Refactoring for Software Isolation

NOVEMBER 16, 2022

James Ivers
Principal Engineer

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.
©2022
Periodic Refactoring Is Key to Keeping Code Healthy

Our ability to work with software significantly influences project cost, schedule, time to field, and other concerns. When the structure of software inhibits development priorities, software needs to be refactored to enable efficient, timely delivery of capabilities.

In this project, we are creating automation that dramatically accelerates an important form of large-scale refactoring.
Software Is Never Done

Change is inevitable
• Requirements change
• Business priorities change
• Programming languages change
• Deployment environments change
• Technologies and platforms change
• Interacting systems change
• …
Refactoring Gets Harder at Scale

Large-Scale Refactoring
- Changes require substantial effort and coordination among multiple teams of developers
- Measured in staff months to years
- Architecture changes and non-local affects

As scale increases,
- cross-team coordination increases
- technical risk increases
- cost and schedule impacts increase
- likelihood of securing funding decreases

Refactoring Sprints
- Changes made by a single team
- Often time-boxed (e.g., a two-week sprint)
- Effects limited to a single service
- E.g., 20% reserve to remove technical debt

“Floss Refactoring”
- Changes made by a single developer
- Intermingled with feature development
- Measured in minutes to hours of time
- Local affects

As scale increases,
- cost and schedule impacts increase
- likelihood of securing funding decreases
Large-Scale Refactoring (LSR) in Industry

• Most respondents had performed LSR multiple times
• Most systems on which they had performed LSR had undergone LSR multiple times
• Mean of 1,500 staff days to perform LSR

We surveyed 107 industry practitioners to understand the state of the practice.


# Tools Used in Large-Scale Refactoring

What tools are used for large-scale refactoring?

<table>
<thead>
<tr>
<th>Tool</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>54.3%</td>
</tr>
<tr>
<td>Other</td>
<td>50.0%</td>
</tr>
<tr>
<td>Manual Efforts</td>
<td>23.9%</td>
</tr>
<tr>
<td>Testing Tools</td>
<td>21.7%</td>
</tr>
<tr>
<td>Code Smells Analysis</td>
<td>15.2%</td>
</tr>
<tr>
<td>Continuous Integration</td>
<td>13.0%</td>
</tr>
<tr>
<td>Text Editor</td>
<td>6.5%</td>
</tr>
<tr>
<td>Version Control/Issue Tracker</td>
<td>6.5%</td>
</tr>
<tr>
<td>IDE Refactoring Features</td>
<td>6.5%</td>
</tr>
<tr>
<td>Refactoring Tool</td>
<td>4.3%</td>
</tr>
<tr>
<td>Visual Modeling</td>
<td>2.2%</td>
</tr>
</tbody>
</table>

Refactoring tools are not widely used in LSR

- < 10% reported using tools designed for refactoring
- Manual effort and custom scripts were reported more often than refactoring tools
Our Solution: An Automated Refactoring Assistant

We have developed an automated refactoring assistant that improves software structure for several common forms of change that involve software isolation.

Our goal: Complete software isolation with only 20% of the effort it takes today.

Software Isolation Is a Recurring Challenge

In software isolation, we seek to improve its modularity, reduce future development costs, and enable its use in new contexts.

Examples include
- strategic reuse
- rehosting on new platforms
- moving to the cloud

There is structure in this data, but that structure doesn't always let us do what we need to do.

A “simple” view of only 68K LOC.
Building on Search-Based Software Engineering

Search-based software engineering frames software engineering problems as optimization problems.

We defined a metric for software isolation, problematic couplings, that enables automated search for refactoring recommendations.

Algorithm 1: Summary of NGSA-II

Input: A dependency graph (G) of the software to be refactored, marked with the isolation goal

Output: A Pareto front of individuals, each of which contains a list of refactorings

1. \( P = \text{build\_initial\_pop}(G) \)
2. \( A = \emptyset \)
3. while termination condition not reached do
4. \( \text{assess\_fitness}(P) \)
5. \( P = P \cup A \)
6. \( \text{sort\_pop}(P) \)
7. \( A = \text{best\_of}(P) \)
8. \( P = \text{make\_new\_pop}(A) \)
9. end

Building on Search-Based Software Engineering

1. Develop extensible graph representation for multiple languages

3. Align semantics with changing nature of software problem to be solved

Algorithm 1: Summary of NGSA-II

Input: A dependency graph (G) of the software to be refactored, marked with the isolation goal

Output: A Pareto front of individuals, each of which contains a list of refactorings

1. $\text{build\_initial\_pop}(G)$
2. $A = \emptyset$
3. while termination condition not reached do
4. \hspace{1em} $\text{assess\_fitness}(P)$
5. \hspace{1em} $P = P \cup A$
6. \hspace{1em} $A = \text{best}\_o(P)$
7. \hspace{1em} $\text{sort}\_\text{pop}(P)$
8. $A = \text{make\_new\_pop}(A)$
9. end

2. Formalize project-specific refactoring goals

4. Define a novel fitness function to focus on modularity improvements

5. Formalize refactorings for use in change operations
Multi-objective Optimization

Our refactoring assistant generates a collection of Pareto-optimal solutions that represent trade-offs among competing objectives.
Current Capabilities

We now support refactoring for two programming languages: Java and C#.

Our refactoring assistant
• scales to at least 1.2M SLOC
• generates recommendations that solve the majority of each software isolation problem
Refactoring Criteria

Solving the right problem in a way that developers will accept is key to success.

We are studying the criteria that matter to developers when refactoring:
Looking Ahead

In the coming year, we will

• integrate a wider range of criteria through
  - enhanced preference expression
  - additional objectives
  - algorithm integration via penalty mechanisms and selection bias
• add refactorings and tune Java performance
• pilots with production code

For more information, go to https://www.sei.cmu.edu/go/knot

Contact us at sei-knot@sei.cmu.edu if you are interested in partnering with us.
Our Team

James Ivers  
*Principal Investigator, Principal Engineer, Software Solutions Division*

Chris Seifried  
*Associate Engineer, Software Solutions Division*

Jonny Loungani  
*Associate Software Engineer, Software Solutions Division*

Ipek Ozkaya  
*Tech. Director, Engineering Intelligent Software Systems, Software Solutions Division*

Mario Benitez  
*Software Architect, Software Solutions Division*

Tamara Marshall-Keim  
*Team Lead of Technical Communications, OCOS & CIO*

Greg Such  
*Program Development Manager, Software Solutions Division*

Andrew Kotov  
*Software Architect, Software Solutions Division*

Collaborators

- Marouane Kessentini, Oakland University
- Khouloud Gaaloul, University of Michigan
- Esther Bae, Carnegie Mellon University
- Owen Donovan, Pennsylvania State University