Untangling the Knot: Recommending Refactorings

James Ivers
May 7, 2020
Topics

Project Overview
Near-term Potential
Long-term Potential
Wrap Up
CMU SEI is a DoD Federally Funded Research and Development Center

Our mission: Engineering and securing software

Established in 1984 at Carnegie Mellon University

~635 employees

Offices in Pittsburgh and DC, with locations near customer facilities in MA, MD, TX, and CA

~$145M in annual funding (~$20M USD(R&E) 6.2 and 6.3 Line funding)
Establishing a Discipline of Software Architecture

Range of methods and practices applicable at different points in the development lifecycle.

- Domains of expertise include IT, C2, tactical, and health informatics
- Technology expertise includes IoT, big data, digital twin, cloud, and machine learning

10+ courses, available in a mix of public, on-site, and eLearning options.

3 professional certificates.

A collection of books for wide dissemination.
Project Overview
Software Structure Enables Our Ability to Innovate

Quickly delivering new capabilities and taking advantage of new technology depend on an ability to evolve software efficiently. The structure of legacy software, however, often fails to support this goal.

A recent anecdote from a DoD contractor: The estimate for isolating a mission capability from the underlying hardware platform was 14,000 staff hours (development only).

This is representative of a class of changes that involve feature isolation – isolating a specific software capability from its context.

Other examples include
- migrating a capability to the cloud
- harvesting a component for reuse
- replacing a proprietary component

Our project aims to allow feature isolation to be done in one-third of the time.
Software Complexity Is a Driver of the Effort Required

Even modest systems are hard to comprehend, and harder to modify.

- A modest application with only 68K lines of code (LOC) contains more than 10K nodes and 50K relations.
- Making a "simple" change, like isolating the code for deployment as a service, can require reasoning about hundreds of dependencies.
Our Goal: Create an Automated Refactoring Assistant

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

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

Our goal is to create an automated assistant for developers that recommends refactoring to isolate software, allowing features to be harvested or replaced in 1/3 of the time it takes to do so manually.

- Uses a semi-automated approach
- Addresses all three labor-intensive activities

In perspective, our work would reduce the cost in the earlier example from 14,000 staff hours to 4,500 staff hours—saving the cost of 9,500 hours of development.
Our Approach

We are adapting search-based optimization algorithms to recommend refactorings that isolate software to support harvesting or replacing features.
Feature Isolation Problems

Many software changes become much easier after isolating a feature from its context. We are focusing on two primary use cases: harvesting and replacing.

**Harvesting** software involves moving a feature from one context to another
- 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 services to a cloud or microservice architecture

**Replacing** software involve removing a feature in favor of another option
- Better options from another supplier
- Removing proprietary/licensed code
Near-term Potential
(i.e., we can do this now)
Problematic Couplings

Basis: Only certain software dependencies interfere with the goal.

If attempting to harvest a feature
- dependencies from software moving to a new context to software that isn't moving are the core problem (red lines)
- all other dependencies are irrelevant to the task, allowing us to focus our analysis and search for solutions
### Analysis of Problematic Couplings

<table>
<thead>
<tr>
<th>Project</th>
<th>Scenario</th>
<th>CALLS</th>
<th>IMPS</th>
<th>INHERITS</th>
<th>READS</th>
<th>USES_TYPE</th>
<th>WRITES</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MissionPlanner</td>
<td>Logger_A</td>
<td>515</td>
<td>0</td>
<td>1</td>
<td>982</td>
<td>255</td>
<td>403</td>
<td>2156</td>
</tr>
<tr>
<td>MissionPlanner</td>
<td>Logger_D</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>MissionPlanner</td>
<td>Radio_A</td>
<td>135</td>
<td>0</td>
<td>0</td>
<td>103</td>
<td>30</td>
<td>43</td>
<td>310</td>
</tr>
<tr>
<td>MissionPlanner</td>
<td>U1_A</td>
<td>2557</td>
<td>2</td>
<td>2</td>
<td>7269</td>
<td>2085</td>
<td>1493</td>
<td>13408</td>
</tr>
<tr>
<td>Duplicati</td>
<td>Logging_D</td>
<td>448</td>
<td>4</td>
<td>2</td>
<td>114</td>
<td>28</td>
<td>0</td>
<td>596</td>
</tr>
<tr>
<td>Duplicati</td>
<td>Server_A</td>
<td>105</td>
<td>3</td>
<td>0</td>
<td>235</td>
<td>56</td>
<td>52</td>
<td>451</td>
</tr>
<tr>
<td>Duplicati</td>
<td>Server_D</td>
<td>65</td>
<td>4</td>
<td>0</td>
<td>320</td>
<td>22</td>
<td>24</td>
<td>435</td>
</tr>
<tr>
<td>ConvNetSharp</td>
<td>GPU_D</td>
<td>529</td>
<td>0</td>
<td>1</td>
<td>495</td>
<td>384</td>
<td>7</td>
<td>1416</td>
</tr>
<tr>
<td>SharpCaster</td>
<td>Activity_D</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>3</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>eShopOnContaine</td>
<td>Eventbus_D</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>19</td>
<td>0</td>
<td>76</td>
</tr>
<tr>
<td>eShopOnContaine</td>
<td>Ordering_A</td>
<td>57</td>
<td>18</td>
<td>31</td>
<td>142</td>
<td>78</td>
<td>51</td>
<td>377</td>
</tr>
<tr>
<td>mRemoteNG</td>
<td>Putty_D</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>32</td>
<td>8</td>
<td>12</td>
<td>64</td>
</tr>
<tr>
<td>mRemoteNG</td>
<td>Rd_p_A</td>
<td>45</td>
<td>1</td>
<td>1</td>
<td>218</td>
<td>4</td>
<td>16</td>
<td>285</td>
</tr>
<tr>
<td>mRemoteNG</td>
<td>Rd_p_D</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>30</td>
<td>1</td>
<td>78</td>
</tr>
</tbody>
</table>

| Total | 4530 | 32  | 44  | 10003 | 3007 | 2103 |

- Code size spans 6K to 750K source LOC
- Scenarios illustrate a range of difficulty – 26 to 13K problematic couplings
- This information can be further analyzed to understand the complexity of a proposed change.
An Illustration of this Analysis

We've analyzed a number of open source projects and one commercial project.

The motivating scenario for this analysis is a desire to **replace the existing pub/sub mechanism** with a better option.

Data is from an open source project ([https://github.com/duplicati/duplicati](https://github.com/duplicati/duplicati)).
## Graph Data

**NODES [10207]**
- Namespaces = 79
- Classes = 866
- Interfaces = 132
- Structs = 24
- Fields = 2094
- Properties = 3160
- Methods = 3278
- Delegates = 13
- Events = 21
- Enums = 80
- Files = 460

**RELATIONSHIPS [49696]**
- Calls = 7986
- Reads = 15448
- Writes = 3984
- Inherits = 179
- Implements = 267
- Locations = 9670
- Declares = 9656
- Type Uses = 2506
Problematic Couplings for the Scenario

Initial problematic coupling count: **2,040**

- These are dependencies from code not being replaced to code that is being replaced
- Each occurrence is counted separately (e.g., two dependencies on the same method will be counted individually)

What does this mean?

We can slice the data a few different ways...
What Kinds of Dependencies did We Find?

Data looks pretty similar to open source examples that we're studying

- Most common relations involve reading/writing fields
- Method calls are common
- Inheritance and and interface implementation aren't common, but are more than in other cases
### Problematic Couplings - Target Type

<table>
<thead>
<tr>
<th>Project</th>
<th>Scenario</th>
<th>CLASS</th>
<th>ENUM</th>
<th>EVENT</th>
<th>FIELD</th>
<th>INTERFACE</th>
<th>METHOD</th>
<th>PROPERTY</th>
<th>STRUCT</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MissionPlanner</td>
<td>Logger_A</td>
<td>437</td>
<td>2</td>
<td>18</td>
<td>326</td>
<td>10</td>
<td>515</td>
<td>787</td>
<td>61</td>
<td>2156</td>
</tr>
<tr>
<td>MissionPlanner</td>
<td>Logger_D</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>MissionPlanner</td>
<td>Radio_A</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>17</td>
<td>134</td>
<td>102</td>
<td>0</td>
<td>310</td>
</tr>
<tr>
<td>MissionPlanner</td>
<td>UI_A</td>
<td>1700</td>
<td>579</td>
<td>82</td>
<td>2395</td>
<td>112</td>
<td>2557</td>
<td>4781</td>
<td>1202</td>
<td>13408</td>
</tr>
<tr>
<td>Duplicati</td>
<td>Logging_D</td>
<td>66</td>
<td>37</td>
<td>0</td>
<td>24</td>
<td>11</td>
<td>451</td>
<td>7</td>
<td>0</td>
<td>596</td>
</tr>
<tr>
<td>Duplicati</td>
<td>Server_A</td>
<td>74</td>
<td>3</td>
<td>24</td>
<td>15</td>
<td>43</td>
<td>105</td>
<td>187</td>
<td>0</td>
<td>451</td>
</tr>
<tr>
<td>Duplicati</td>
<td>Server_D</td>
<td>14</td>
<td>10</td>
<td>2</td>
<td>192</td>
<td>14</td>
<td>64</td>
<td>139</td>
<td>0</td>
<td>435</td>
</tr>
<tr>
<td>ConvNetSharp</td>
<td>GPU_D</td>
<td>546</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>529</td>
<td>337</td>
<td>0</td>
<td>1416</td>
</tr>
<tr>
<td>SharpCaster</td>
<td>Activity_D</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>eShopOnContainers</td>
<td>Eventbus_D</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>28</td>
<td>4</td>
<td>0</td>
<td>76</td>
</tr>
<tr>
<td>eShopOnContainers</td>
<td>Ordering_A</td>
<td>97</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>54</td>
<td>57</td>
<td>158</td>
<td>0</td>
<td>377</td>
</tr>
<tr>
<td>mRemoteNG</td>
<td>Putty_D</td>
<td>34</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>20</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>mRemoteNG</td>
<td>Rdp_A</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>1</td>
<td>46</td>
<td>211</td>
<td>0</td>
<td>285</td>
</tr>
<tr>
<td>mRemoteNG</td>
<td>Rdp_D</td>
<td>10</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>Testcase</td>
<td>PubSub_D</td>
<td>363</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>334</td>
<td>1335</td>
<td>0</td>
<td>2040</td>
</tr>
</tbody>
</table>

|               |          | 3431  | 691  | 140   | 3009  | 278    | 4866   | 8080    | 1263   |

By type, most dependencies are on Properties, Classes, and Methods.
What do the Dependencies Point To? - 2

<table>
<thead>
<tr>
<th>Target Type</th>
<th>PC Count</th>
<th># Unique Targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>363</td>
<td>15</td>
</tr>
<tr>
<td>Event</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Method</td>
<td>334</td>
<td>51</td>
</tr>
<tr>
<td>Property</td>
<td>1335</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td><strong>2040</strong></td>
<td><strong>144</strong></td>
</tr>
</tbody>
</table>

By name, dependencies concentrate on a smaller number of elements—144 unique names that collectively cover 2040 dependencies.
Where are All the Types Defined?

All 15 classes are defined in namespaces
• The classes are declared across 11 namespaces
• There are no dependencies on nested classes

All 129 elements (Events, Methods, and Properties) are all defined in classes
• Specifically, they are defined across 17 classes
• 9 of these classes are not included in the above list of 15 classes

So, basically there are dependencies on 24 classes (or members thereof)
• The initial scenario identified 53 classes, of which less than half are issues in a replacement scenario
Long-term Potential
(i.e., give us 6-12 months)
Our Approach

We are adapting search-based optimization algorithms to recommend refactorings that isolate software to support harvesting or replacing capabilities.
Local search based on problematic couplings

- Illustrative of what we're working towards
- Not yet what we'd consider a "good" solution, but encouraging
Current Work – Multi-objective Search

Multi-objective genetic algorithms like NSGA-II allow us to employ multiple fitness functions and generate Pareto-optimal solutions.

We are exploring fitness functions to find a combination that yields recommendations that developers will accept.

Candidate fitness functions include
- solving the core problem – minimizing problematic couplings
- reducing work – minimizing code changes and unrealized interfaces
- maintainable code – improving a range of code quality metrics
- understandable code – maximizing semantic coherence
Refactoring Recommendations

Our prototype generates recommendations as a sequence of refactorings:

- clear directions for a developer
- independently reviewable prior to changing code
- built on refactorings supported by development environments
- future potential to automate application of the refactorings to code

Best solution:
Fitness = 33

Step 2: MoveClass (Duplicati.Library.AutoUpdater.AutoUpdateSettings)
Step 3: MoveClass (Duplicati.Library.Utility.WorkerThread?)
Step 4: MoveInterface (Duplicati.Server.Serialization.Interface.ISchedule)
Step 5: MoveInterface (Duplicati.Server.Serialization.Interface.IBackup)
Step 6: MoveInterface (Duplicati.Server.Serialization.Interface.ISetting)
Step 7: MoveClass (Duplicati.Server.Strings.Program)
Step 8: MoveClass (Duplicati.Server.Database.Backup)
Step 9: MoveClass (Duplicati.Library.Localization.Short.IC)
Step 10: MoveClass (Duplicati.Server.Database.Notification)
Step 11: MoveClass (Duplicati.Server.WebServer.IndexHtmlHandler)
Step 13: MoveClass (Duplicati.Server.Database.TempFile)
Step 16: MoveInterface (Duplicati.Library.Interface.ICommandLineArgument)
Step 20: MoveClass (Duplicati.Library.Server.Lifecycle)
Step 21: MoveClass (Duplicati.Library.Interface.Strings.DataTypes)
Step 23: MoveInterface (Duplicati.Library.Serialization.Interface.IFilter)
Step 24: MoveInterface (Duplicati.Library.Localization.ILocalizationService)
Step 30: MoveClass (Duplicati.Library.Interface.UserInputException)
Step 32: MoveClass (Duplicati.Library.UpdatePollThread)
Step 33: MoveClass (Duplicati.Library.AutoUpdater.UpdateInfo)
Step 34: MoveClass (Duplicati.Server.Strings.Server)
Step 35: MoveClass (Duplicati.Library.Common.IO.Util)
Step 36: MoveInterface (Duplicati.Library.Utility.IFilter)
Step 38: MoveInterface (Duplicati.Library.Common.IO.IsSystemIO)
Wrap Up

Please let us know if you are interested. For example, if you
• would like to discuss potential use for estimation
• have ideas on important objectives to consider
• have access to data we could use to validate the research
• would like to discuss opportunities to pilot the refactoring recommendations

Our current prototype
• is open source, but relies on one commercial tool
• handles C# code, with future potential for Java
• has been tested on up to 1.2M SLOC code bases
For More Information

James Ivers
Initiative Lead
Architecture Analysis, Design, and Automation
jivers@sei.cmu.edu
412-268-7793