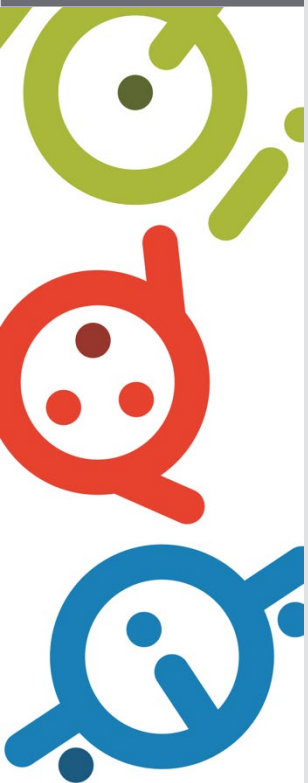
Carnegie Mellon University
Software Engineering Institute



Research Review 2021

Untangling the Knot: Automating Software Isolation

November 8, 2021

James Ivers

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Carnegie Mellon University Software Engineering Institute

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.

DM21-0908

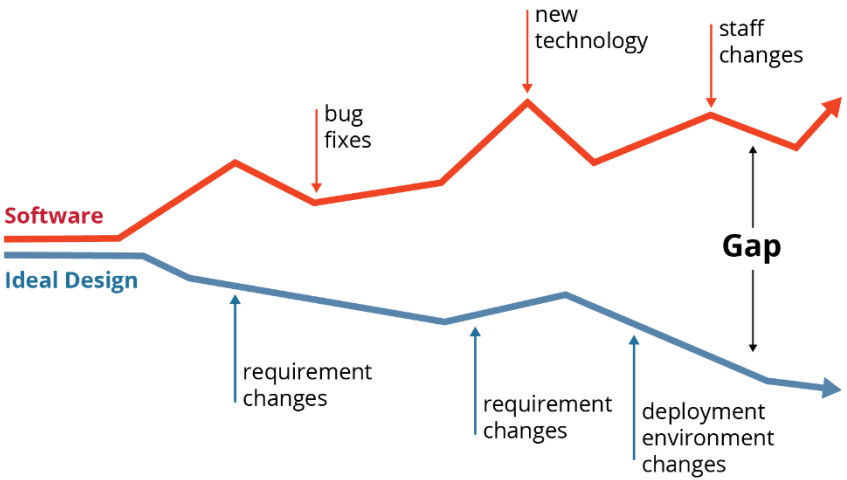
Software Is an Essential Building Material

Our ability to work with software significantly influences project cost, schedule, time to field, and other concerns.

The ability to efficiently build, change, and evolve software depends on its architecture and how that architecture is realized in code.

Architectures that are well aligned with needs allow faster changes with greater confidence.

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
- …

To adapt to such changes, we need to periodically improve software structure (architecture) to match today's needs.

Software Structure Becomes a Barrier to Software Evolution

Many evolution projects start with a common problem – isolating software:

- Reusing capability in a different system or rehosting on a different platform
- Factoring out common capability as a shared asset
- Decomposing a monolith into more modular code
- Migrating capabilities to a cloud or microservice architecture

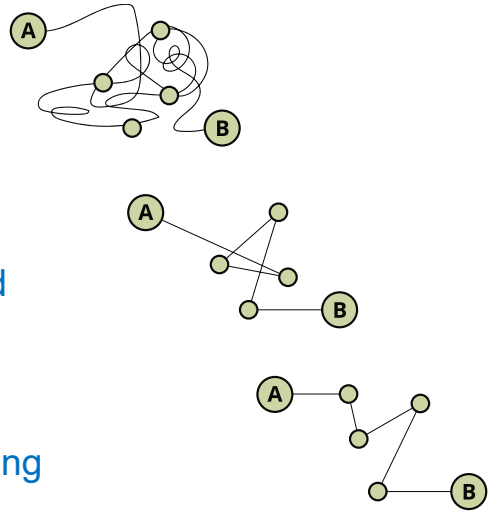
Refactoring Promises to Help

Refactoring is a known technique for improving the structure of software, but it is typically a labor-intensive process in which developers must

- figure out where to make changes
- figure out which refactoring(s) to use
- implement refactorings by rewriting code

Few tools recommend how to refactor code

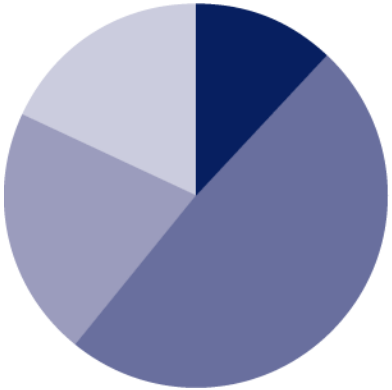
Many modern IDEs support code refactoring



Our Focus: Large-Scale Refactoring

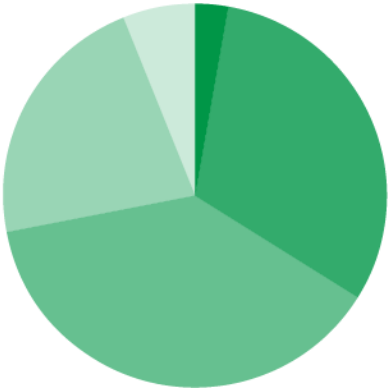
We surveyed practitioners to understand how large-scale refactoring is performed today.

Large-scale refactoring involves pervasive changes across a code base or extensive changes to a substantial element of the system (e.g., greater than 10K LOC).



How common is large-scale refactoring?

- **12%** I've participated in 5 or more
- **49%** I've participated in 2–4
- **21%** I've participated in 1
- **18%** I've never participated



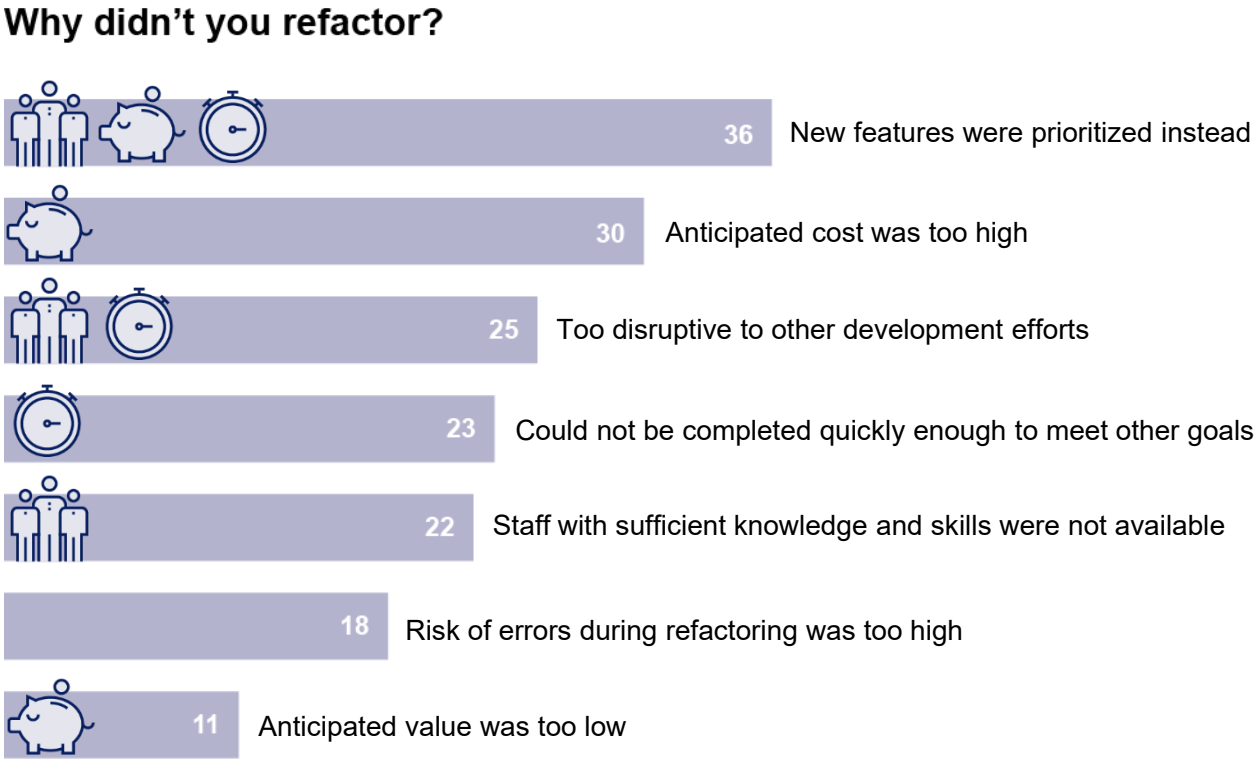
What are the sizes of these systems?

- **3%** 10M+ lines of code
- **31%** 1.1M–10M LOC
- **38%** 100K–1M LOC
- **22%** 10–100K LOC
- **6%** Less than 10K LOC

Organizations Often Defer Large-Scale Refactoring

70% of the respondents wanted to perform large-scale refactoring, but **did not do so.**

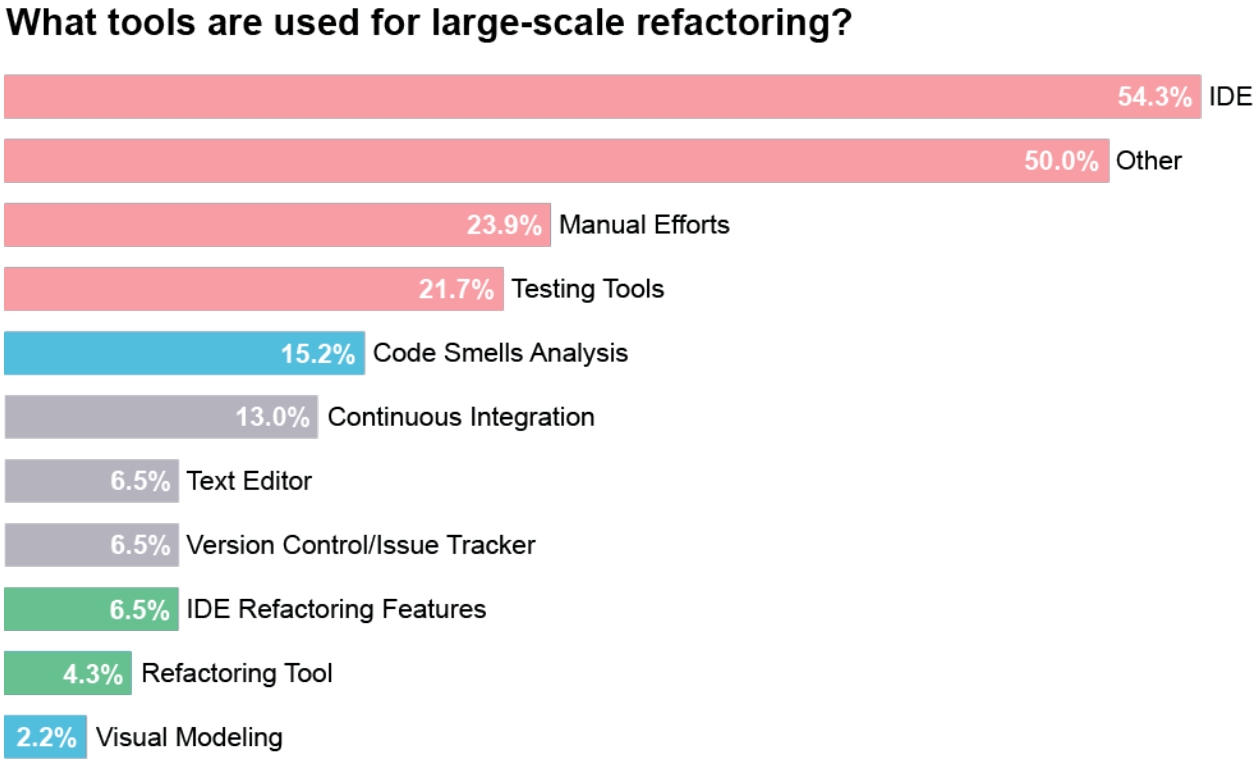
These reasons match what we have heard from many different organizations over many years.



Do Today's Tools Support Large-Scale Refactoring?

Our survey results show that

- developers rely heavily on their typical development tools, custom scripts, and manual efforts
- few tools cited support deciding where and how to refactor code used
- specialized refactoring tools are not commonly used



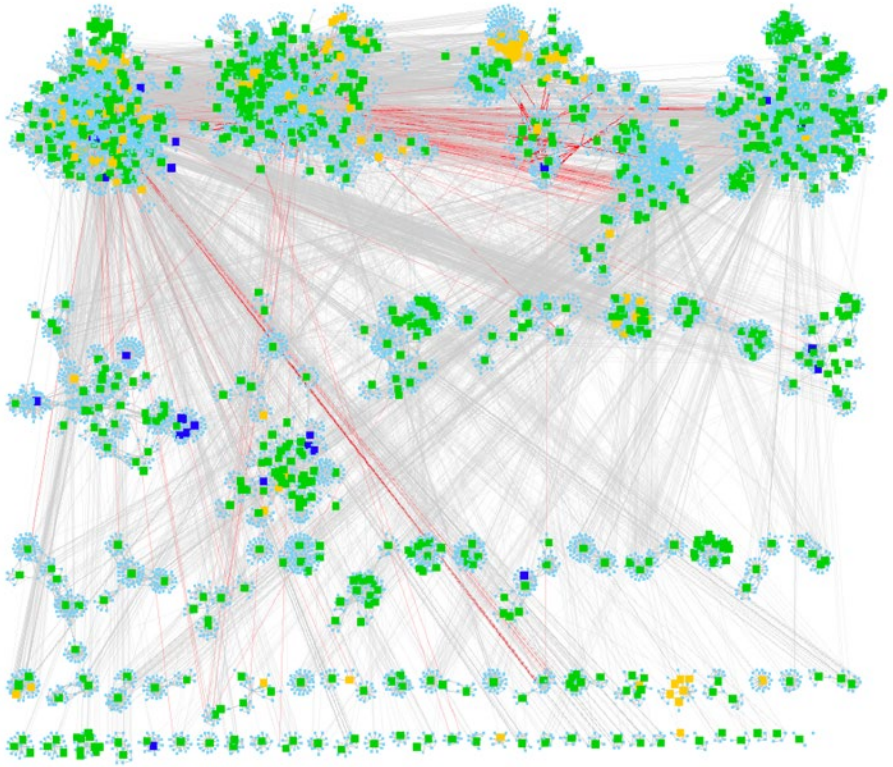
Our Solution: An Automated Refactoring Assistant

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

- Solves project-specific problems
- Uses a semi-automated approach
- Allows refactoring to be completed in less than 1/3 of the time required by manual approaches

J. Ivers, I. Ozkaya, R. L. Nord, C. Seifried. **Next Generation Automated Software Evolution: Refactoring at Scale**. *28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20)*. 2020.

Key Concept – Problematic Couplings



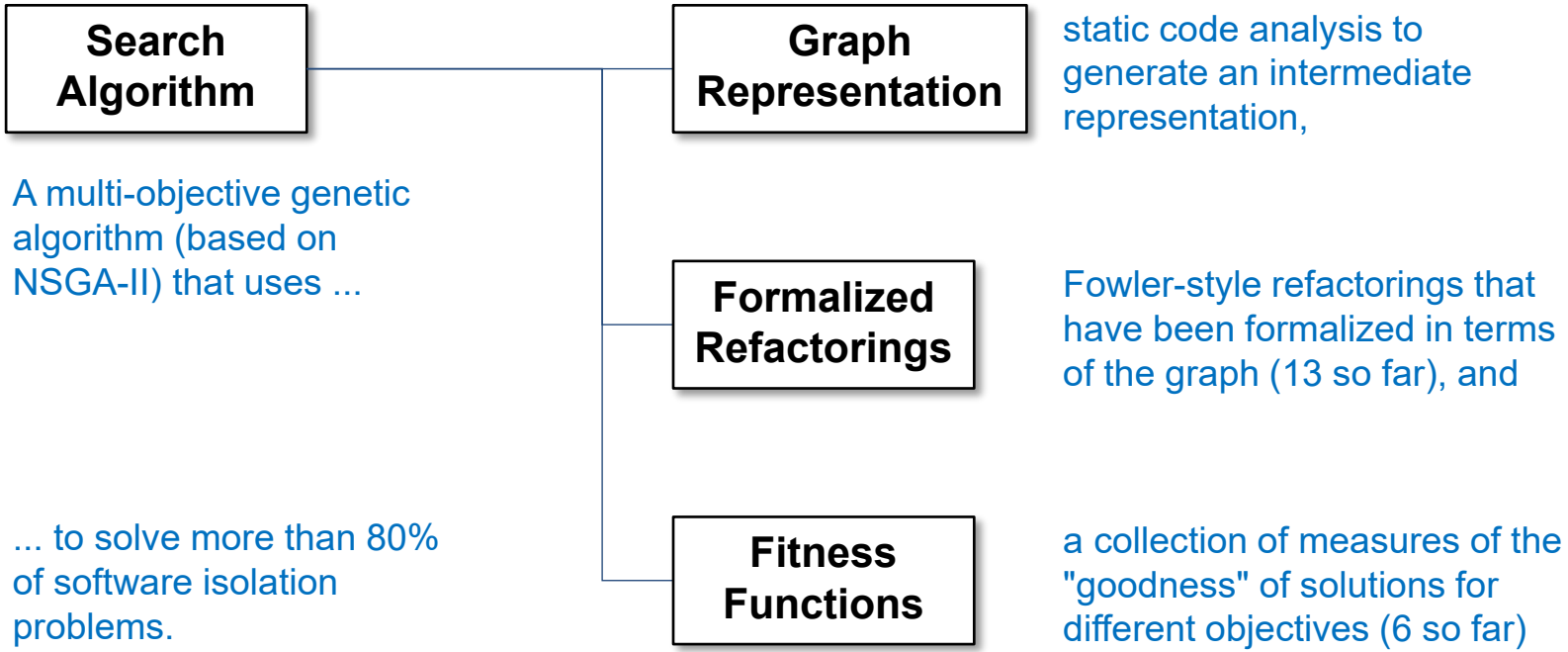
Only certain software dependencies interfere with any particular goal.

For example, if we want to reuse a feature:

- The core problem is dependencies (red lines) from software being reused to software that isn't
- All other dependencies are irrelevant to the goal, allowing us to focus our analysis and search for solutions

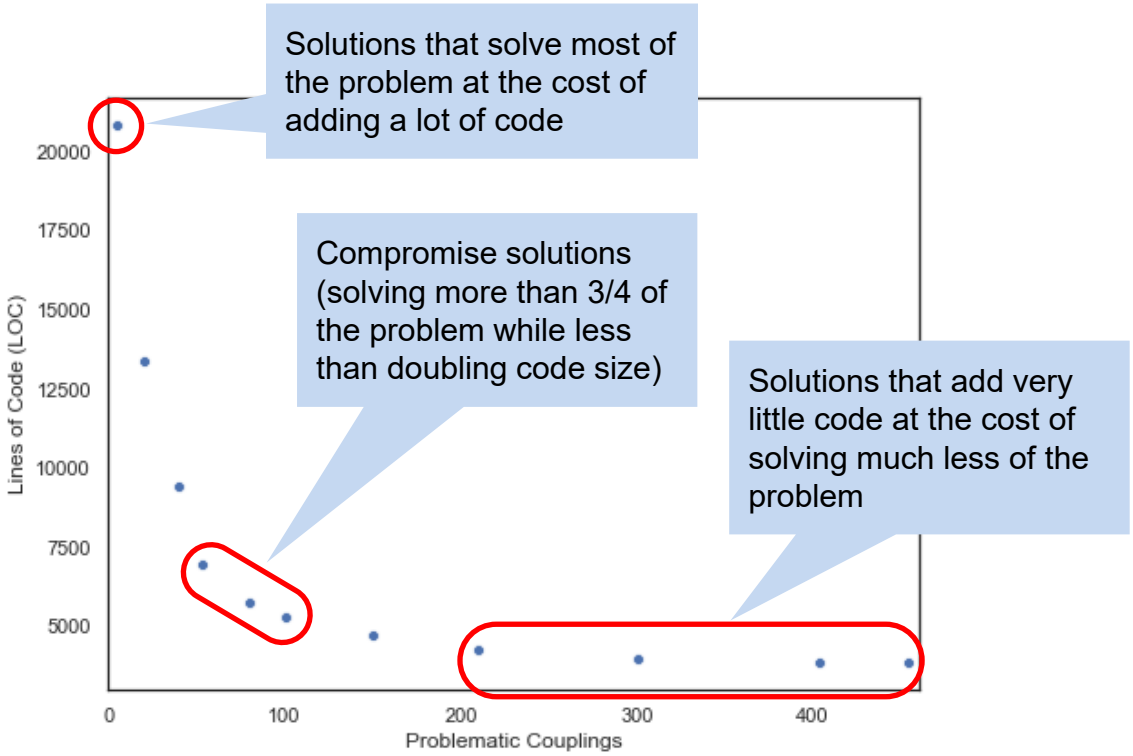
This insight enables us to apply **search-based software engineering** techniques and treat this as an **optimization problem**.

SEI's Automated Refactoring Assistant Prototype



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

Pareto-Optimal Solutions

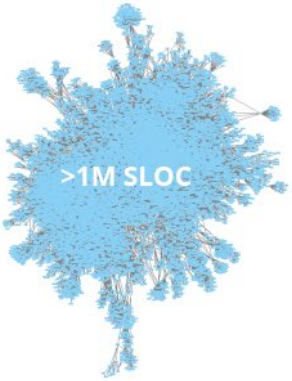


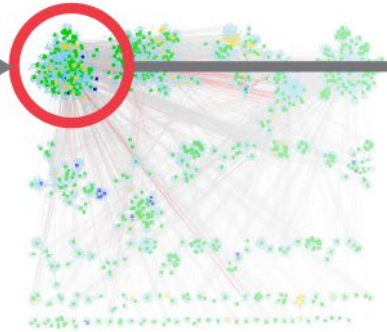
When optimizing for multiple objectives, there is no single best answer; instead we generate options that represent trade-offs among competing objectives.

This allows developers to choose the trade-offs that best match their needs.

Generating Refactoring Recommendations

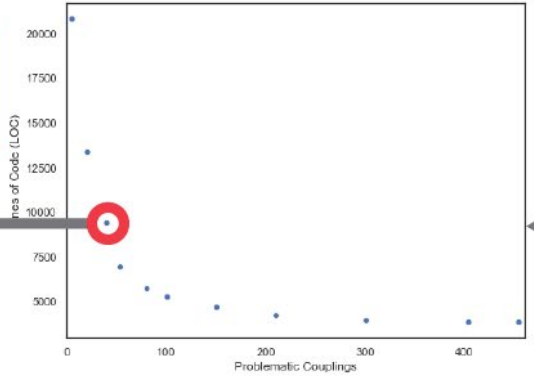
Carnegie Mellon University
Software Engineering Institute





Select Objectives
- minimize problematic couplings
- minimize code changes
- maximize code quality
- ...

Our prototype uses a multi-objective genetic algorithm to generate a set of Pareto-optimal solutions (recommendations)



Select and implement a solution that suits your context.

Refactoring Recommendations - 1

Step 1: MoveClass (Duplicati.Server.Database.Notification)
Step 2: MoveInterface (Duplicati.Server.Serialization.Interface.IBackup)
Step 3: MoveClass (Duplicati.Server.Database.Backup)
Step 4: MoveClass (Duplicati.Server.WebServer.RESTMethods.RequestInfo)
Step 5: MoveInterface (Duplicati.Server.Serialization.Interface.ISetting)
Step 6: ExtractStaticClass (Duplicati.Library.AutoUpdater.UpdaterManager, {RunFromMostRecent(MethodInfo,System.String,Duplicati.Library.AutoUpdater.AutoUpdateStrategy), InstalledBaseDir, INSTALLED_BASE_DIR}) -> new_class_name_1
> Supply a more meaningful name for the new Class (new_class_name_1).
Step 7: MoveInstanceMethod (Duplicati.Server.EventPollNotify.SignalNewEvent(), Duplicati.Server.Database.Connection)
> Convert the instance method to a static method by adding a new parameter with a type of the original declaring class. Also, update all references to this within the method to use the new parameter.
> Convert the member Duplicati.Server.EventPollNotify.m_eventNo to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_lock to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_waitQueue to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
Step 8: MoveInterface (Duplicati.Server.Serialization.Interface.ISchedule)

The refactoring assistant generates **step-by-step instructions** that

- are independently reviewable
- can be selectively applied

Refactoring Recommendations - 2

Step 1: **MoveClass** (Duplicati.Server.Database.Notification)
Step 2: **MoveInterface** (Duplicati.Server.Serialization.Interface.IBackup)
Step 3: **MoveClass** (Duplicati.Server.Database.Backup)
Step 4: **MoveClass** (Duplicati.Server.WebServer.RESTMethods.RequestInfo)
Step 5: **MoveInterface** (Duplicati.Server.Serialization.Interface.ISetting)
Step 6: **ExtractStaticClass** (Duplicati.Library.AutoUpdater.UpdaterManager, {RunFromMostRecent(MethodInfo,System.String,Duplicati.Library.AutoUpdater.AutoUpdateStrategy), InstalledBaseDir, INSTALLED_BASE_DIR}) -> new_class_name_1
> Supply a more meaningful name for the new Class (new_class_name_1).
Step 7: **MoveInstanceMethod** (Duplicati.Server.EventPollNotify.SignalNewEvent(), Duplicati.Server.Database.Connection)
> Convert the instance method to a static method by adding a new parameter with a type of the original declaring class. Also, update all references to this within the method to use the new parameter.
> Convert the member Duplicati.Server.EventPollNotify.m_eventNo to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_lock to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_waitQueue to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
Step 8: **MoveInterface** (Duplicati.Server.Serialization.Interface.ISchedule)

Solves the problem of recommending **which refactorings to apply**.

- Uses a vocabulary familiar to developers
- References refactorings that many modern IDEs implement

Refactoring Recommendations - 3

Step 1: MoveClass (**Duplicati.Server.Database.Notification**)
Step 2: MoveInterface (**Duplicati.Server.Serialization.Interface.IBackup**)
Step 3: MoveClass (**Duplicati.Server.Database.Backup**)
Step 4: MoveClass (**Duplicati.Server.WebServer.RESTMethods.RequestInfo**)
Step 5: MoveInterface (**Duplicati.Server.Serialization.Interface.ISetting**)
Step 6: ExtractStaticClass (**Duplicati.Library.AutoUpdater.UpdaterManager, {RunFromMostRecent(MethodInfo,System.String,Duplicati.Library.AutoUpdater.AutoUpdateStrategy), InstalledBaseDir, INSTALLED_BASE_DIR}**) -> new_class_name_1
> Supply a more meaningful name for the new Class (new_class_name_1).
Step 7: MoveInstanceMethod (**Duplicati.Server.EventPollNotify.SignalNewEvent(), Duplicati.Server.Database.Connection**)
> Convert the instance method to a static method by adding a new parameter with a type of the original declaring class. Also, update all references to this within the method to use the new parameter.
> Convert the member Duplicati.Server.EventPollNotify.m_eventNo to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_lock to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_waitQueue to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
Step 8: MoveInterface (**Duplicati.Server.Serialization.Interface.ISchedule**)

Solves the problem of recommending **where to apply the refactorings**.

- Provides clear parameters identifying where each refactoring should be applied

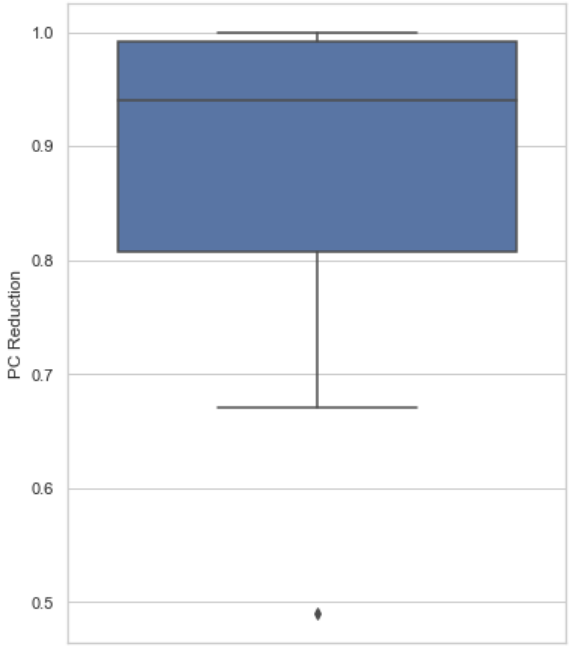
Refactoring Recommendations - 4

Step 1: MoveClass (Duplicati.Server.Database.Notification)
Step 2: MoveInterface (Duplicati.Server.Serialization.Interface.IBackup)
Step 3: MoveClass (Duplicati.Server.Database.Backup)
Step 4: MoveClass (Duplicati.Server.WebServer.RESTMethods.RequestInfo)
Step 5: MoveInterface (Duplicati.Server.Serialization.Interface.ISetting)
Step 6: ExtractStaticClass (Duplicati.Library.AutoUpdater.UpdaterManager, {RunFromMostRecent(MethodInfo,System.String,Duplicati.Library.AutoUpdater.AutoUpdateStrategy), InstalledBaseDir, INSTALLED_BASE_DIR}) -> new_class_name_1
> Supply a more meaningful name for the new Class (new_class_name_1).
Step 7: MoveInstanceMethod (Duplicati.Server.EventPollNotify.SignalNewEvent(), Duplicati.Server.Database.Connection)
> Convert the instance method to a static method by adding a new parameter with a type of the original declaring class. Also, update all references to this within the method to use the new parameter.
> Convert the member Duplicati.Server.EventPollNotify.m_eventNo to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_lock to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
> Convert the member Duplicati.Server.EventPollNotify.m_waitQueue to public to allow Duplicati.Server.EventPollNotify.SignalNewEvent to continue to access it.
Step 8: MoveInterface (Duplicati.Server.Serialization.Interface.ISchedule)

Provides context-specific instructions on **secondary changes** that enable the refactoring.

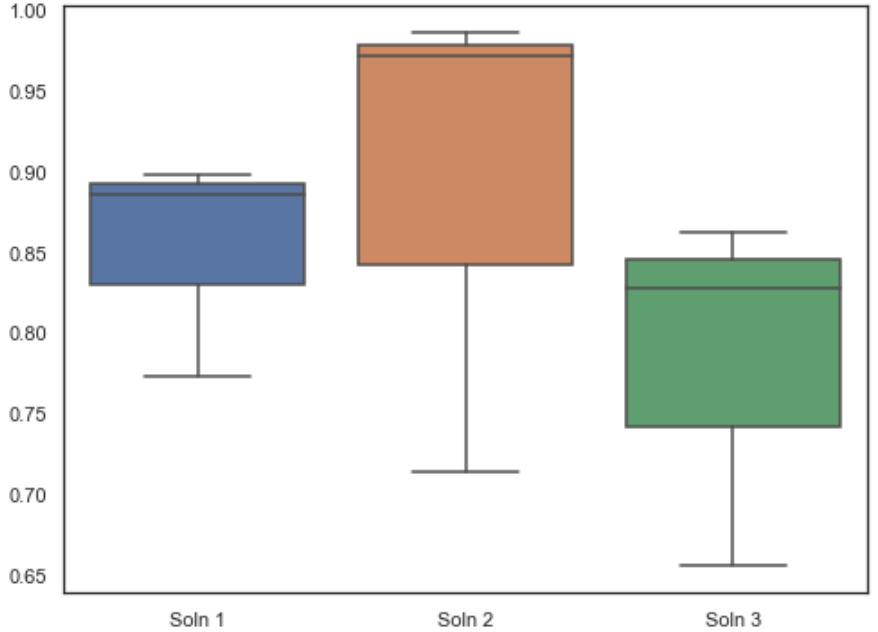
Results from 14 Open Source Scenarios

Solution Completeness



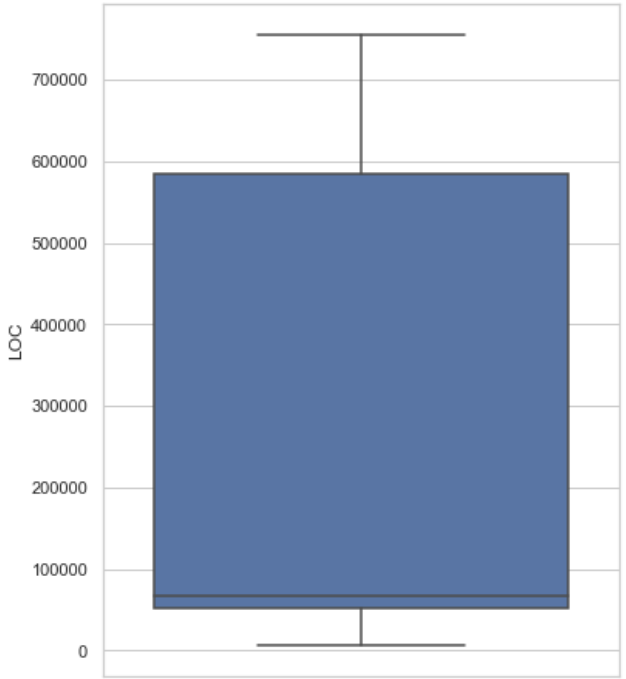
Mean PC Reduction = 87.9%
Mean Original PC Counts = 1,419.5

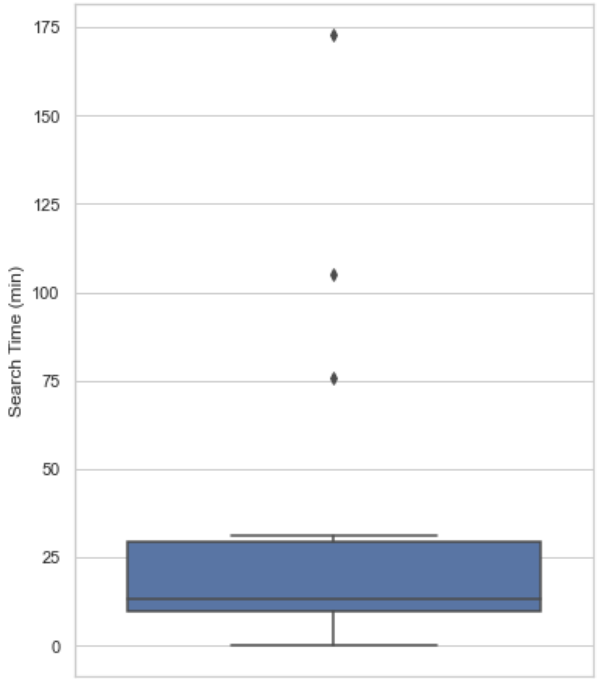
Solution Quality



Mean Acceptable Refactorings = 84.6%

Scalability of Our Solution





Scales to at least 1.2M SLOC of C# code

Search time is measurable in minutes to hours on a typical development laptop

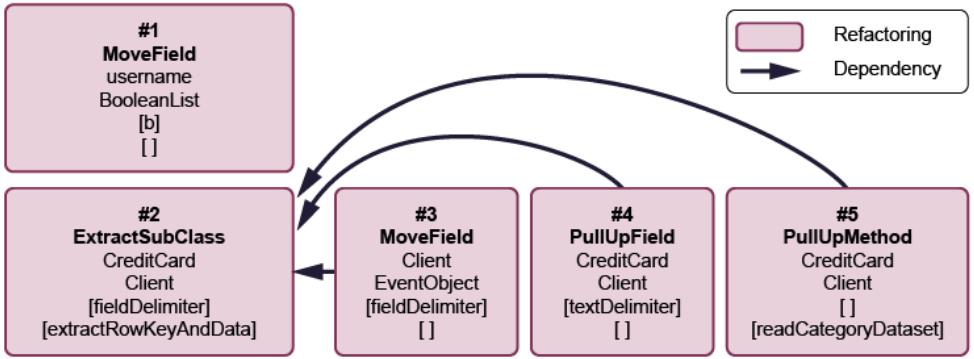
Coming Soon

Extend analysis to **refactor Java code** (ETA – early 2022)

Incorporate **constraint mechanisms** to generate solutions that accommodate common development constraints

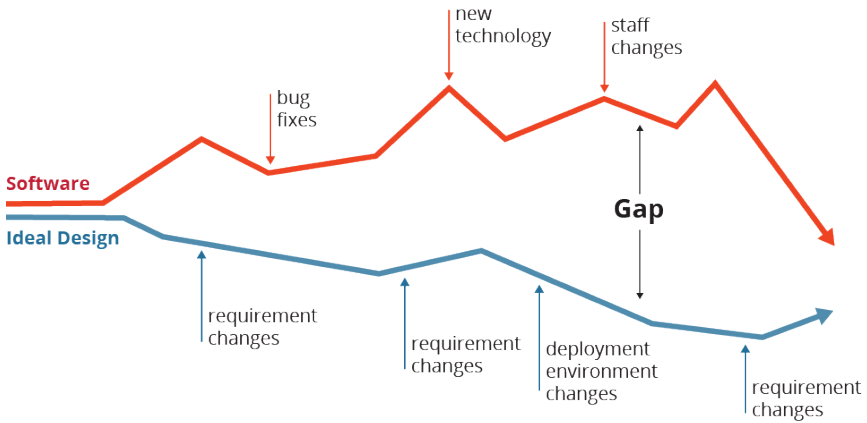
Build on our **refactoring dependency theory** to

- speed algorithm convergence
- help users understand and explore recommended solutions



C. Abid, J. Ivers, T. Ferreira, M. Kessentini, F. Kahla, I. Ozkaya. **Intelligent Change Operators for Multi-Objective Refactoring.** *Intl. Conference on Automated Software Engineering* (ASE). 2021.

Next-Generation Automation for Software Evolution



We are applying AI for Software Engineering to bend the cost curve for software evolution

- significantly reduce the time, cost, and disruption involved in refactoring software
- help organizations evolve software proactively and as frequently as needed rather than reactively or as a last resort

J. Ivers, I. Ozkaya, R. L. Nord. **Can AI Close the Design-Code Abstraction Gap?** *Software Engineering Intelligence Workshop 2019*, co-located with Intl. Conference on Automated Software Engineering.

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

The Knot Team

SEI Team

James Ivers

Chris Seifried

Ipek Ozkaya

Robert Nord

- Mario Benítez
- Vaughn Coates
- Andrew Kotov
- Reed Little
- Craig Mazzotta
- Scott Pavetti
- Scott Sinclair
- Jake Tannenbaum

External Collaborators

Research Collaborators

- Thiago Ferreira (University of Michigan, Assistant Professor)
- Clem Izurieta (Montana State University, Associate Professor)
- Marouane Kessentini (University of Michigan, Associate Professor)
- Chris Timperley (Carnegie Mellon University, Systems Scientist)

Students

- Chaima Abid (University of Michigan)
- Gavin Austin (Montana State University)
- Jared Frank (University of Pittsburgh)
- Carly Jones (Carnegie Mellon University)
- Katie Li (Carnegie Mellon University)
- Red Rajput (Carnegie Mellon University)
- Amy Tang (Carnegie Mellon University)
- Jeff Yackley (University of Michigan)