AN EMPIRICAL INVESTIGATION INTO CODE SMELL ELIMINATION SEQUENCES FOR ENERGY EFFICIENT SOFTWARE

GREEN IT

Garima Dhaka (garima.dhaka1992@gmail.com) & Paramvir Singh (singhpv@nitj.ac.in)
Department Of Computer Science and Engineering
Dr B R Ambedkar National Institute Of Technology, Jalandhar
G T Road, Amritsar Bye - Pass Road, Jalandhar, Punjab, India 144011

Code Smell · Refactoring · Architecture · Energy

INTRODUCTION

- Information Technology (IT) sector has significant contribution toward human greenhouse gas emissions worldwide.
- Hence, one of the vital modern day challenges for researchers and industry practitioners is to optimize the energy consumption behavior of software applications.

RESEARCH QUESTIONS

RQ1 What energy consumption trends are observed across sample applications after individually removing god class, long method and feature envy code smells?

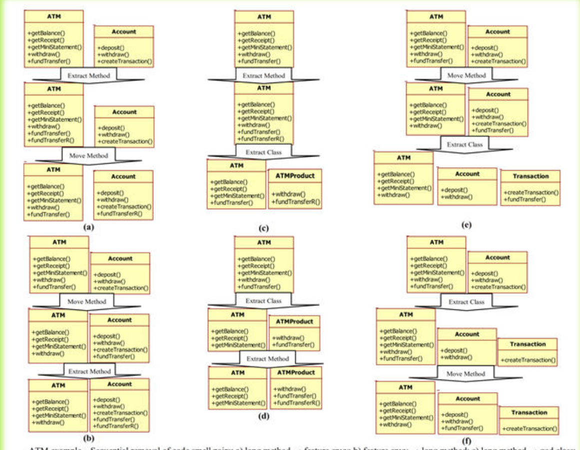
RQ2 Do the refactored versions obtained by removing the selected set of three code smells in different sequences yield different energy consumption behaviour?

RQ3 Do any particular code smell removal sequences yield minimum/maximum energy consumption values across all sample applications?

RQ4 Is there any relationship between architecture metrics and energy consumption of refactored versions of sample applications within the context of code smell removal?

MOTIVATION

- *Castillo and Piattini* [1] analyzed the impact of removing god class code on software energy consumption behavior that resulted in increased energy consumption.
- *Park et al.* [2] investigated the impact of applying single instances of various refactoring techniques on software energy consumption behaviour. We mapped our selected set of three code smells to their preferred refactoring techniques as defined in JDeodorant, see Figure 2).



ATM example – Sequential removal of code smell pairs; a) long method → feature envy; b) feature envy → long method; c) long method → god class; d) god class → long method; e) feature envy → god class; f) god class → feature envy

Figure 1: Motivational Example

*Code Smell	+Refactoring Technique	Energy Consumption (μJ)		Effect
		Original	*Refactored*	
Feature Envy	Move Method	70.066	69.786	↓
God Class	Extract Class	83.077	83.600	↑
Long Method	Extract Method	72.361	74.780	↑

Descriptions

Feature Envy (F): Occurs when method belonging to one class makes extensive use of another class.
God Class (G): Indicates the violation of single responsibility design principle. It occurs when class is too large and has lots of responsibilities.
Long Method (L): Occurs when a method has too many lines of code, making it hard to read and understand.

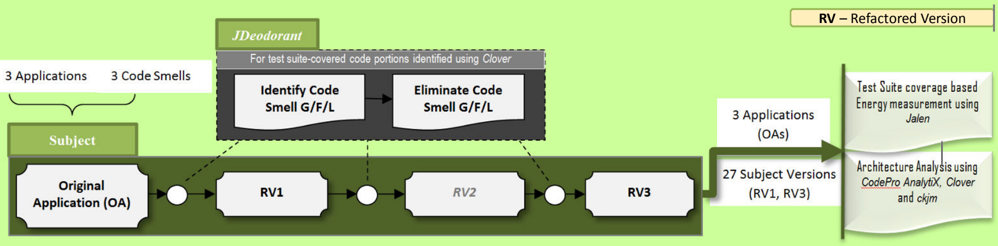
+*Move Method*: It creates a new method in the class (say, B), which the old method (say, in class A) is more frequently accessing. Then either move the entire old method from class A to class B or delegate the responsibility to the new method created in class B.
+*Extract Class*: It extracts some of the functionalities from the original class (generally god class) to a newly created class.
+*Extract Method*: It extracts some functionalities of the original method to a newly created method, and replaces those functionalities with a call to the newly created method.

Figure 2: Mapping code smells with energy consumption.

SAMPLE APPLICATIONS

Application	Classes	Test Cases	Cov. (%)
JHotDraw 6.2.0	552	3654	54.7
Commons BeanUtils 1.9.3	322	1660	78.3
Commons IO 2.6	262	1157	91.9

METHODOLOGY



EMPIRICAL RESULTS AND TRENDS

Ver. ► / ▼	O	FGL	FLG	GFL	GLF	LGF	LFG
LOC							
FGL	↑↑		↓↓	↑↑	↑↕	↑↑	↑↑
FLG	↑↑	↑↑		↑↑	↑↑	↑↑	↑↑
GFL	↑↑	↓↓	↓↓		↓↓	↓↓	↓↓
GLF	↑↑	↓↕	↓↓	↑↑		↓↕	↕↕
LGF	↑↑	↓↓	↓↓	↑↑	↑↕		↑↕
LFG	↑↑	↓↓	↓↓	↑↑	↕↕	↓↕	
NOC							
FGL	↑↑		↓↕	↑↕	↑↕	↑↕	↑↕
FLG	↑↑	↑↕		↑↑	↑↑	↑↑	↑↑
GFL	↑↑	↓↕	↓↓		↓|	↓↕	↓↕
GLF	↑↑	↓↕	↓↓	↑|		↓↕	↕↕
LGF	↑↑	↓↕	↓↓	↑↕	↑↕		↑↑
LFG	↑↑	↓↕	↓↓	↑↕	↕↕	↓↓	
NOM							
FGL	↑↑		↓↓	↑↑	↑↑	↑↑	↑↑
FLG	↑↑	↑↑		↑↑	↑↑	↑↑	↑↑
GFL	↑↑	↓↓	↓↓		↓↓	↓↓	↓↓
GLF	↑↑	↓↓	↓↓	↑↑		↓↓	↕↕
LGF	↑↑	↓↓	↓↓	↑↑	↑↑		↑↑
LFG	↑↑	↓↓	↓↓	↑↑	↕↕	↓↓	
CBO							
FGL	↑↑		↓↕	↑↑	↑↑	↑↑	↑↑
FLG	↑↑	↑↕		↑↑	↑↑	↑↑	↑↑
GFL	↑↑	↓↓	↓↓		↓↕	↓↕	↓↕
GLF	↑↑	↓↓	↓↓	↑↕		↓↕	↕↕
LGF	↑↑	↓↓	↓↓	↑↕	↑↕		↑↑
LFG	↑↑	↓↓	↓↓	↑↕	↕↕	↓↓	
CC							
FGL	↑↕		↓↕	↑↕	↑|	↑|	↑↕
FLG	↑↕	↑↕		↑↕	↑↕	↑↕	↑↕
GFL	↑↕	↓↕	↓↕		↓↕	↓↕	↓|
GLF	↑↕	↓|	↓↕	↑↕		↓|	↕↕
LGF	↑↕	↓|	↓↕	↑↕	↑|		↑↕
LFG	↑↕	↓↕	↓↕	↑|	↕↕	↓↕	

Metric ► / Smell ▼	LOC				NOC				NOM				CBO				CC			
	O	*F*	*G*	*L*	*O*	*F*	*G*	*L*	*O*	*F*	*G*	*L*	*O*	*F*	*G*	*L*	*O*	*F*	*G*	*L*
F	↓↑		↓↓	↓↕	↓|		↓↓	↓|	↓↑		↓↓	↓↓	↓↕		↓↓	↓↕	↓↕		↓↕	↓↕
G	↑↑	↑↑		↑↕	↑↑	↑↑		↑↑	↑↑	↑↑		↑↕	↑↑	↑↑		↑↑	↑↕	↑↕		↑↕
L	↑↑	↑↕	↓↕		↑|	↑|	↓↓		↑↑	↑↑	↓↓		↑↕	↑↕	↓↓		↑↕	↑↕	↓↕	

Here, ↑ represents increasing trend, ↓ represents a decreasing trend, | represents no change in values, and ↕ represents no specific trend. For each cell value, the first arrow represents the energy trend (across all applications) for the refactored version in the related row header against the version in the related column header. Similarly, the second arrow represents the respective metric trend.

FUTURE DIRECTIONS

- More software from varied domains can be experimented to validate our findings.
- Other refactoring techniques for removing code smells can be tested for energy efficiency.
- Sequential elimination of other code smells for energy efficiency of software.
- Ideal tradeoff between software maintainability and sustainability can be empirically derived.

CONCLUSIONS

- Uniform energy consumption trends across all applications are observed when refactored versions are created by eliminating all instances of individual code smells.
- Refactored versions generated by applying GFL sequence yield minimum energy consumption whereas LFG sequence lead to maximum energy consumption as compared to those generated by applying other sequences.
- A number of relationships between architecture metrics and energy consumption are also observed, signifying the impact of such metrics on software energy consumption.

REFERENCES

1. R. Perez-Castillo and M. Piattini, "Analyzing the harmful effect of god class refactoring on power consumption," IEEE Software, vol. 31, no. 3, pp. 48-54, 2014.
2. J. J. Park, J. E. Hong, and S. H. Lee, "Investigation for Software Power Consumption of Code Refactoring Techniques," In Int'l Conf. on Software Engg. and Knowledge Engg., pp. 717-722. 2014.

PLEASE SHARE YOUR FEEDBACK

For more information visit to www.pvsingh.com/g_dhaka