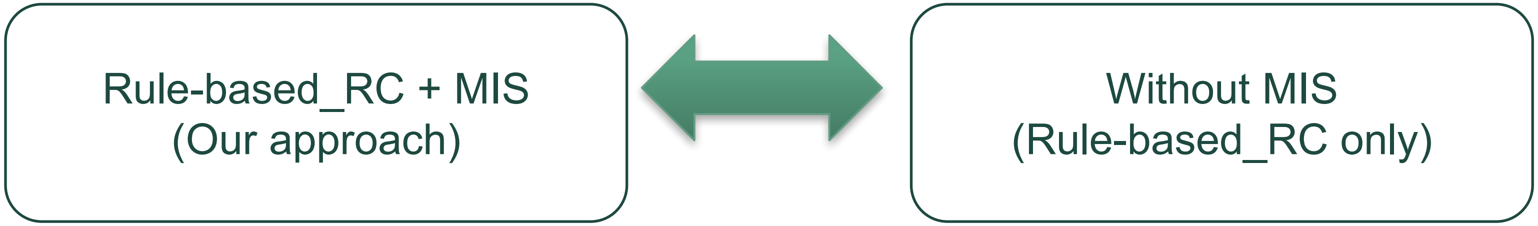
(a) Number of propagated methods for accommodating changes on Columba



(b) Number of propagated classes for accommodating changes on Columba

Effect of Multiple Refactorings

- Experimental design
  - Effect of multiple refactorings



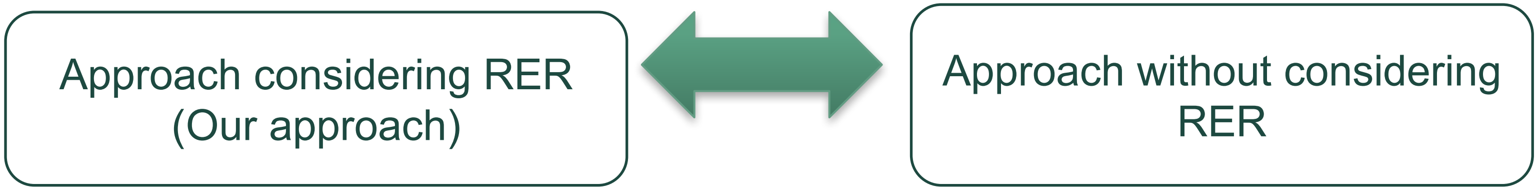
Rule-based_RC + MIS
(Our approach)

Without MIS
(Rule-based_RC only)

Comparing 1) Fitness [Han'2013]; 2) # of iterations and Elapsed time (sec)

- Rule-based_RC: Approach of rule-based identification of refactoring candidates
- MIS: Approach of grouping into MISs

  - Effect of RER



Approach considering RER
(Our approach)

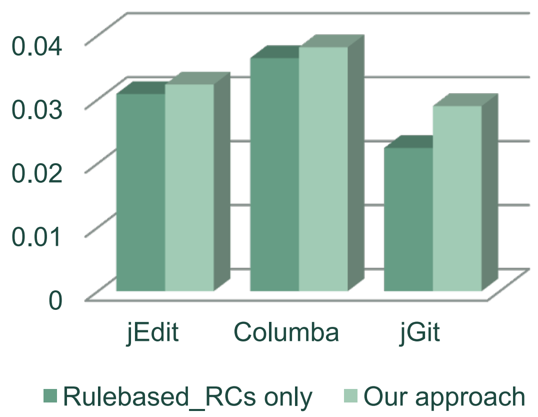
Approach without considering RER

Comparing 1) Fitness [Han'2013]; 2) deviation between *actual* and *expected* maintainability

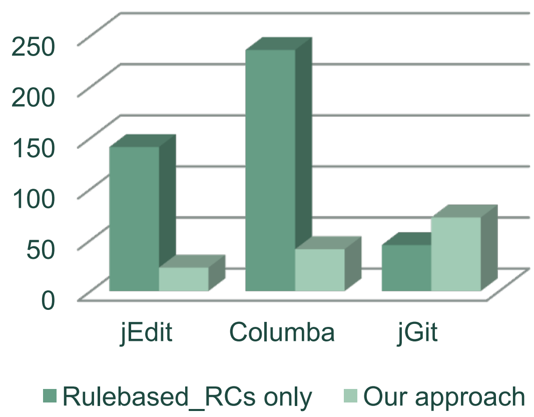
Effect of Multiple Refactorings

- Results
  - Summary

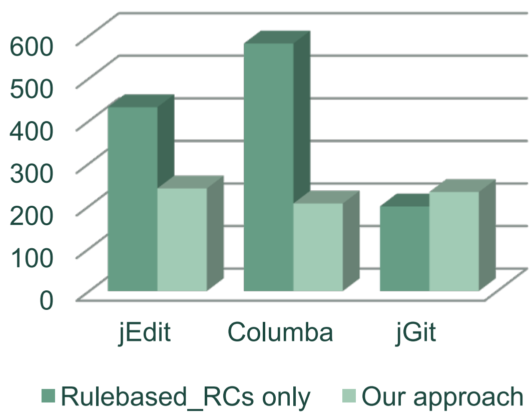
Fitness



# of iterations



Elapsed Time (sec)

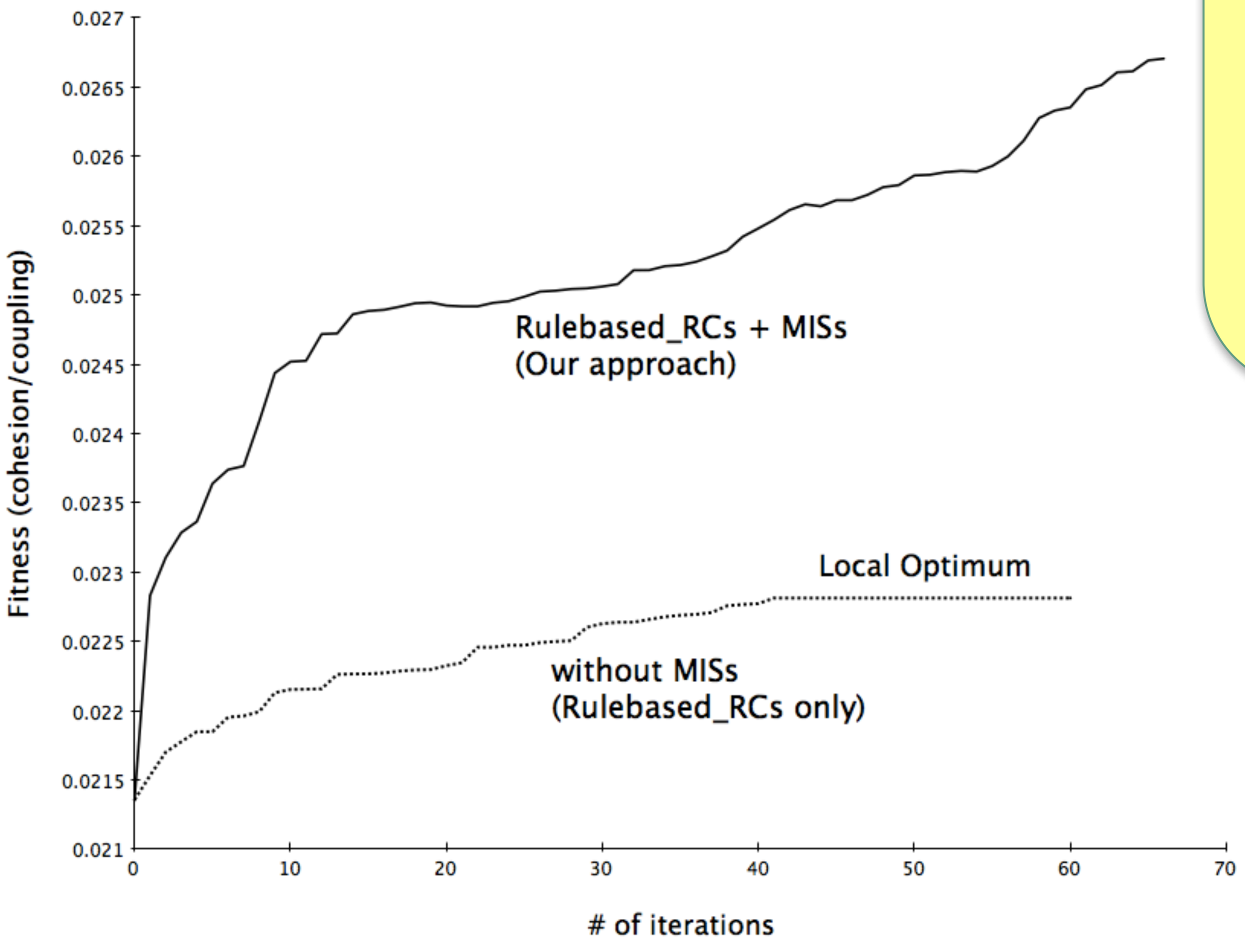


- Rule-based_RC: Approach of rule-based identification of refactoring candidates
- MIS: Approach of grouping into MISs

- Rulebased_RCs only: approach without MISs
- Our approach: approach with Rulebased_RCs + MISs

Effect of Multiple Refactorings

- Results
  - Ex) jGit



→ In jGit, big refactoring results in local optimum

During the iterative process, it finds the refactoring candidates in the same place

→ Selecting refactorings globally helps to prevent this problem

Effect of RER

- Results
  - Summary

Subject	Comparators	Fitness fn.	Accumulated deviation
jEdit	Not_RER	0.032379	9246
	Our approach	0.033472	846
Columba	Not_RER	0.030720	40758
	Our approach	0.037123	481
jGit	Not_RER	0.023602	13058
	Our approach	0.028192	913

- Not_RER: approach without considering RER
- Our approach: approach considering RER
- Accumulated deviation

$$\sum_{i=0}^{\#\ of\ Iteration} |Expected_i - Actual_i|$$

$Expected_i$: expected maintainability on i-th iteration

$Actual_i$: actual maintainability on i-th iteration

Related Work

Related Work (1/2)

- Refactoring identification based on static metrics [Tahvildari’2003; Zhao’2006]
  - The used metrics are all static
  - Neither clear rules for detecting design flaws nor a method of how to apply refactorings
  - No quantitative method for evaluating the effect of refactorings

Related Work (2/2)

- Determining refactoring sequences to be applied by selecting the best refactoring in a greedy way [Tsantalis’2009; Han’2013]
  - Inefficient to select just one best refactoring for the iteration of refactoring identification process

- Analysis of dependencies or conflicts between refactoring candidates [Mens’2007; Hotta’2012]
  - Only considered syntactic dependency

Conclusion and Future Work

Conclusion

- Provide the methods for supporting systematic refactoring identification
  - Develop the method for dynamic information-based identification of refactoring candidates
  - Develop the method for RER-aware grouping entities of MIS and selecting multiple refactorings

Future Work

- We plan to consider more types of refactorings
  - For example, Pull Up Method refactoring and Form Template Method refactoring
  - Our framework of refactorings' effect evaluation
    - Can support to easily extend considering refactorings to other various type of refactorings
      - Because it provides the method of assessment and impact analysis of elementary refactorings
      - The action of big refactoring comprises of elementary refactorings

Thank You .

KAIST
SE LAB
KAIST Software Engineering Laboratory

Reference (1/4)

- [Parnas’1994(ICSE)] D. Parnas, Software aging, in: Proceedings of The 16th International Conference on Software Engineering (ICSE94), IEEE Computer Society Press, 1994. pp. 279–287.
- [Fowler’1999] M. Fowler, K. Beck, Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, 1999.
- [Zarnekow’2005] Zarnekow R and Brenner W. 2005. 'Distribution of cost over the application lifecycle - A multi-case study', Proceedings of the Thirteenth European Conference on Information Systems, Regensburg.
- [Robbes’2010] R. Robbes, D. Pollet, M. Lanza, Replaying ide interactions to evaluate and improve change prediction approaches, in: 7th IEEE Working Conference on Mining Software Repositories (MSR), 2010, IEEE, pp. 161–170.
- [Arisholm’2004] E. Arisholm, L. Briand, A. Føyen, Dynamic coupling measurement for object-oriented software, IEEE Transactions on Software Engineering (2004) 491–506.
- [Han’2010] A.-R. Han, S.-U. Jeon, D.-H. Bae, J.-E. Hong, Measuring behavioral dependency for improving change-proneness prediction in uml-based design models, The Journal of Systems & Software 83 (2010) 222–234.
- [Han’2013] Ah-Rim Han, Doo-Hwan Bae, Dynamic profling-based approach to identifying cost-effective refactorings, Information and Software Technology (IST), published on-line version (Dec. 2012). (http://dx.doi.org/10.1016/j.infsof.2012.12.002)
- [Musa’1993] J. Musa, Operational profiles in software-reliability engineering,, IEEE Software 10 (1993) 14–32.
- [Sharieh’2008] Ahmad Sharieh, Wagdi Al_Rawagepfeh, Mohammed H. Mahafzah, and Ayman Al Dahamsheh, “An Algorithm for Finding Maximum Independent Set in a Graph”, European Journal of Scientific Research, Vol.23, No.4 (2008), pp.586-596

Reference (2/4)

- [Tahvildari‘2003(CSMR)] A metric-based approach to enhance design quality through meta-pattern transformations, Proc. European Conf. Software Maintenance and Reeng.
- [Kerievsky'2005] Refactoring to patterns, Pearson Education.
- [Jeon'2002(APSEC)] An automated refactoring approach to design pattern-based program transformations in java programs, IEEE Software Engineering Conference in Asia-Pacific.
- [Tsantalis'2009(TSE)] Identification of move method refactoring opportunities, Software Engineering, IEEE Transactions.
- [Higo'2008(JSME)] A metric-based approach to identifying refactoring opportunities for merging code clones in a java software system, Journal of Software Maintenance and Evolution: Research and Practice.
- [Lee'2011(SPE)] Automated scheduling for clone-based refactoring using a competent GA, Softw., Pract. Exper.
- [Zibran'2011] Conflict-aware optimal scheduling of code clone refactoring: A constraint programming approach, in: Program Comprehension (ICPC), 2011 IEEE 19th International Conference on, IEEE.
- [Harman'2011(ICSTW)] Refactoring as testability transformation, in: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, IEEE.
- [Fagin’2003] R. Fagin, R. Kumar, D. Sivakumar, Comparing top k lists, in: Proceedings of the Fourteenth Annual ACM–SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2003, pp. 28–36.
- [Bonja’2006] C. Bonja, E. Kidanmariam, Metrics for class cohesion and similarity between methods, in: Proceedings of the 44th Annual Southeast Regional Conference, 2006, pp. 91–95.

Reference (3/4)

- [Tsantalis'2010(JSS)] A. Chatzigeorgiou, Identification of refactoring opportunities introducing polymorphism, Journal of Systems and Software.
- [Simon'2001(CSMR)] Metrics based refactoring, in: Software Maintenance and Reengineering, 2001. Fifth European Conference on, IEEE.
- [Seng'2006(GECCO)] Search-based determination of refactorings for improving the class structure of object-oriented systems, Proceedings of the 8th annual conference on Genetic and evolutionary computation.
- [O'Keeffe'2008(JSS)] Search-based refactoring for software maintenance, The Journal of Systems & Software.
- [Tsantalis'2011(CSMR)] Ranking Refactoring Suggestions based on Historical Volatility, Software Maintenance and Reengineering (CSMR), IEEE.
- [DuBois'2004(CRE)] Refactoring - improving coupling and cohesion of existing code, in: Proceedings of the 11th Working Conference on Reverse Engineering, IEEE Computer Society.
- [Liu'2008(IET)] Conflict-aware schedule of software refactorings, Software, IET.
- [Mens'2007(SoSyM)] Analysing refactoring dependencies using graph transformation, Software and Systems Modeling.
- [Brooks'2012] Metrics based Refactoring for cleaner code, http://www.grahambrooks.com/blog/metrics-based-refactoring-for-cleaner-code/
- [Alon'1986] Noga Alon, L´aszl´o Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of algorithms, 7(4):567–583, 1986.
- [Johnson'1998] Johnson, David S and Yannakakis, Mihalis and Papadimitriou, Christos H, "On generating all maximal independent sets", Information Processing Letters, Vol. 27, No. 3, (1988), pp. 119—123
- [Luby'1986] Michael Luby. A simple parallel algorithm for the maximal independent set problem. SIAM journal on computing, 15(4):1036–1053, 1986.

Reference (4/4)

- [Hotta'2012] K. Hotta, Y. Higo, and S. Kusumoto. Identifying, tailoring, and suggesting form template method refactoringopportunities with program dependence graph. In 16th European Conference on Software Maintenance and Reengineering (CSMR'12), pages 53–62, 2012.
- [Zhao'2006] L. Zhao and J.H. Hayes. Predicting classes in need of refactoring: An application of static metrics. In Proceedings of the workshop on predictive models of software engineering (PROMISE), associated with ICSM2006, pages 1–5, 2006.
- [Henderson'1996] B. Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity, Prentice-Hall Inc., Upper Saddle River, NJ, USA, 1996.