Decomposing Object-Oriented Class Modules Using an Agglomerative Clustering Technique

Marios Fokaefs, Joerg Sander
Department of Computing Science
University of Alberta, Canada

Nikolaos Tsantalis, Alexander Chatzigeorgiou
Department of Applied Informatics
University of Macedonia, Greece

Outline

- Introduction
- Methodology
- Evaluation
- Future Work
- Conclusions

Problem

- Software increases in size and complexity.
- Programmers focus on functionality rather than design and maintenance.

➔Large and complex classes that are prone to changes (God Classes)

- A class
  - must implement only one concept
  - must have only one reason to change

Solution: Extract Class refactoring

Identifying God Classes

- Metrics (Trifu& Marinescu05, Tahvildari& Kontogiannis03, DuBois et al.04)
  - Require human interpretation
  - Cannot be fully automated
  - Do not suggest solutions
- Names (Demeyer et al.02)
- History of changes (Demeyer et al.02)
- Visualization (Simon et al.01, Joshi and Joshi09)
  - Requires human interpretation
  - Not scalable
- Conceptual criteria (De Lucia et al.08)
  - Highly dependent on developers' naming patterns

Our solution

1. Identifies Extract Class opportunities
   Clustering recognizes conceptually meaningful groups of similar entities
2. Suggests a set of refactoring solutions
   Filtering eliminates behavior-affecting changes
3. Ranks solutions based on their impact on the design quality
4. Identifies new concepts.

Clustering

- Entities : Attributes and methods
- Original set of entities : a class
- Cluster : extracted entities/a concept/a class

- Algorithm?
- Distance metric? Distance threshold?
- Hierarchical
  - Linkage Method?
  - Distance Threshold?

Clustering (Distance metric)

- Jaccard distance
- Entity sets (Tsantalis and Chatzigeorgiou09)
  - Attribute: All the methods that access the attribute
  - Method: All the methods that access or are accessed by the method and all the attributes accessed by the method.
  - The entity itself
    - $d_{ij}=0$ iff i=j
  - References (as local attributes)
  - Do not include foreign entities

$$d_{\alpha,\beta} = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Clustering (Linkage Method)

- Linkage method is the criterion to merge the two closest clusters
- Single linkage: minimum distance between the entities of the two clusters.
  - Favors less coupled clusters (Anquetil&Lethbridge99)
  - **Our distance metric favors more cohesive clusters**
- Complete linkage: maximum distance
  - Favors more cohesive clusters (Anquetil&Lethbridge99)
- Average linkage: average distance

Clustering (Distance threshold)

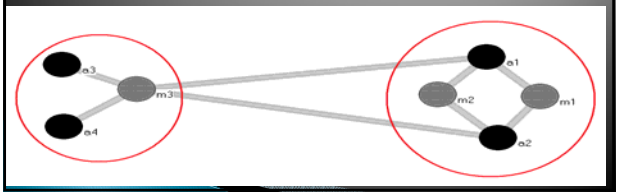
- The distance threshold is the cut-off value that determines the actual clusters
- We apply the algorithm for several values ranging from 0.1 to 0.9
- We exclude duplicate suggestions
  - i.e. When two clusters produced by two different thresholds contain exactly the same entities
- We include similar suggestions
  - i.e. When a cluster is a subset of another

Example (Source Code)

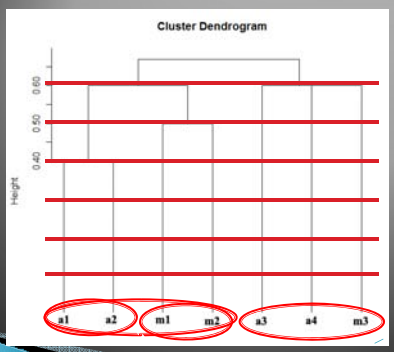
- a1=name
  - S_{a1}={name, changeJob, modifyName, getTelephoneNUmber}
- a2=job
  - S_{a2}={job, changeJob, modifyName, getTelephoneNumber}
- a3=officeAreaCode
  - S_{a3}={officeAreaCode, getTelephoneNumber}
- a4=officeAreaNumber
  - S_{a4}={officeAreaNUmber, getTelephoneNumber}
- m1=changeJob
  - S_{m1}={changeJob, job, name}
- m2=modifyName
  - S_{m2}={modifyName, job, name}
- m3=getTelephoneNumber
  - S_{m3}={getTelephoneNumber, officeAreaCode, officeAreaNumber}

Example (Distances)

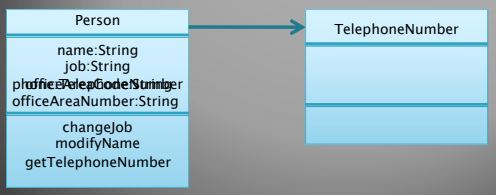
	a1	a2	a3	a4	m1	m2
a2	0.4					
a3	0.8	0.8				
a4	0.8	0.8	0.67			
m1	0.6	0.6	1	1		
m2	0.6	0.6	1	1	0.5	
m3	0.71	0.71	0.6	0.6	0.67	0.67



Example (Dendrogram)



Example (Resulted Refactoring)



Filtering the results

- Preconditions to preserve funtionality
  - The class to be extracted should contain
    - more than one entity.
    - at least one method.
- Preconditions to preserve behavior
  - The class to be extracted should NOT contain
    - a method that overrides any abstract or concrete method of the super class of the source class
    - a method that makes any super method invocations
    - a synchronized method

Ranking of results

- Entity Placement (Tsantalis &Chatzigeorgiou09)
  - Uses entity sets
  - Based on Jaccard distance
  - Combines the notions of cohesion and coupling
- $EntityPlacement_C = \frac{\sum_{e_i \in C} distance(e_i, C)}{|entities \in C|} \Big/ \frac{\sum_{e_i \notin C} distance(e_i, C)}{|entities \notin C|}$
- Entity Placement for the system is a weighted average of the Entity Placement values for all the classes
- The **lower** the EP value, the better the refactored design
  - The suggested refactorings are sorted by the EP value in ascending order

Virtual application

1. Create a new empty class
2. For each extracted entity change its origin class from the source class to the new class
3. Update the entity sets of all the entities that access or are accessed by the extracted entities
4. Insert the extracted entities in the entity set of the new class
5. Remove the extracted entities from the entity set of the source class

Experimental Evaluation

- 2 projects – Independent assessment
  - eRisk
    - Board game
    - Developed by undergraduate students
  - SelfPlanner
    - Intelligent web-based calendar application
    - Academic research project
    - Developed by a graduate student with experience in OO programming
- Questions:
  - Can you find a name for the cluster of entities suggested to be extracted?
  - Would you apply the suggested refactoring?

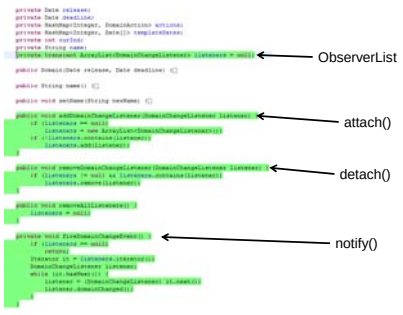
Discussion of Results (eRisk)

- Total Suggestions: 37
  - Assigned names: 28
  - Applied: 16
- High rate in ability to identify new concepts: **75,6%**
- Good rate in applying the refactoring: **43,2%**
- By-products
  - Reusability of new classes (e.g. Utility classes)
  - Extract superclass or interface opportunities
  - Move method/attribute opportunities
  - Extract inner class opportunity

Discussion of Results (SelfPlanner)

- Total Suggestions: 14
  - Assigned names: 12
  - Applied: 9
- High rate in ability to identify new concepts: **86%**
- Very good rate in applying the refactoring: **64%**
- By-product
  - The method was able to identify the Subject role in an Observer pattern instance.

SelfPlanner Example



Future Work

- Try and compare:
  - Different clustering algorithms
  - Different distance metrics
  - Different linkage methods
- Extend our evaluation process
  - Larger scale open source and commercial projects
  - Expert assessment
- Implement the application of Extract Class refactoring
- Complete the integration in JDeodorant

Conclusions

- We developed an effective method for
  - Identifying Extract Class opportunities
  - Using Hierarchical Agglomerative clustering
- We evaluated our method on two projects:
  - It identifies new **concepts** that can be extracted into new classes
  - It suggests **behavior** preserving solutions
  - It **ranks** the results to improve the user's understanding

Thank you

Questions?