A review of Machine Learning Approaches for Move Method Refactoring Recommendation

Liviu-Stefan Neacsu-Miclea
ICA 256/2

Introduction

- Software entropy leads to gradual design degradation.
- Refactoring improves structure without changing behavior.
- Move Method addresses Feature Envy and cohesion issues.
- Manual identification is difficult in large systems.
- Automation is essential for scalable, consistent maintenance.

Motivation

- Move Method is a complex search task.
- SBSE frames refactoring as optimization.
- Need for semantic-aware methods beyond simple metrics.

Related Work
Qmove - Couto et al. (2018)

- Relies on six QMOOOD quality attributes.
- Evaluates the gain of each candidate Move Method operation.
- Uses a greedy search to pick best metric-improving moves.
- Performs well on Feature Envy cases compared to older tools.
- Limited by metric expressiveness; can generate false positives.

Table 1: Design Metrics for Design Properties

Design Metric	Design Property
DSC (Design Size in Classes)	Size
NOH (Number of Hierarchies)	Hierarchies
ANA (Average Number of Ancestors)	Abstraction
DAM (Data Access Metrics)	Encapsulation
DCC (Direct Class Coupling)	Coupling
CAM (Cohesion Among Methods of Class)	Cohesion
MOA (Measure of Aggregation)	Composition
MFA (Measures of Functional Abstraction)	Inheritance
NOP (Number of Polymorphic Methods)	Polymorphism
CIS (Class Interface Size)	Messaging
NOM (Number of Methods)	Complexity

Table 2: Equations for Quality Attributes

Quality Attribute	Equation
Reusability	-0.25*Coupling +0.25*Cohesion +0.5*Messaging +0.5*Size
Flexibility	+0.25*Encapsulation -0.25*Coupling +0.5*Composition +0.5*Polymorphism
Understandability	-0.33*Abstraction +0.33*Encapsulation -0.33*Coupling +0.33*Cohesion -0.33*Polymorphism -0.33*Complexity -0.33*Size
Functionality	+0.12*Cohesion +0.22*Polymorphism +0.22*Messaging +0.22*Size +0.22*Hierarchies
Extendibility	+0.5*Abstraction -0.5*Coupling +0.5*Inheritance +0.5*Polymorphism
Effectiveness	+0.2*Abstraction +0.2*Encapsulation +0.2*Composition +0.2*Inheritance +0.2*Polymorphism

Algorithm 1: Proposed Approach Algorithm

```
1  Input: methods, a list with every method and their respective class from the
   analyzed system
2  Output: recommendations, an ordered sequence of Move Method refactoring
   that can be applied to the analyzed system
3  begin
4      potRefactor := ∅
5      currentMetrics := calculateMetrics()
6      for each method m in methods do
7          if m can be automatically refactored to a class C then
8              potRefactor := potRefactor + {m, C}
9          end
10     end
11     candidates := ∅
12     metrics := ∅
13     while potRefactor ≠ ∅ do
14         for each refactoring ref in potRefactor do
15             applyRefactoring(ref)
16             metrics := calculateMetrics()
17             undoRefactoring(ref)
18             if fitness(metrics) > fitness(currentMetrics) then
19                 candidates := candidates + {ref, metrics}
20             end
21         end
           /* find the refactoring with the best metrics */
22         bestRefactoring := maxMetrics(candidates)
23         applyRefactoring(bestRefactoring)
24         potRefactor := potRefactor \ {bestRefactoring}
25         recommendations := recommendations + {bestRefactoring}
26         currentMetrics := bestRefactoring.metrics
27     end
28 end
```

Related Work
Methodbook – Batova et al. (2013)

- Introduces semantic analysis through Relation Topic Models.
- Treats classes and methods as text documents with latent topics.
- Recommends moves based on conceptual similarity, not metrics.
- Produces fewer but more meaningful suggestions.
- Sensitive to naming quality and comment sparsity.

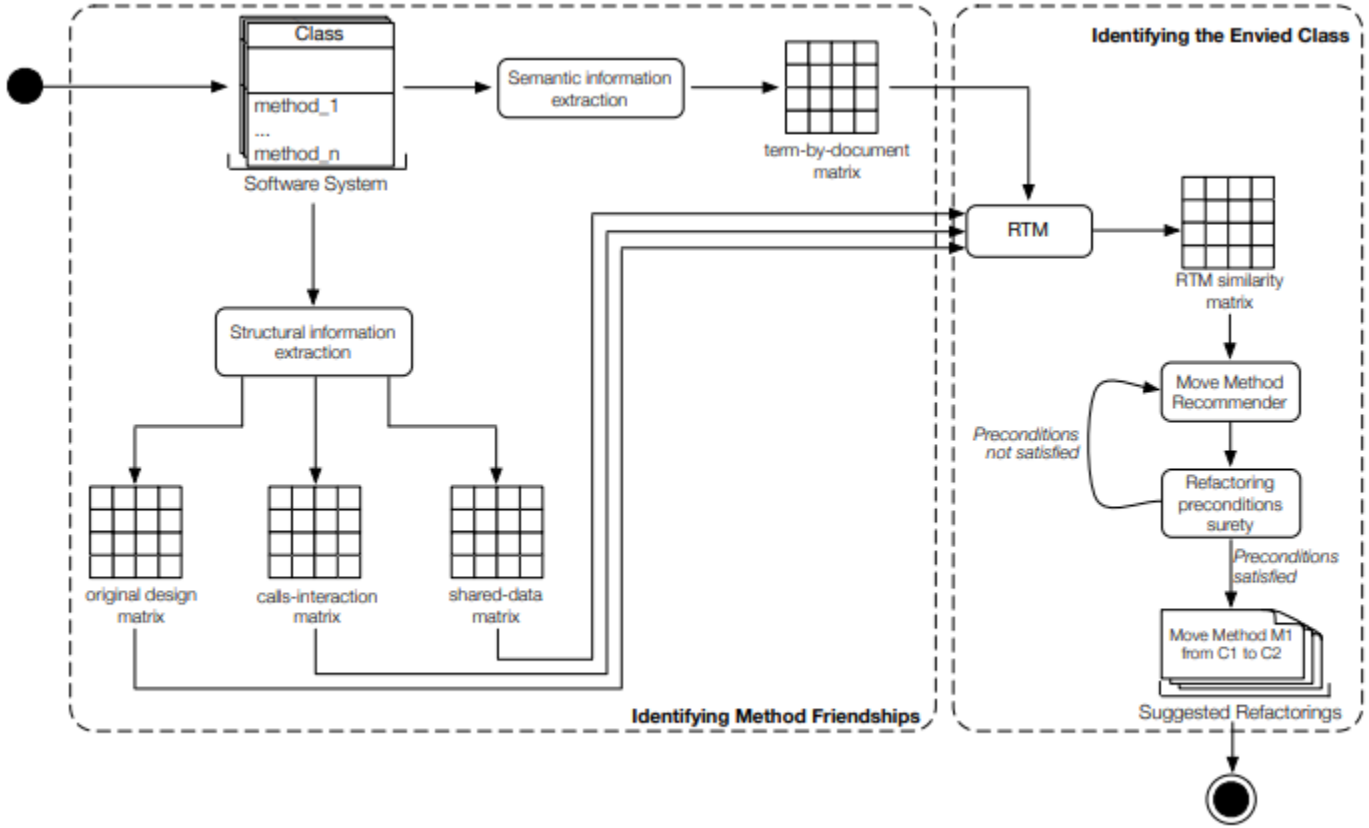

Fig. 1: The process used by Methodbook to identify move method refactoring operations

Related Work
PathMove – Kurbatova et al. (2020)

- Learns structural representations via AST path extraction.
- Inspired by code2vec: code transformed into path embeddings.
- Uses SVM classifier trained on real refactoring examples.
- Captures fine-grained syntactic relationships between entities.
- Outperforms older tools like JMove and JDeodorant in precision.

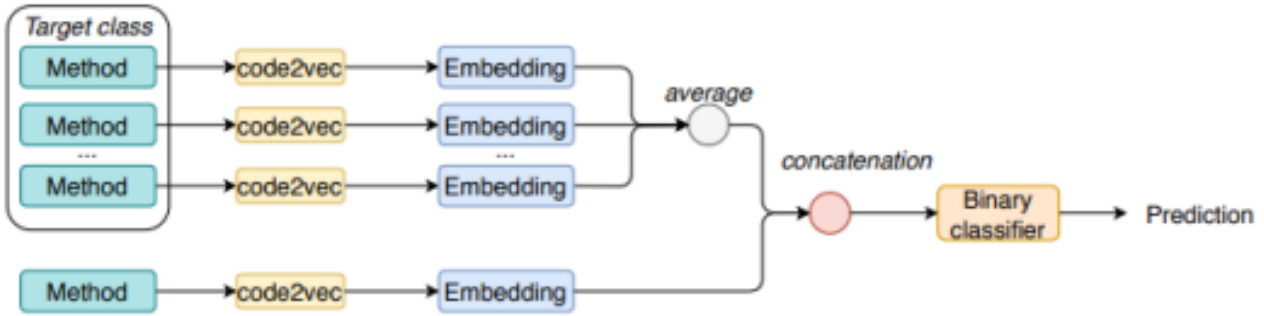

Figure 1: Path move classifier workflow

Related Work
Rmove – Cui et al. (2022)

- Hybrid model combining structural and semantic embeddings.
- Builds method dependency graphs for richer structural context.
- Extracts features from code text and concatenates them.
- Tests multiple ML models, including traditional and deep neural networks.
- Achieves highest precision among compared approaches.

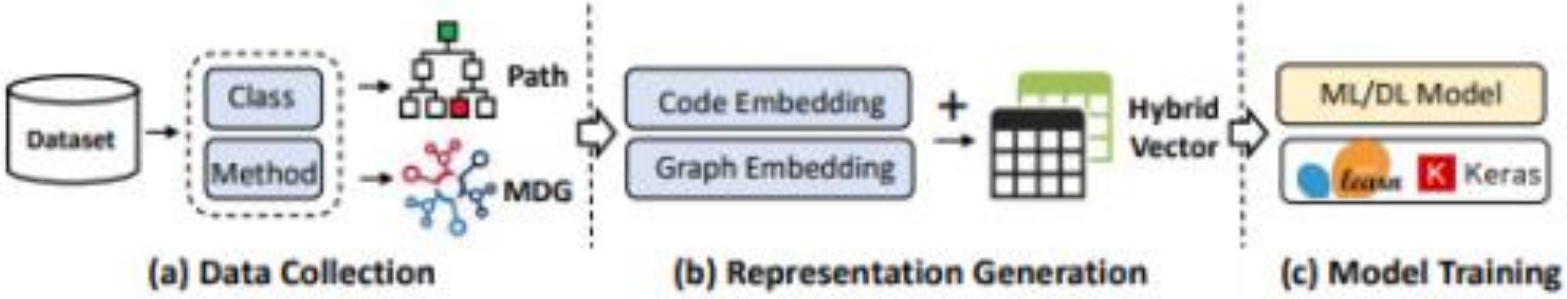

Figure 2: RMove method pipeline

Discussion

- Strong shift from metrics → topics → learned representations.

- Modern methods capture nuances metrics cannot express.
- Liminations:
  - Data-hungry – Deep models require large, curated training datasets.
  - Black-box – Explainability remains a challenge for developer trust.
  - Costs – Computational cost is higher than traditional approaches.

Conclusions

- Move Method recommendation evolved significantly.
- Hybrid models currently deliver best performance.
- Explainable models will be crucial for adoption.
- IDE integration offers potential for context-aware suggestions.
- LLMs may enable multi-step, intent-driven refactoring workflows.

Thank you!

References

- Gabriele Bavota, Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. Methodbook: Recommending move method refactorings via relational topic models. IEEE Transactions on Software Engineering, 40(7):671–694, 2013.
- Christian Marlon Souza Couto, Henrique Rocha, and Ricardo Terra. A qualityoriented approach to recommend move method refactorings. In Proceedings of the XVII Brazilian Symposium on Software Quality, pages 11–20, 2018
- Di Cui, Siqi Wang, Yong Luo, Xingyu Li, Jie Dai, Lu Wang, and Qingshan Li. Rmove: recommending move method refactoring opportunities using structural and semantic representations of code. In 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 281–292. IEEE, 2022.
- Zarina Kurbatova, Ivan Veselov, Yaroslav Golubev, and Timofey Bryksin. Recommendation of move method refactoring using path-based representation of code. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pages 315–322, 2020.