Carnegie Mellon University Software Engineering Institute

Refactoring for Software Isolation

NOVEMBER 16, 2022

James Ivers
Principal Engineer

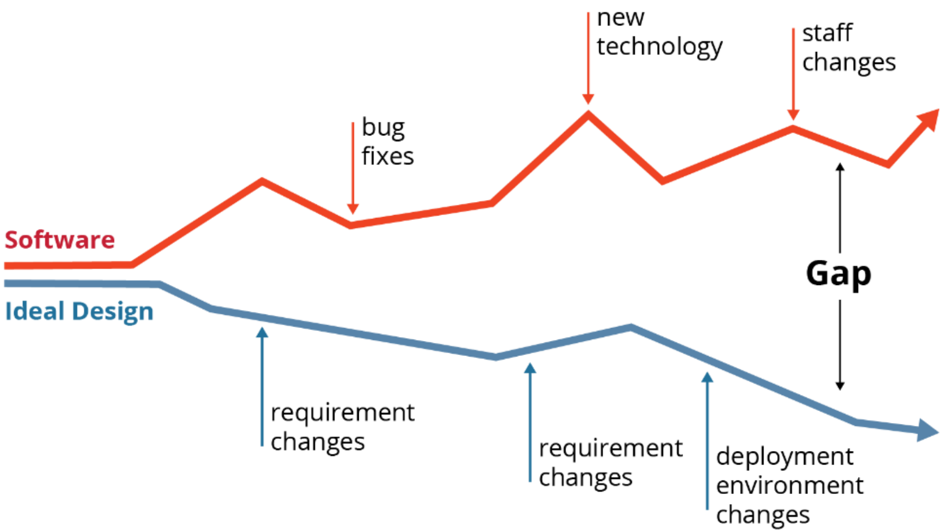
Periodic Refactoring Is Key to Keeping Code Healthy

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.

Carnegie Mellon University Software Engineering Institute

Software Is Never Done



Change is inevitable

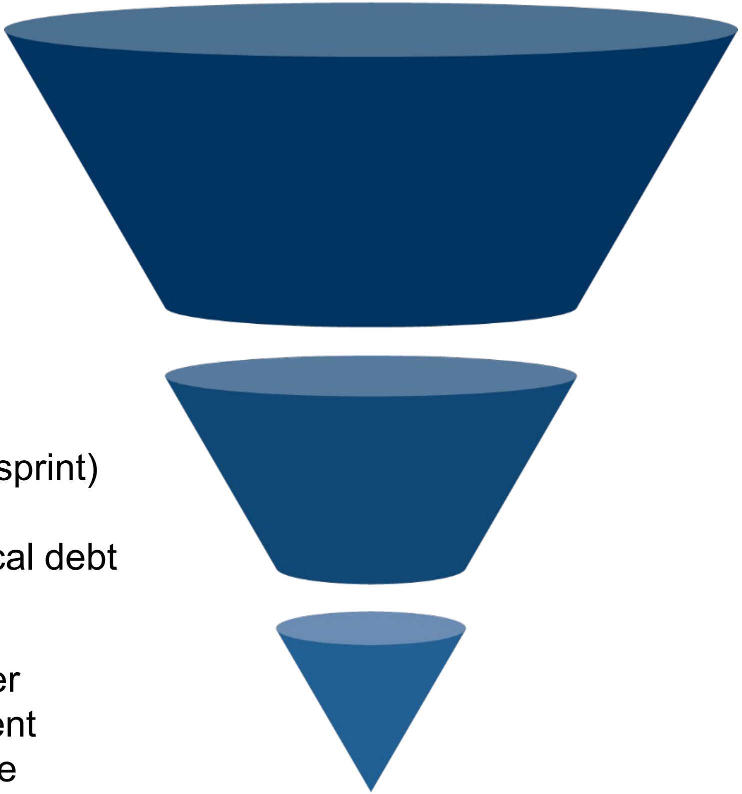
- Requirements change
- Business priorities change
- Programming languages change
- Deployment environments change
- Technologies and platforms change
- Interacting systems change
- ...

Refactoring Gets Harder at Scale

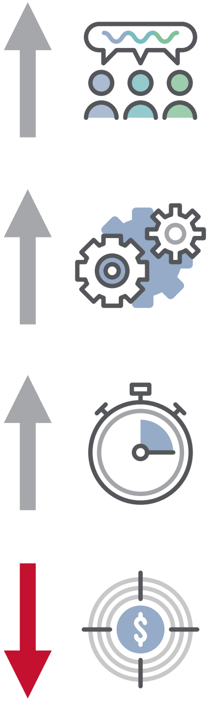
Large-Scale Refactoring
- Changes require substantial effort and coordination among multiple teams of developers
- Measured in staff months to years
- Architecture changes and non-local affects

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,



cross-team coordination increases

technical risk increases

cost and schedule impacts increase

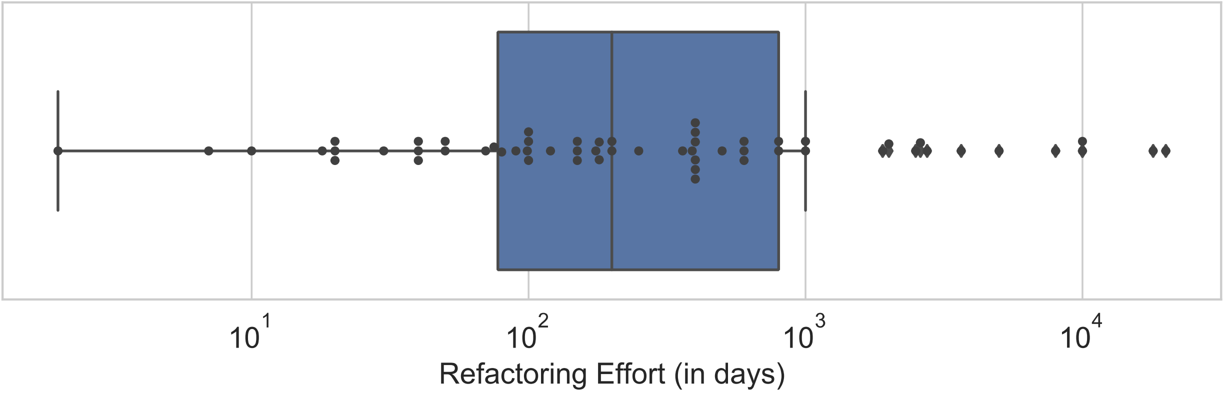
likelihood of securing funding *decreases*

Large-Scale Refactoring (LSR) in Industry

Carnegie Mellon University Software Engineering Institute

- 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.

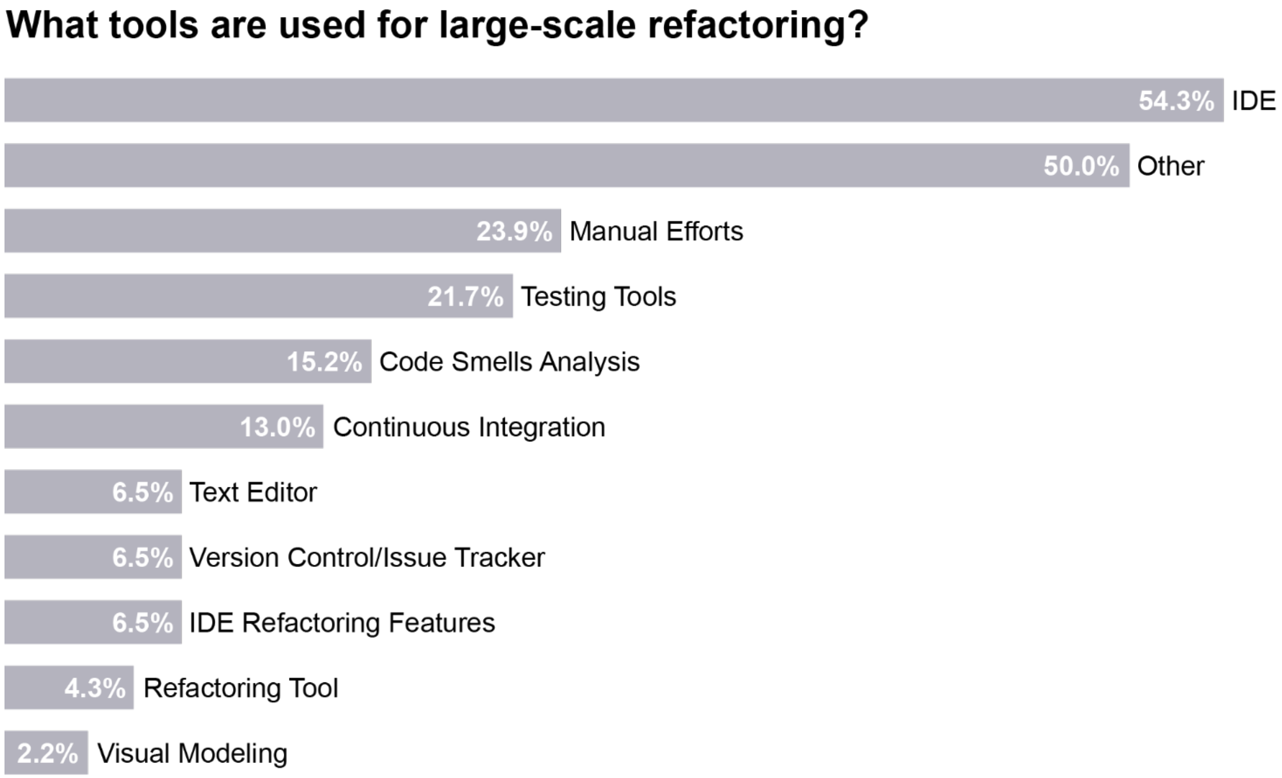


J. Ivers, R. Nord, I. Ozkaya, C. Seifried, C. Timperley, M. Kessentini. **Industry Experiences with Large-Scale Refactoring.** *Foundations of Software Engineering: Software Engineering in Practice* (ESEC/FSE). November 2022.

J. Ivers, R. Nord, I. Ozkaya, C. Seifried, C. Timperley, M. Kessentini. **Industry's Cry for Tools That Support Large-Scale Refactoring.** *Intl. Conference on Software Engineering: Software Engineering in Practice* (ICSE-SEIP). May 2022.

Tools Used in Large-Scale Refactoring

Carnegie Mellon University Software Engineering Institute



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

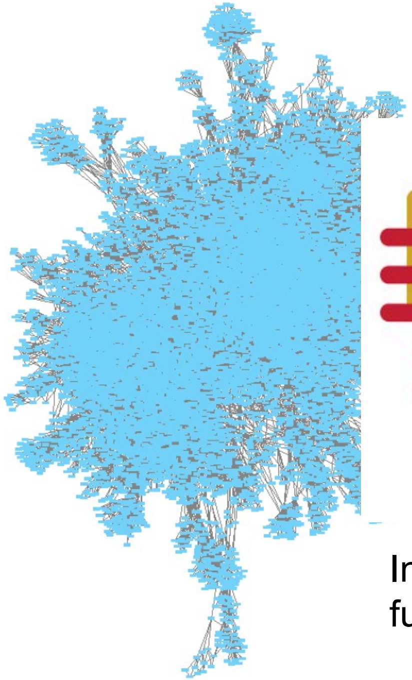
We have developed an automated refactoring assistant that improves software structure for several common forms of change that involve software isolation.

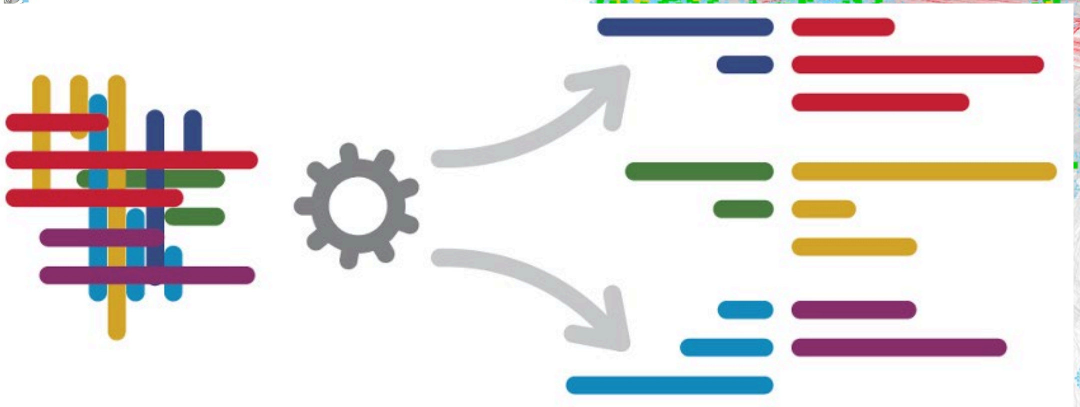
Our goal: Complete software isolation with only **20% of the effort** it takes today.

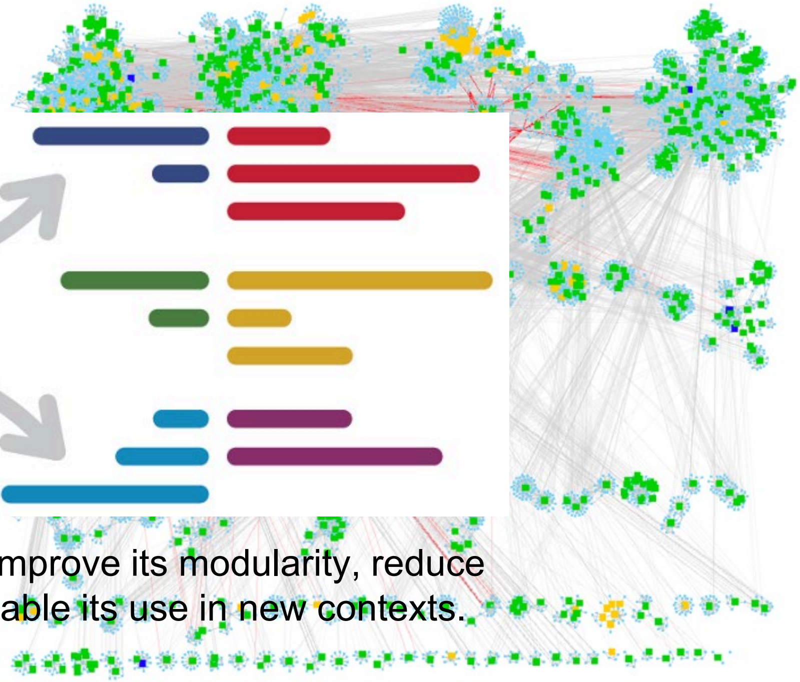
J. Ivers, C. Seifried, I. Ozkaya. **Untangling the Knot: Enabling Architecture Evolution with Search-Based Refactoring.** *19th IEEE International Conference on Software Architecture (ICSA 2022)*. March 2022.

Carnegie Mellon University Software Engineering Institute

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.

A “simple” view of only 68K LOC.

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.

Building on Search-Based Software Engineering

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

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

```
P = build_initial_pop(G)
A = ∅
while termination condition not reached do
    assess_fitness(P)
    P = P ∪ A
    sort_pop(P)
    A = best_of(P)
    P = make_new_pop(A)
end
```

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

K. Deb, A. Pratap, S. Agarwal, T. Meyarivan. **A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.** *IEEE Transactions on Evolutionary Computation*. 2002.

Building on Search-Based Software Engineering

Carnegie Mellon University Software Engineering Institute

1. Develop extensible graph representation for multiple languages

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

```
P = build_initial_pop(G)
A = ∅
while termination condition not reached do
    assess_fitness(P)
    P = P ∪ A
    sort_pop(P)
    A = best_of(P)
    P = make_new_pop(A)
end
```

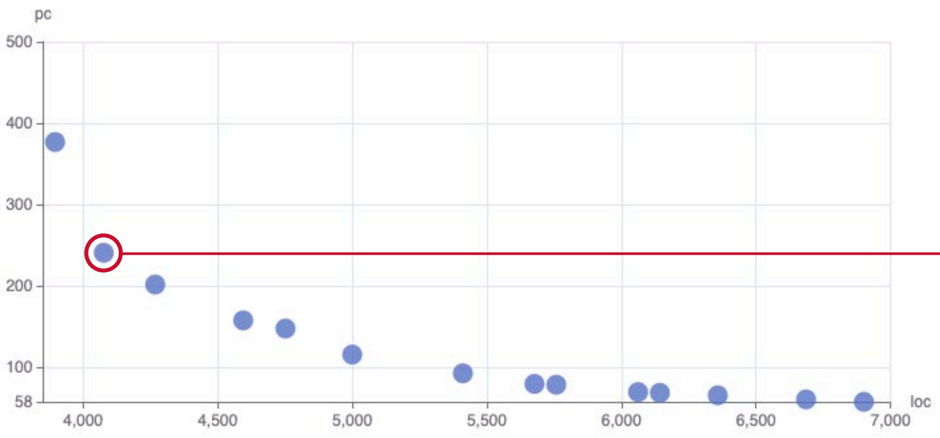
2. Formalize project-specific refactoring goals

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

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.

▾ Solution 12 -- aec = 4, dec = 0, loc = 4077, lscc = 0.56, pc = 241, rfLen = 18, rfOptions = 95, sparsity = 50, typesChanged = 8

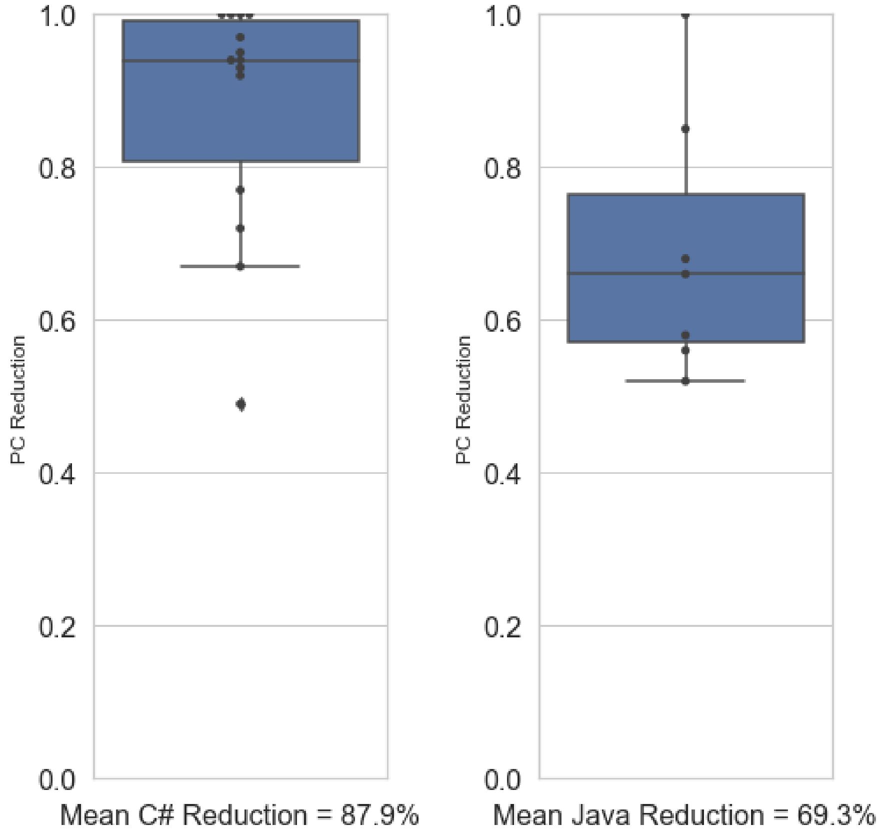
- Step 1: ExtractStaticClass (Duplicati.Library.Utility.Utility, {ForceStreamRead(Stream,byte[],int), ParseBoolOption(IDictionary<string,string>,string), ClientFilenameStringComparison, ParseBool(string,bool), ReadFileWithDefaultEncoding(string), EPOCH}) -> new_class_name_1
- Step 2: ExtractStaticClass (Duplicati.Library.Utility.Timeparser, {ParseTimeSpan(string), ParseTimeInterval(string,DateTime)}) -> new_class_name_2
- Step 3: ExtractStaticClass (Duplicati.Library.AutoUpdater.UpdaterManager, {RunFromMostRecent(System.Reflection.MethodInfo,string[],AutoUpdateStrategy), InstalledBaseDir, INSTALLED_BASE_DIR}) -> new_class_name_3
- Step 4: ExtractStaticClass (Duplicati.Library.Common.Platform, {IsClientWindows, IsClientPosix}) -> new_class_name_4
- Step 5: ExtractStaticClass (Duplicati.Library.Utility.Utility, {IsFSCaseSensitive, CachedIsFSCaseSensitive}) -> new_class_name_5
- Step 6: MoveInterface (Duplicati.Server.Serialization.Interface.ISetting)
- Step 7: MoveInterface (Duplicati.Server.Serialization.Interface.IBackup)
- Step 8: MoveClass (Duplicati.Server.Strings.Program)
- Step 9: ExtractStaticClass (Duplicati.Library.Localization.Short.LC, {L(string,object), L(string), L(string,object,object)}) -> new_class_name_6
- Step 10: MoveInterface (Duplicati.Server.Serialization.Interface.ISchedule)
- Step 11: MoveClass (Duplicati.Server.Database.TempFile)
- Step 12: MoveClass (Duplicati.Server.Database.Notification)
- Step 13: MoveInstanceMethod (Duplicati.Server.WebServer.RESTMethods.RequestInfo.ReportClientError(string,System.Net.HttpStatusCode), Duplicati.Server.WebServer.RESTMethods.Backups)
- Step 14: MoveInstanceMethod (Duplicati.Server.EventPollNotify.SignalNewEvent(), Duplicati.Server.Database.Connection)
- Step 15: MoveClass (Duplicati.Server.Strings.Server)
- Step 16: MoveClass (Duplicati.Library.AutoUpdater.UpdateInfo)
- Step 17: MoveEnum (Duplicati.Library.UsageReporter.ReportType)
- Step 18: MoveInterface (Duplicati.Server.Serialization.Interface.IFilter)

Current Capabilities

Carnegie Mellon University Software Engineering Institute

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

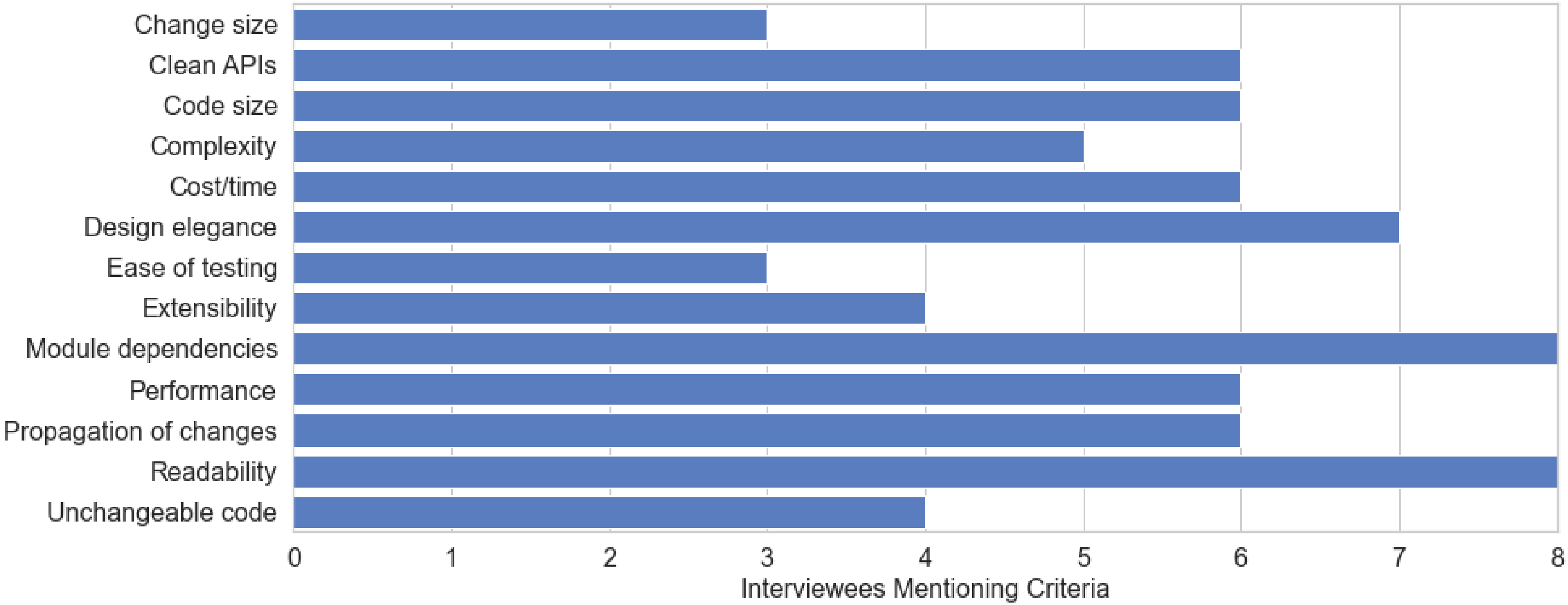


Refactoring Criteria

Carnegie Mellon University Software Engineering Institute

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

Carnegie Mellon University Software Engineering Institute

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

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0878