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

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

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

- What energy consumption trends are observed across sample applications after individually removing god class, long method and feature envy code smells?
- Do the refactored versions obtained by removing the selected set of three code smells in different sequences yield different energy consumption behaviour?
- Do any particular code smell removal sequences yield minimum/maximum energy consumption values across all sample applications?
- 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).

SAMPLE APPLICATIONS

<table>
<thead>
<tr>
<th>Application</th>
<th>Classes</th>
<th>Test Cases</th>
<th>Cov. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw 6.2.0</td>
<td>552</td>
<td>3654</td>
<td>54.7</td>
</tr>
<tr>
<td>Commons BeanUtils 1.9.3</td>
<td>322</td>
<td>1660</td>
<td>78.3</td>
</tr>
<tr>
<td>Commons IO 2.6</td>
<td>262</td>
<td>1157</td>
<td>91.9</td>
</tr>
</tbody>
</table>

METHODOLOGY

- For each code smell, we have identified three refactoring techniques that can be applied to remove it.
- We then refactored the sample applications using these techniques and measured the energy consumption for each refactored version.

EMPIRICAL RESULTS AND TRENDS

- 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.

CONCLUSIONS

- 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.

FUTURE DIRECTIONS

- Additional refactoring techniques for removing code smells can be tested for energy efficiency.
- More real-world applications can be experimented to validate the findings.

REFERENCES