An efficient method for assessing the impact of refactoring candidates on maintainability based on matrix computation

Ah-Rim Han[1], Doo-Hwan Bae[2]

1. Korea University, South Korea
2. KAIST, South Korea

2014. 12. 4

Contents

- Introduction
- Overview
- Calculating Delta Table
- Evaluation
- Conclusion and Future Work

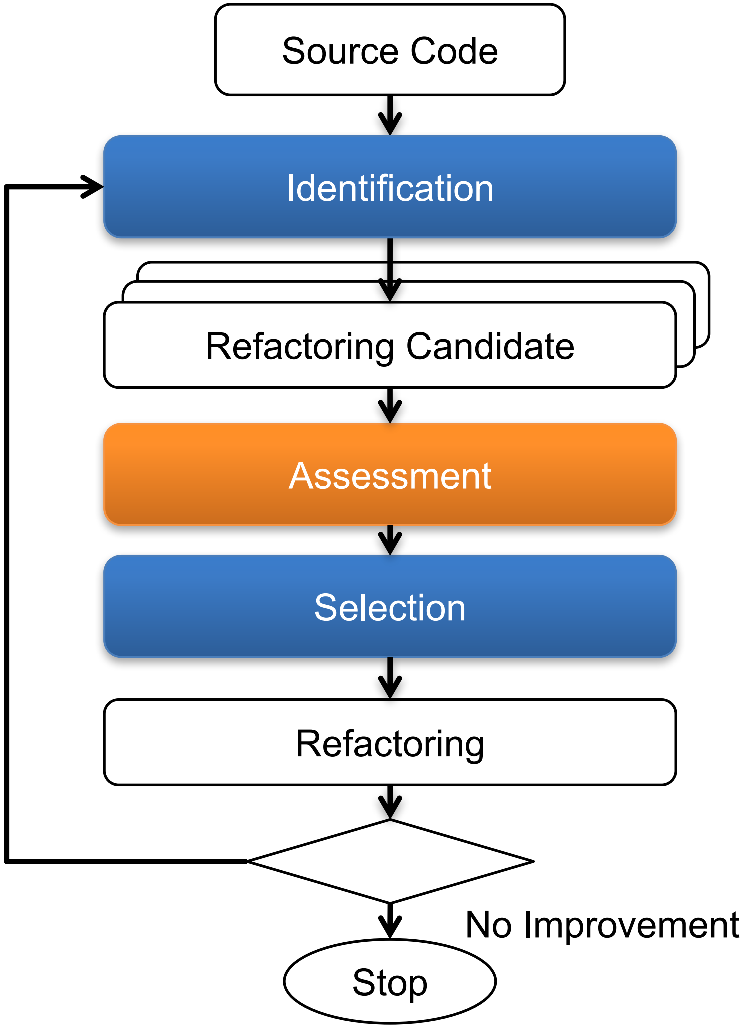
Refactoring in Practice

- Why need refactoring?
  - Design quality of the software degrades overtime
  - Helps better accommodate changes, fix bugs, and reduce maintenance costs

- Although refactoring is beneficial, it is not widely used
  - Lack of systematic methods and tool support
  - Difficult to decide
    - Where to apply which refactoring
    - Which refactorings should be applied first
    - Which refactoring is better
    - …

→ Need to automate the "refactoring identification process"

Studies on Automating Refactoring Identification Process



Stepwise selection approach

- Stepwise selection approach [Han et al. 2013, Tsantalis et al. 2009]
- Search-based refactoring [Seng et al. 2006, Lee et al. 2011, O'Keefee et al. 2008]

→ A large number of refactoring candidates needs to be examined

Motivation

- The cost for assessing refactoring candidates is computation-intensive
- For quantifying and ranking refactoring candidates,
  - Each refactoring candidate is actually or virtually applied
  - Metrics of all classes existing in a system should be calculated
    - For obtaining each metric of a class, all the relations between inner entities or outer entities should be examined

→ **Entity Placement Metric (EPM) [Tsantalis et al. 2013]**

$$EntityPlacement_{System} = \sum_{C_i} \frac{|entities \in C_i|}{|all\ entities|} \boxed{EntityPlacement_{C_i}.}$$

EPM for a class

$$\boxed{EntityPlacement_C} = \frac{\frac{\sum_{e_i \in C} distance(e_i, C)}{|entities \in C|}}{\frac{\sum_{e_j \notin C} distance(e_j, C)}{|entities \notin C|}},$$

→ **We need an efficient (fast and cheaper) method for assessing the impact of refactoring candidates, even there is loss of precision**

$$distance(e, C) = 1 - \frac{|S_e \cap S_C|}{|S_e \cup S_C|}$$

Goal and Our Approach

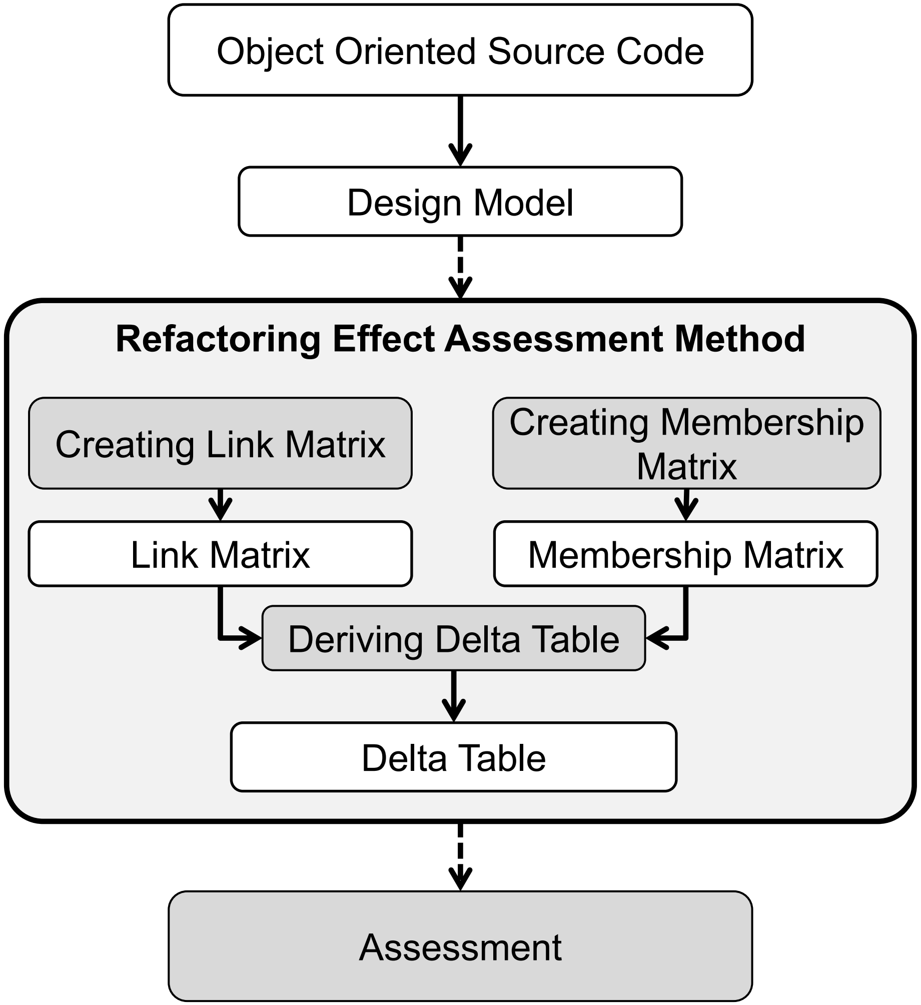
→ We provide an efficient method for refactoring candidates

- Using a more simplistic design model
- Performing fast matrix-based computations

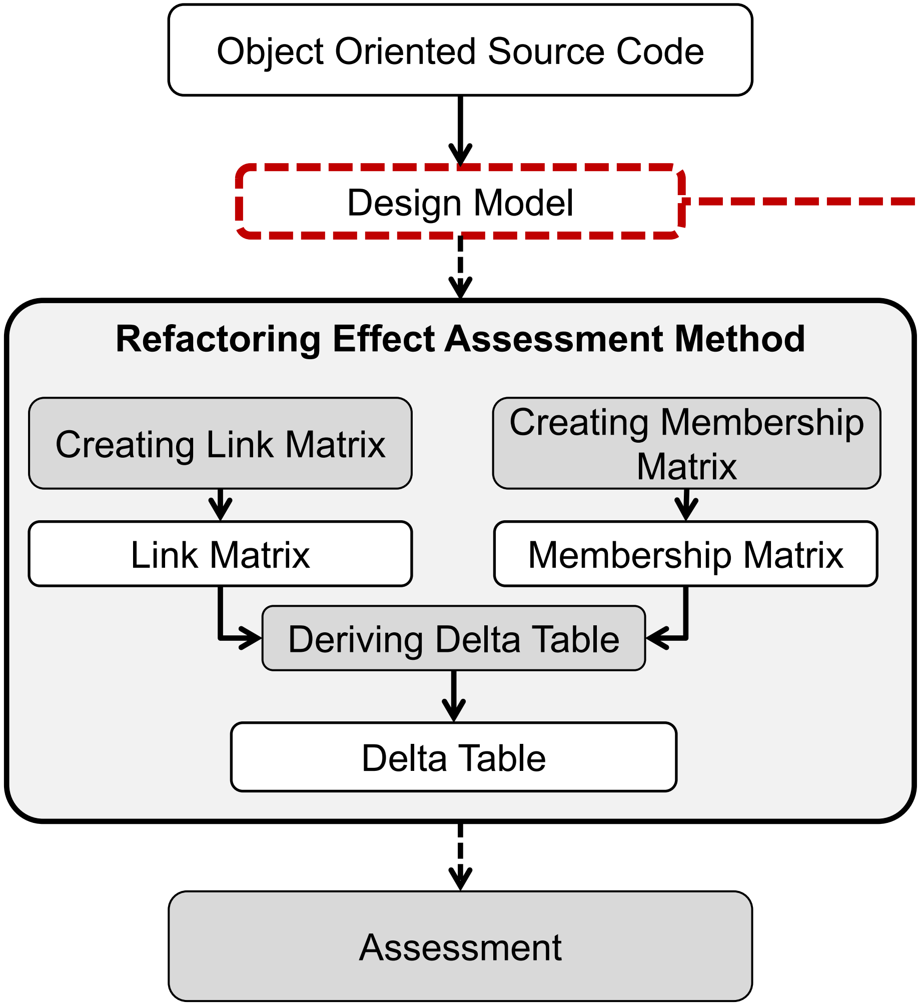
→ In search-based software engineering (SBSE) community [Harman et al. 2013], they address the need for new forms of *surrogate metrics*

- Retain the essence of computationally expensive metrics
- Even sacrifice some degree of precision (for performance)
- Can be used to cheaply assess an approximate fitness to guide a search based approach

Overview : Refactoring Effect Assessment Method



Overview : Refactoring Effect Assessment Method



Object Oriented Source Code

Design Model

Refactoring Effect Assessment Method

Creating Link Matrix

Creating Membership Matrix

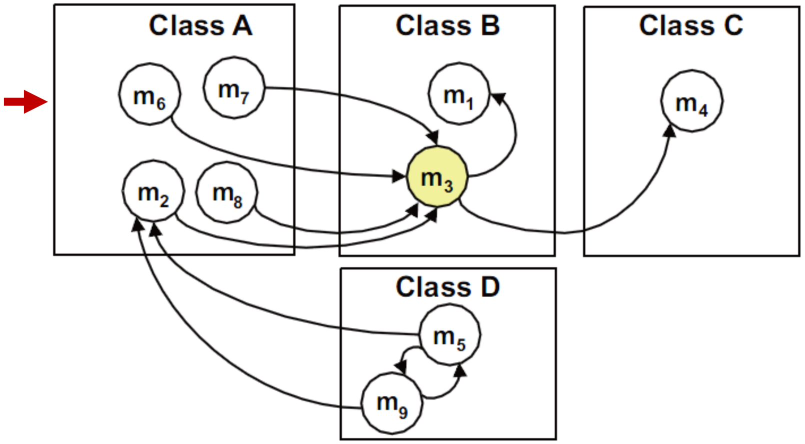
Link Matrix

Membership Matrix

Deriving Delta Table

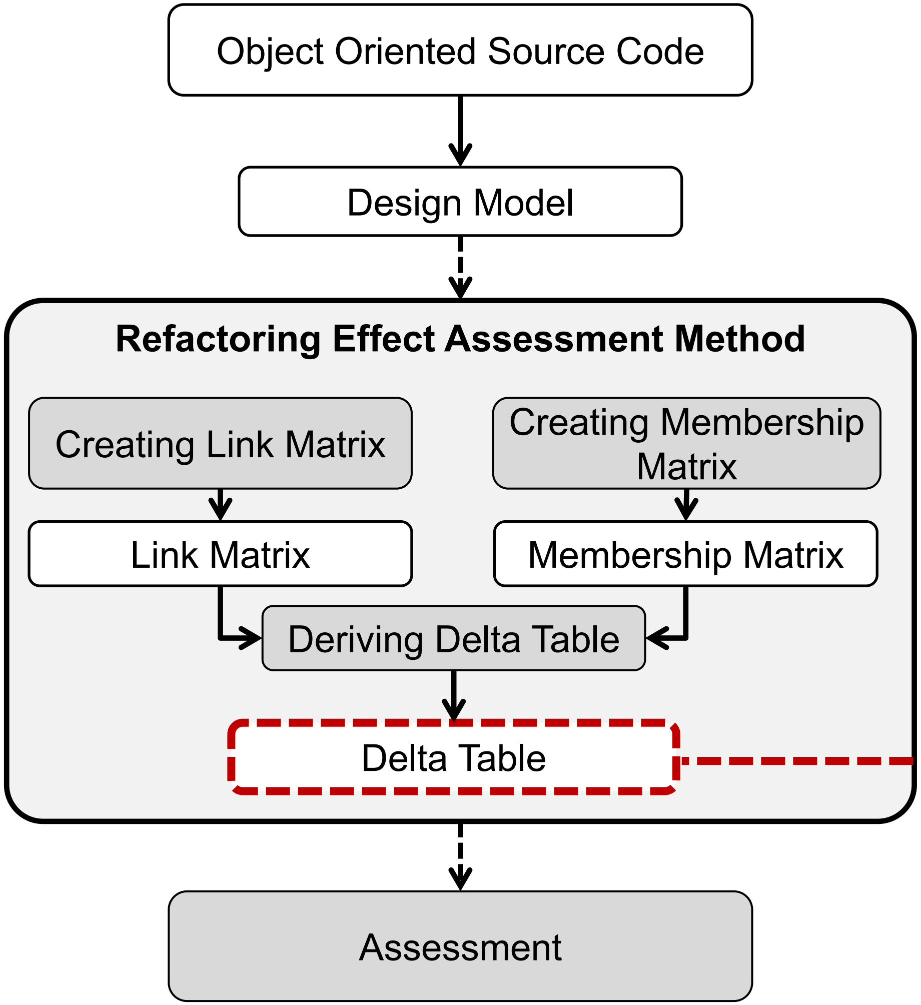
Delta Table

Assessment



G = (V, E)
V = {methods, attributes}
E = {method calls, attributes accesses}

Overview : Refactoring Effect Assessment Method



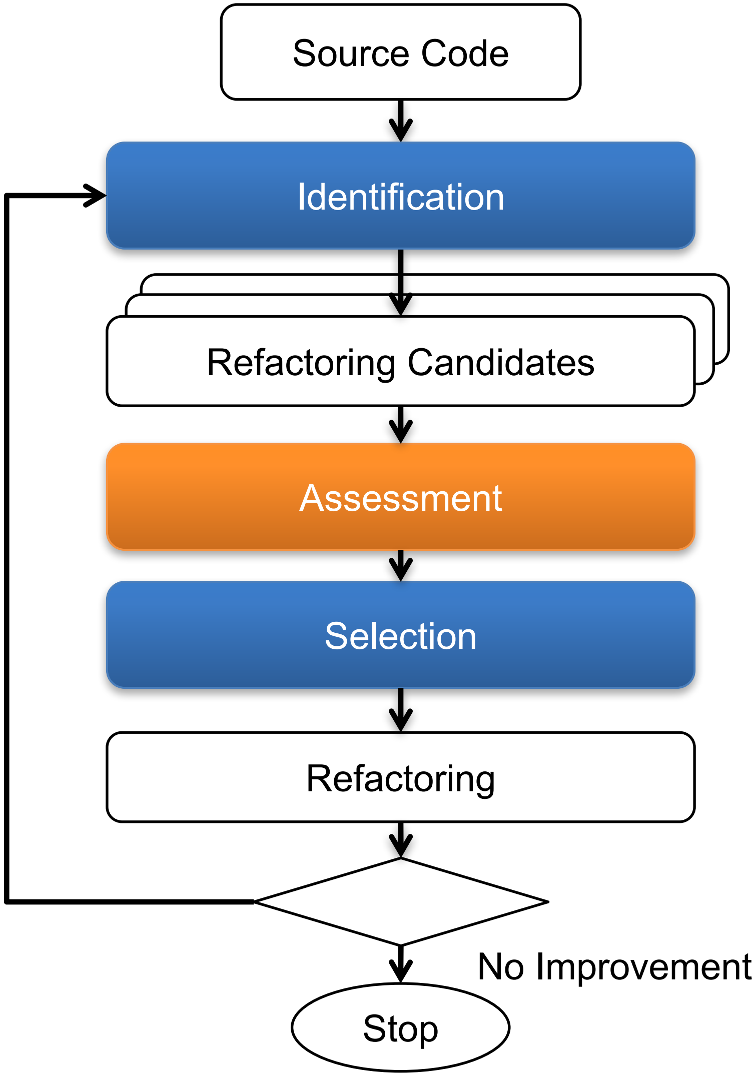
Δ dependency

Target Class

Moving Method

D_n	A	B	C	D
m1	1	-	1	1
m2	-	-1	0	-2
m3	-3	-	0	1
m4	0	-1	-	0
m5	1	2	2	-
m6	0	-1	0	0
m7	0	-1	0	0
m8	0	-1	0	0
m9	1	2	2	-

How to Use in the Entire Refactoring Identification Process

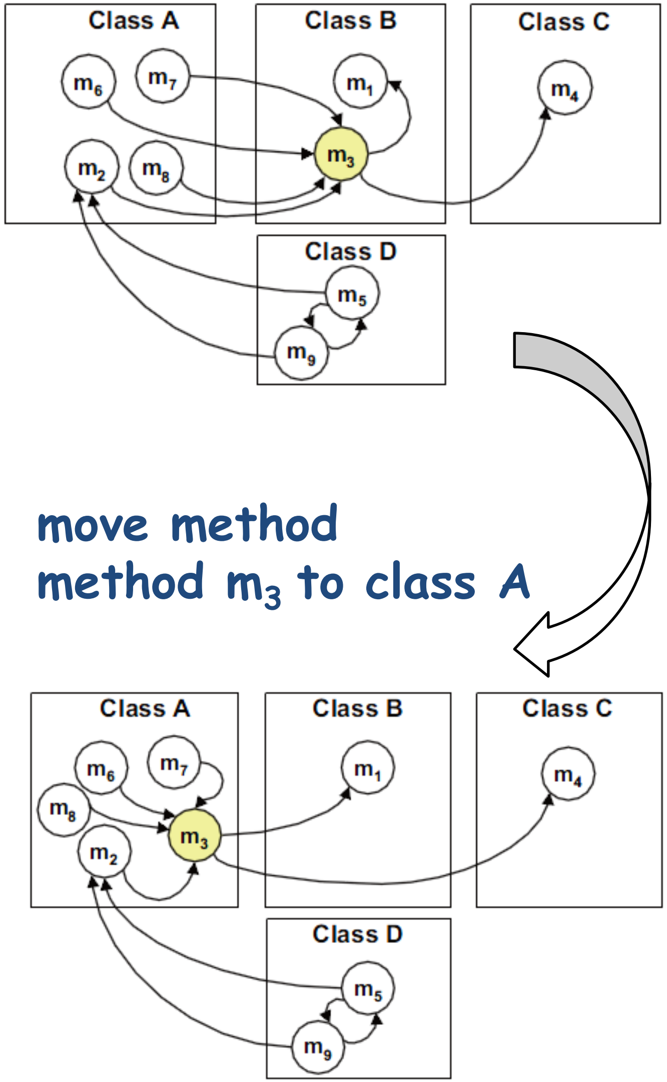


Δ dependency

Target Class

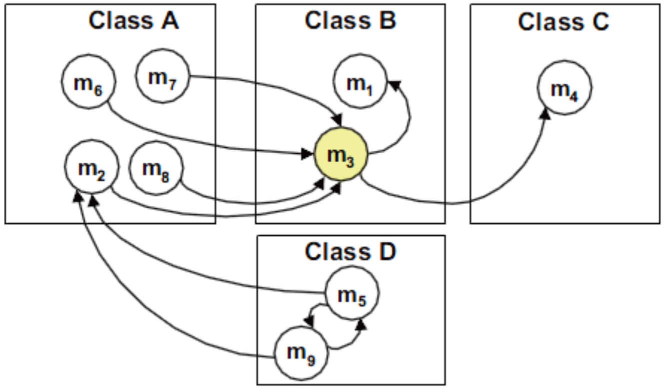
Moving Method

Dn	A	B	C	D
m1	1	-	1	1
m2	-	-1	0	-2
m3	-3	-	0	1
m4	0	-1	-	0
m5	1	2	2	-
m6	0	-1	0	0
m7	0	-1	0	0
m8	0	-1	0	0
m9	1	2	2	-



move method
method m_3 to class A

Calculating Delta Table(1/4)



- Link matrix (L)
  - L(e_1, e_2): entity e_1 has a link to entity e_2
- Membership matrix (M)
  - M(e, C): entity e is placed in class C

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
m8	1	0	0	0
m9	0	0	0	1

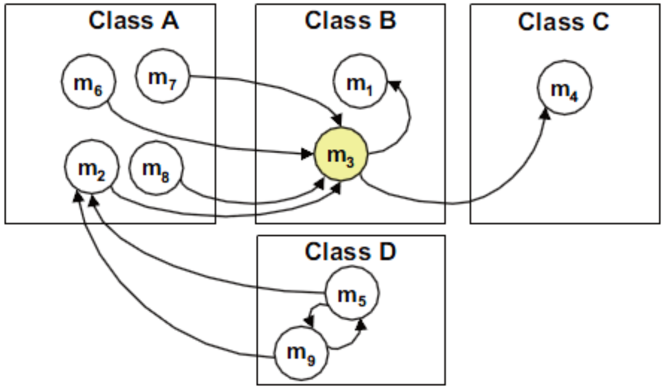
Internal link matrix (L_{Int})

L_{Int}	m1	m2	m3	m4	m5	m6	m7	m8	m9
m1	0	0	1	0	0	0	0	0	0
m2	0	0	0	0	0	0	0	0	0
m3	1	0	0	0	0	0	0	0	0
m4	0	0	0	0	0	0	0	0	0
m5	0	0	0	0	0	0	0	0	2
m6	0	0	0	0	0	0	0	0	0
m7	0	0	0	0	0	0	0	0	0
m8	0	0	0	0	0	0	0	0	0
m9	0	0	0	0	2	0	0	0	0

External link matrix (L_{Ext})

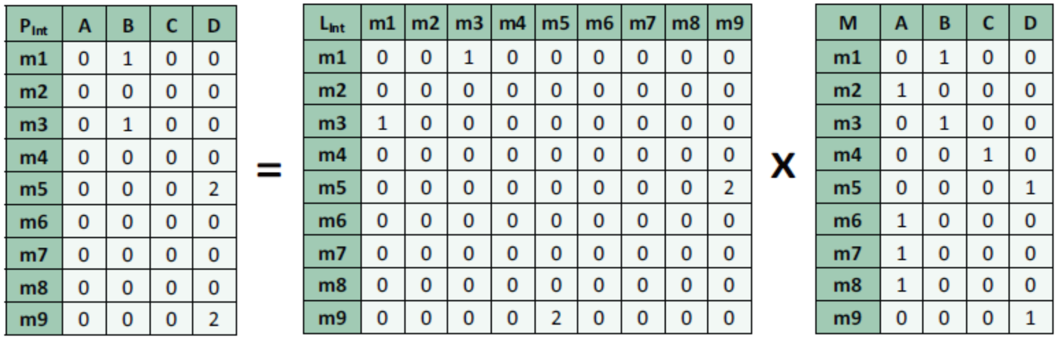
L_{Ext}	m1	m2	m3	m4	m5	m6	m7	m8	m9
m1	0	0	0	0	0	0	0	0	0
m2	0	0	1	0	1	0	0	0	1
m3	0	1	0	1	0	1	1	1	0
m4	0	0	1	0	0	0	0	0	0
m5	0	1	0	0	0	0	0	0	0
m6	0	0	1	0	0	0	0	0	0
m7	0	0	1	0	0	0	0	0	0
m8	0	0	1	0	0	0	0	0	0
m9	0	1	0	0	0	0	0	0	0

Calculating Delta Table(2/4)



- Projection matrix (L X M = P)
  - P(e_1, C): entity e_1 has a link to an entity which is placed in class C

Internal projection matrix (P_{Int})



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	2
m6	0	0	0	0
m7	0	0	0	0
m8	0	0	0	0
m9	0	0	0	2

=

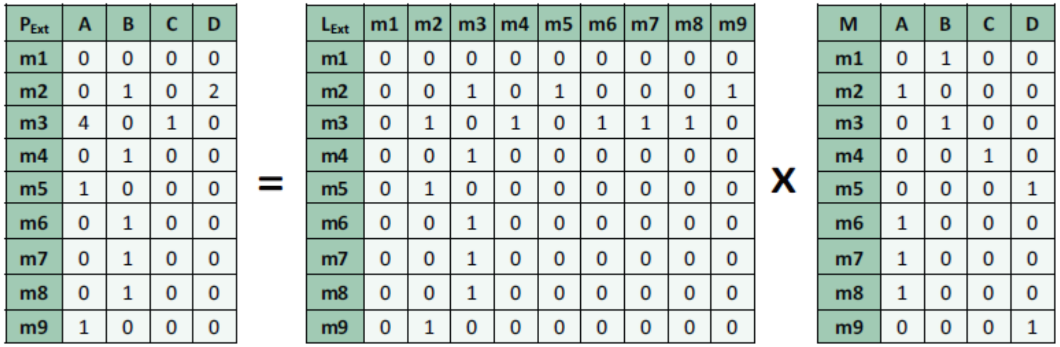
L_{Int}	m1	m2	m3	m4	m5	m6	m7	m8	m9
m1	0	0	1	0	0	0	0	0	0
m2	0	0	0	0	0	0	0	0	0
m3	1	0	0	0	0	0	0	0	0
m4	0	0	0	0	0	0	0	0	0
m5	0	0	0	0	0	0	0	0	2
m6	0	0	0	0	0	0	0	0	0
m7	0	0	0	0	0	0	0	0	0
m8	0	0	0	0	0	0	0	0	0
m9	0	0	0	0	2	0	0	0	0

X

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
m8	1	0	0	0
m9	0	0	0	1

(a) $P_{Int} = L_{Int} \times M$

External projection matrix (P_{Ext})



P_{Ext}	A	B	C	D
m1	0	0	0	0
m2	0	1	0	2
m3	4	0	1	0
m4	0	1	0	0
m5	1	0	0	0
m6	0	1	0	0
m7	0	1	0	0
m8	0	1	0	0
m9	1	0	0	0

=

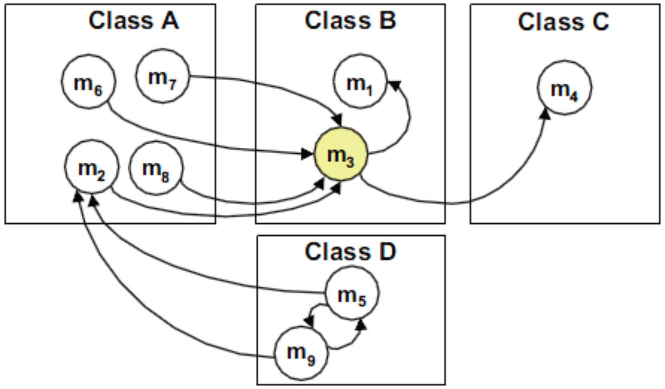
L_{Ext}	m1	m2	m3	m4	m5	m6	m7	m8	m9
m1	0	0	0	0	0	0	0	0	0
m2	0	0	1	0	1	0	0	0	1
m3	0	1	0	1	0	1	1	1	0
m4	0	0	1	0	0	0	0	0	0
m5	0	1	0	0	0	0	0	0	0
m6	0	0	1	0	0	0	0	0	0
m7	0	0	1	0	0	0	0	0	0
m8	0	0	1	0	0	0	0	0	0
m9	0	1	0	0	0	0	0	0	0

X

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
m8	1	0	0	0
m9	0	0	0	1

(b) $P_{Ext} = L_{Ext} \times M$

Calculating Delta Table(3/4)



- Inverse function: $Inv(P_{Int})$
  - $P_{Int}[e_1, C_1] = N > 0$ → internal link(s) exists from entity e_1 to class C_1

→ Therefore, moving the entity e_1 to other classes will potentially increase the external link(s) in the system

  - To simulate the effect of the application of moving entities

→ We use the inverse function to inverse the values of entries in P_{Int}
∴ $P_{Int}[e_1, C_1] \leftarrow 0$, $P_{Int}[e_1, c\ (c \in \text{all classes} \ \&\ c \neq C_1)] \leftarrow N$

Internal projection matrix (P_{Int})

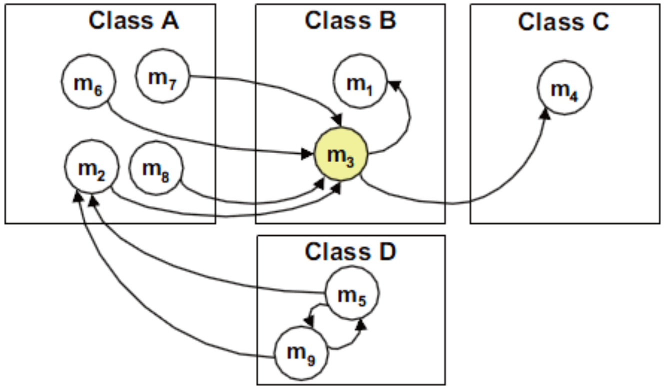
P_{Int}	A	B	C	D
m1	0	1	0	0
m2	0	0	0	0
m3	0	1	0	0
m4	0	0	0	0
m5	0	0	0	2
m6	0	0	0	0
m7	0	0	0	0
m8	0	0	0	0
m9	0	0	0	2

Inverse func.

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	2	2	2	0
m6	0	0	0	0
m7	0	0	0	0
m8	0	0	0	0
m9	2	2	2	0

Calculating Delta Table(4/4)



- Formulation
  - $P_{Int} = L_{Int} \times M$; $P_{Ext} = L_{Ext} \times M$; **$D = Inv(P_{Int}) - P_{Ext}$**

Delta Table (D)

D	A	B	C	D
m1	1	0	1	1
m2	0	-1	0	-2
m3	-3	0	0	1
m4	0	-1	0	0
m5	1	2	2	0
m6	0	-1	0	0
m7	0	-1	0	0
m8	0	-1	0	0
m9	1	2	2	0

=

$Inv(P_{Int})$	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	2	2	2	0
m6	0	0	0	0
m7	0	0	0	0
m8	0	0	0	0
m9	2	2	2	0

–

P_{Ext}	A	B	C	D
m1	0	0	0	0
m2	0	1	0	2
m3	4	0	1	0
m4	0	1	0	0
m5	1	0	0	0
m6	0	1	0	0
m7	0	1	0	0
m8	0	1	0	0
m9	1	0	0	0

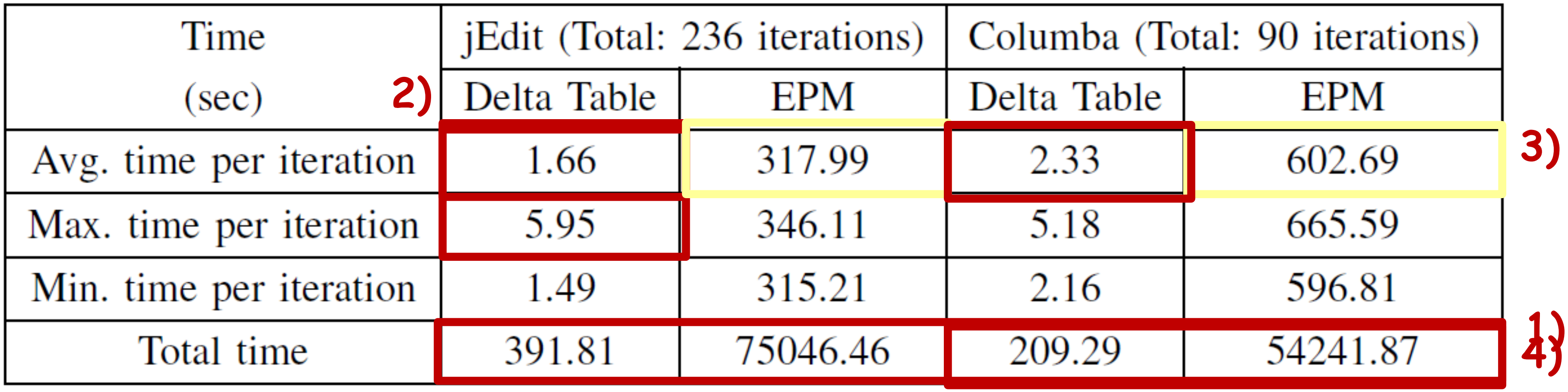
(c) $D = Inv(P_{Int}) - P_{Ext}$

Evaluation

- Research questions
  1) ***Efficiency***: By how much our method is efficient for assessing the impact of refactoring candidates?
  2) ***Usefulness***: Does the refactoring identification approach based on our method help improve maintainability?
- Comparators
  - Delta Table (our approach) vs. EPM [Tsantalis et al. 2009]
- Experimental subjects

Name (Version)	**jEdit** (jEdit-4.3)	**Columba** (Columba-1.4)
Type	Text editor	Email clients
Class #	952	1506
Methods #	6487	8745
Attributes #	3523	3967

Results: Total Time



Time (sec)	jEdit (Total: 236 iterations)		Columba (Total: 90 iterations)	
	Delta Table	EPM	Delta Table	EPM
Avg. time per iteration	1.66	317.99	2.33	602.69
Max. time per iteration	5.95	346.11	5.18	665.59
Min. time per iteration	1.49	315.21	2.16	596.81
Total time	391.81	75046.46	209.29	54241.87

1) Total time : our approach (delta table) < approach with EPM

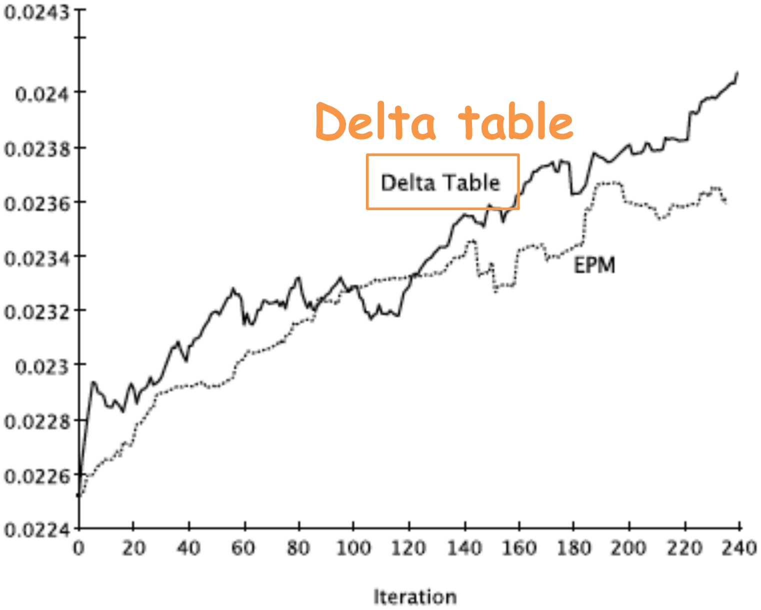
2) Max. time per iteration in our approach is the time taken for the first iteration (e.g., constructing design model, link and membership matrices)

3) As system become larger (jEdit: 952, Columba: 1506 classes), computation time is increased

4) Rate of increased time with respect to the system size is much less in our approach (e.g., for jEdit at 90 iterations, our approach: 154.63[sec], approach with EPM: 28,622[sec])

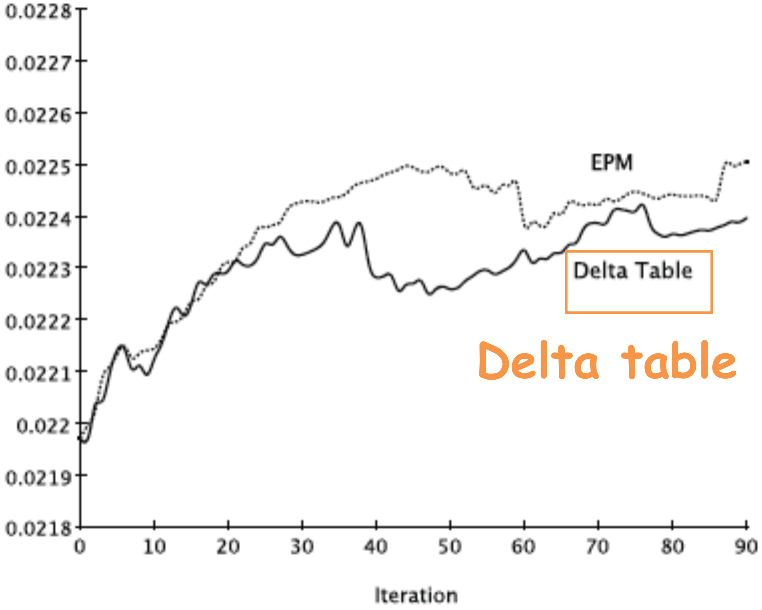
Results: Maintainability Improvement

- jEdit



(c) Maintainability evaluation function [2]

- Columba



(c) Maintainability evaluation function [2]

- The values of maintainability evaluation functions for our approach are increased in both projects (jEdit and Columba)

Conclusion and Future Work

- Summary
  - Propose an efficient method for assessing the impact of refactoring candidates faster

- Future Work
  - *Correlation analysis with the existing metrics*
  - Trade-off analysis between precision and speed
  - Scalability tests
    - To investigate the capability of assessing a large number of refactoring candidates

References

- [Harman et al. 2013] Dynamic adaptive search based software engineering needs fast approximate metrics, in Proceedings of the 4th International Workshop on Emerging Trends in Software Metrics.
- [Tsantalis et al. 2009] Identification of move method refactoring opportunities, IEEE Transactions on Software Engineering.
- [O'Keeffe et al. 2008] Search-based refactoring for software maintenance, The Journal of Systems & Software.
- [Lee et al. 2011] Automated scheduling for clone-based refactoring using a competent GA, Softw., Pract. Exper.
- [Seng et al. 2006] 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.
- [Han et al. 2013] Dynamic profiling-based approach to identifying cost-effective refactorings, Information Software and Technology
- [Bonja et al. 2006] C. Bonja, E. Kidanmariam, Metrics for class cohesion and similarity between methods, in: Proceedings of the 44th Annual Southeast Regional Conference

Thank you.