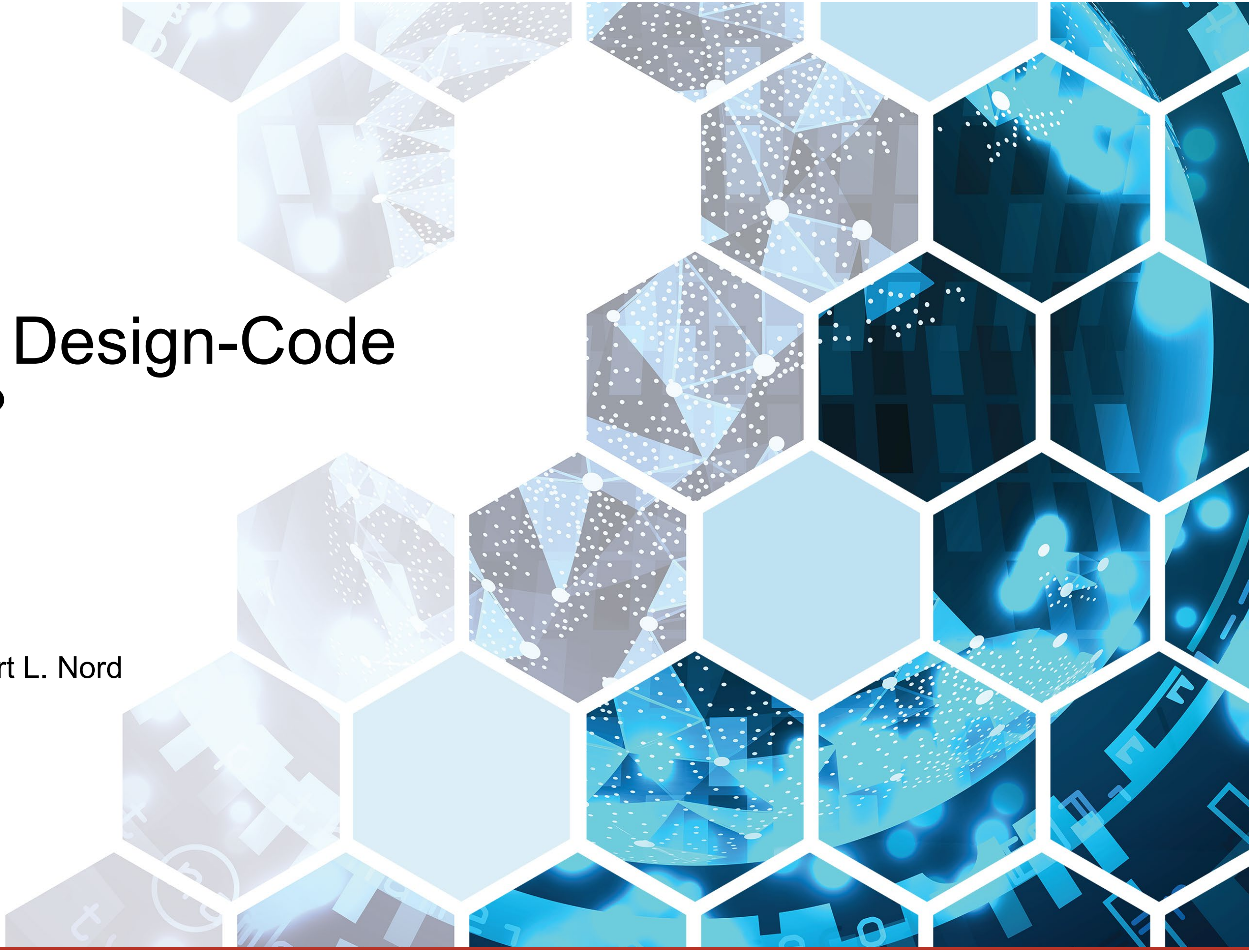
Can AI Close the Design-Code Abstraction Gap?

James Ivers, Ipek Ozkaya, and Robert L. Nord
{jivers, ozkaya, rn}@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



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.

DM19-1197

A Bit of Context

Software architecture is an important abstraction that helps organizations satisfy a wide range of business and mission goals.

- Our team has decades of experience creating and applying architecture approaches to a wide range of government and commercial systems
- We primarily deal with large-scale changes to existing systems (e.g., modernization)
- A common impediment is that architecture and design documentation is often missing or out of date

When architecture and design information differ from code, we generally

- Trust the code
- Lose the ability to apply architectural analyses (e.g., diagnosing root causes or the implications of a potential change)

Common Challenges

"I need to ..."

- Move this monolith to the cloud
- Decide whether to modernize a system or start over
- Grab this bit of functionality for use in a new product
- Replace that bit of functionality with a newer version from another vendor
- Determine whether what my contractor delivered will be maintainable

Recently, an organization wanted to isolate a mission capability from its underlying hardware platform in preparation for moving it to a new platform. Their contractor's response – an estimate of 14,000 staff hours (development only).

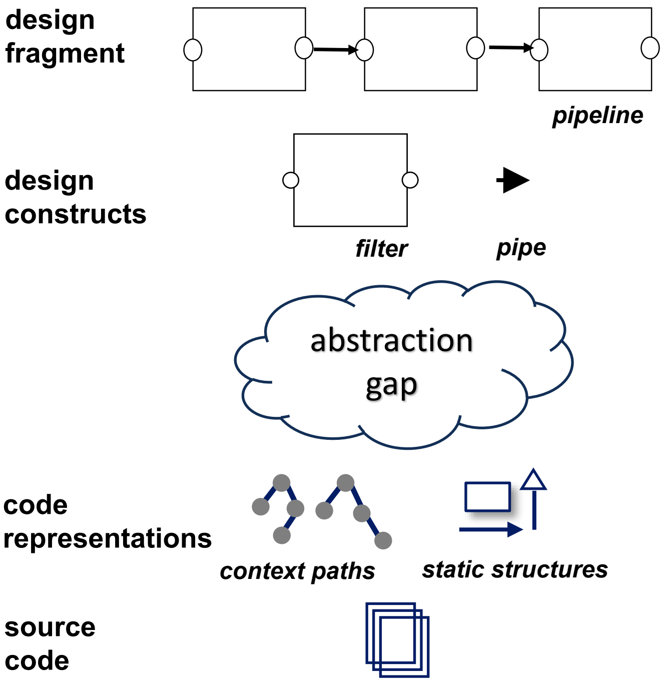
Is this reasonable?

How Can AI for Software Engineering Help?

We are motivated to help create a new generation of automation for architects that helps bridge the gap between architecture abstractions and code. We are encouraged by potential applications of AI to

- Detect design abstractions, allowing us to
    - Recover "as implemented" designs
    - Check that implementation conform to "as intended" designs
- Refactor code to improve its design

Detecting Design Abstractions



design fragment

pipeline

design constructs

filter

pipe

abstraction gap

code representations

context paths

static structures

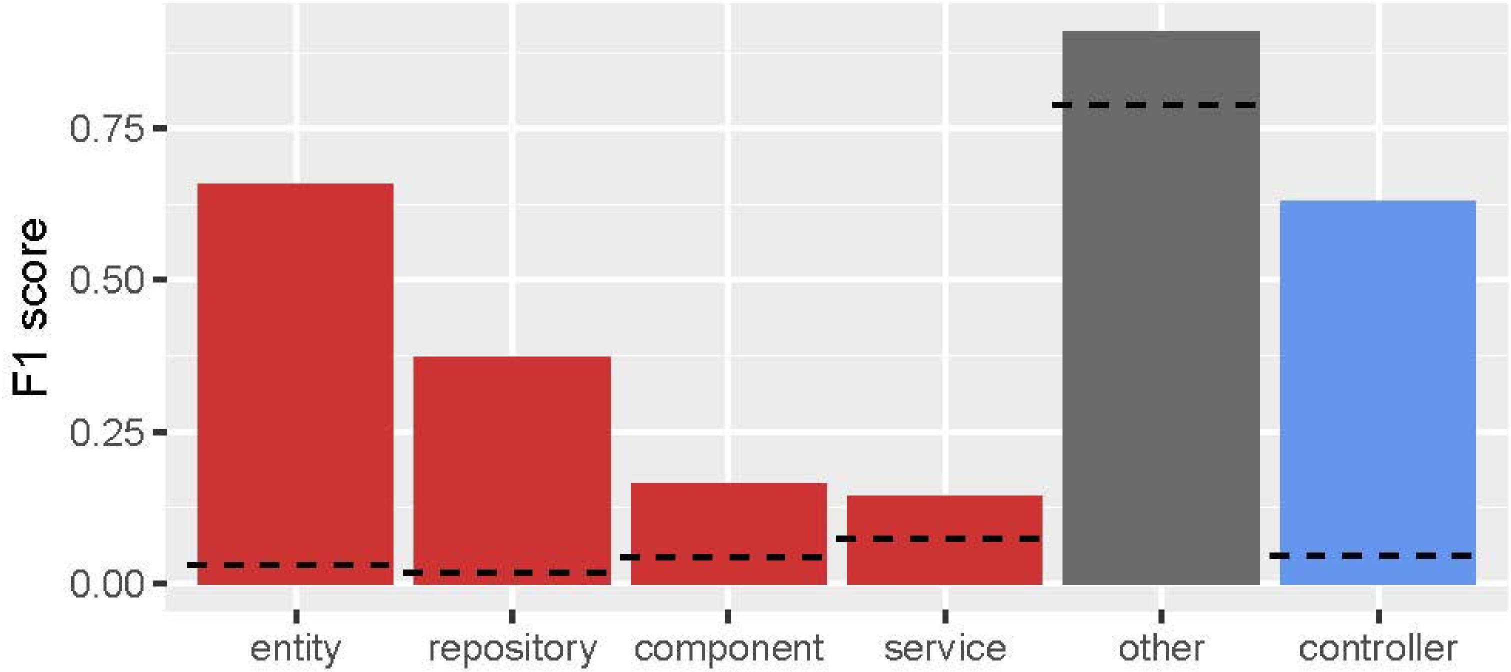
source code

Machine learning classification is a promising approach to recognizing the presence of design abstractions from source code

- Growing successes in the literature
- Imperfect matches are wanted
- Our feature engineering is based on combinations of structural dependencies and context paths

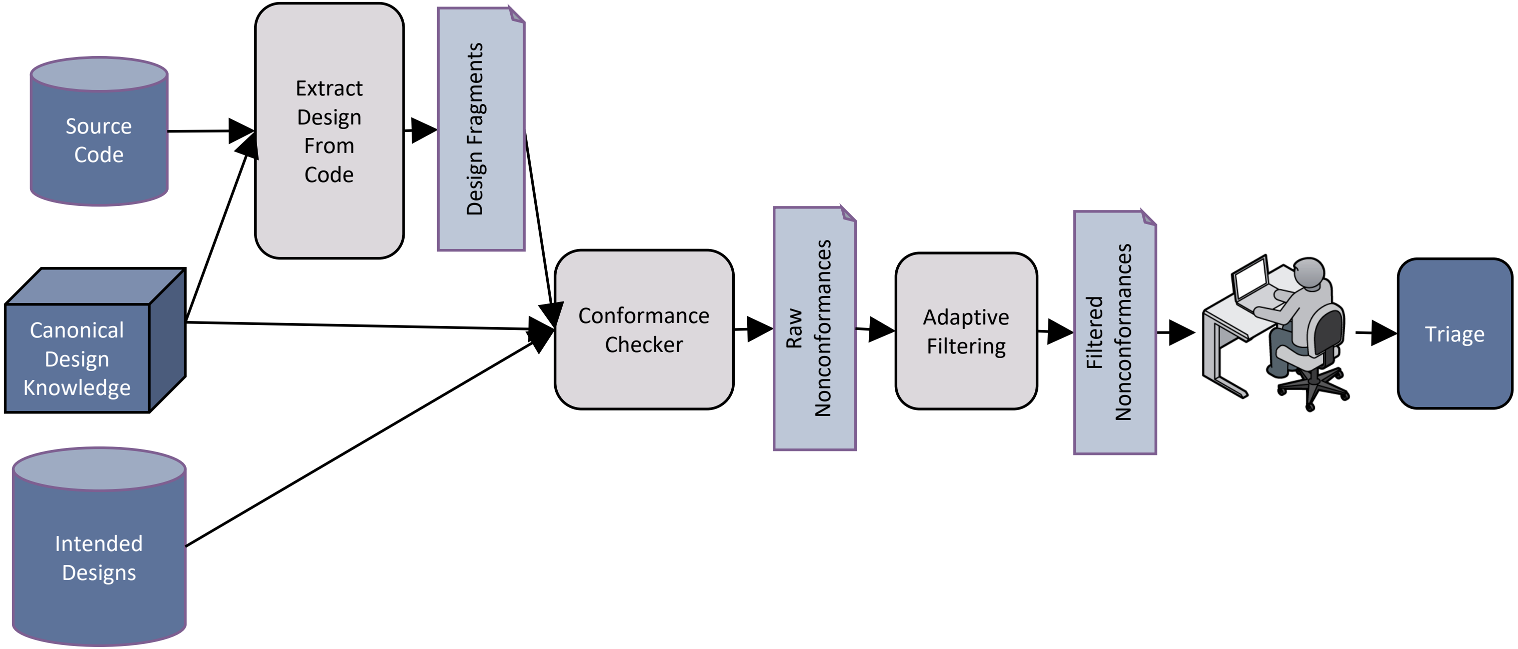
Allamanis, Barr, Devanbu, & Sutton. A Survey of Machine Learning for Big Code and Naturalness. ACM Comput. Surv. 51, 4 (2018).
Zanoni, Fontana, & Stella. On applying machine learning techniques for design pattern detection. J. Syst. Softw. 103, C (2015).
Alon, Zilberstein, Levy, & Yahav. code2vec: learning distributed representations of code. Proc. ACM Program. Lang. 3, POPL (2019).

Initial Progress



MVC Design Constructs – – random guessing

Intended Application - Automated Conformance Checker



Refactoring Code to Improve Its Design

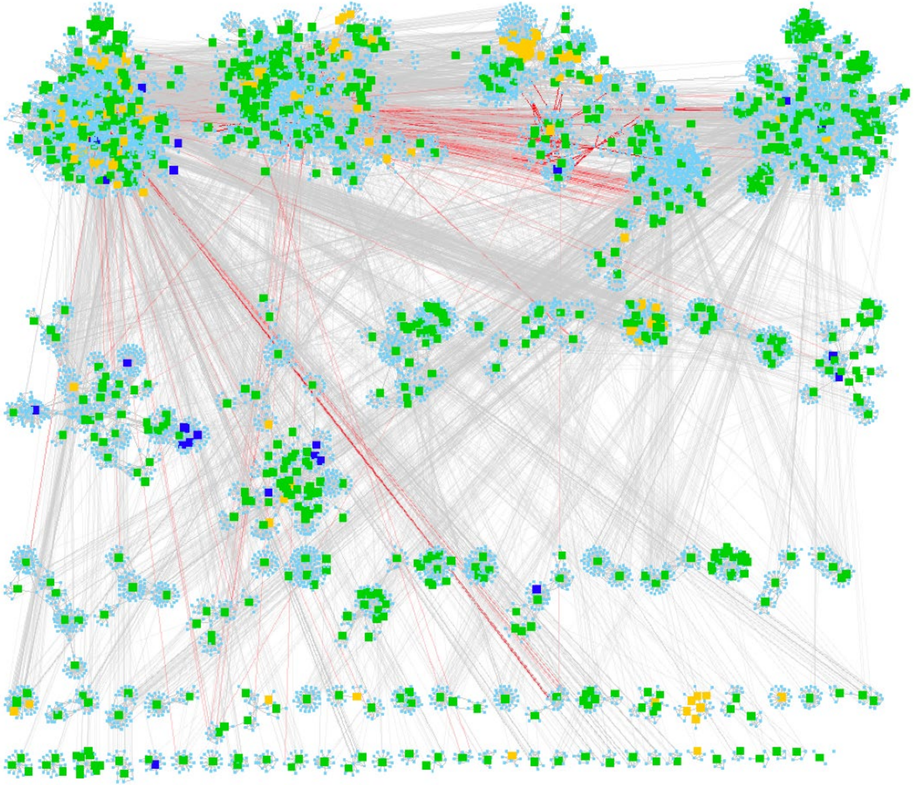
Search-based software engineering approaches have shown promise in improving code quality across a codebase.

Harman & Tratt. Pareto Optimal Search Based Refactoring at the Design Level. GECCO 2007.
Mkaouer, Kessentini, Bechikh, Cinnéide, & Deb. On the Use of Many Quality Attributes for Software Refactoring: A Many-Objective Search-Based Software Engineering Approach. *Empir. Softw. Eng.* (2015).
Ouni, Kessentini, Sahraoui, Inoue, & Deb. Multi-Criteria Code Refactoring Using Search-Based Software Engineering: An Industrial Case Study. ACM Trans. Softw. Eng. Methodol. (2016).

Our goal is to apply these approaches to recommend refactorings that achieve *project-specific goals (*specifically: isolating functionality for harvesting or replacement).

- Multi-objective algorithms like NSGA-II fit naturally with the need to accommodate design trade-offs
- Pareto-optimal solutions are acceptable in this domain
- Reasonable efficiency on large search spaces (code size and refactoring options)

Problem Framing

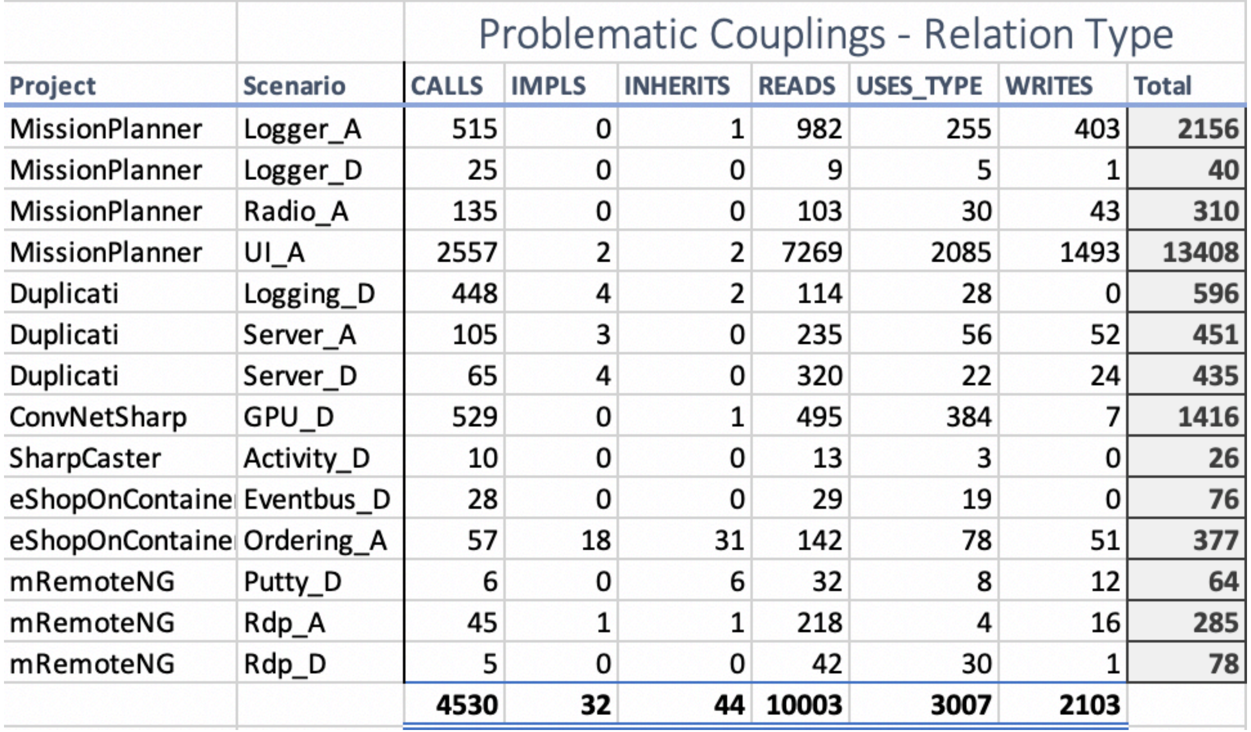


Basis: Only certain software dependencies interfere with our goals.

Approach: Focus search on solutions that reduce those dependencies.

- Counting those dependencies is an objective basis for fitness.
- Reducing scope of search (by 1 to 4 orders of magnitude) promotes scalability.

Problematic Couplings

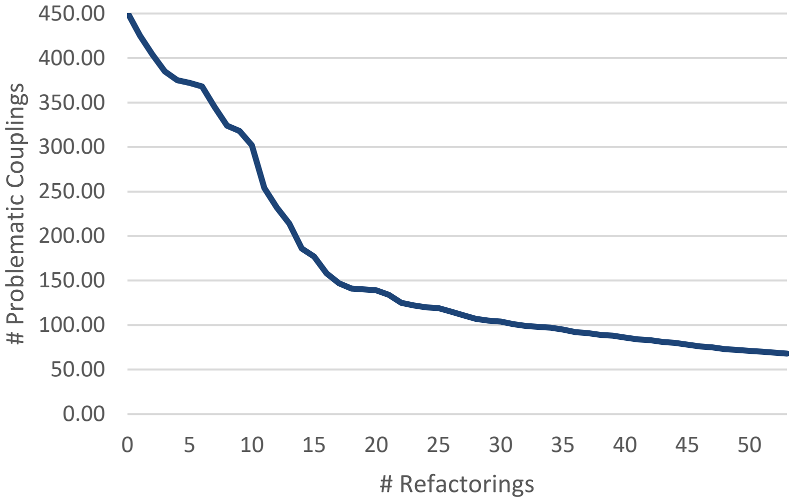


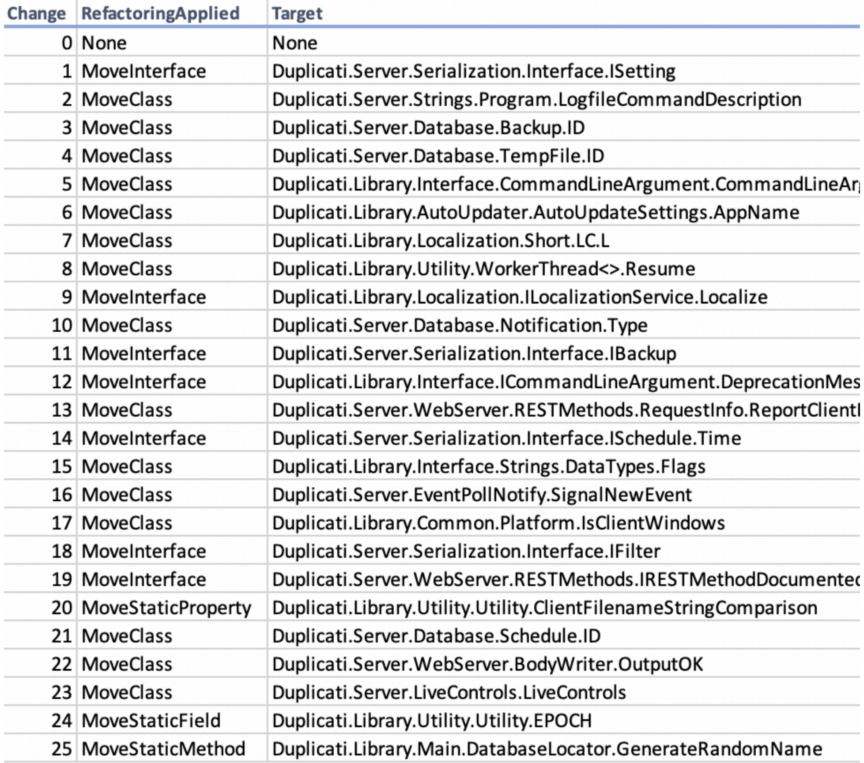
		Problematic Couplings - Relation Type						
Project	**Scenario**	**CALLS**	**IMPLS**	**INHERITS**	**READS**	**USES_TYPE**	**WRITES**	**Total**
MissionPlanner	Logger_A	515	0	1	982	255	403	**2156**
MissionPlanner	Logger_D	25	0	0	9	5	1	**40**
MissionPlanner	Radio_A	135	0	0	103	30	43	**310**
MissionPlanner	UI_A	2557	2	2	7269	2085	1493	**13408**
Duplicati	Logging_D	448	4	2	114	28	0	**596**
Duplicati	Server_A	105	3	0	235	56	52	**451**
Duplicati	Server_D	65	4	0	320	22	24	**435**
ConvNetSharp	GPU_D	529	0	1	495	384	7	**1416**
SharpCaster	Activity_D	10	0	0	13	3	0	**26**
eShopOnContaine	Eventbus_D	28	0	0	29	19	0	**76**
eShopOnContaine	Ordering_A	57	18	31	142	78	51	**377**
mRemoteNG	Putty_D	6	0	6	32	8	12	**64**
mRemoteNG	Rdp_A	45	1	1	218	4	16	**285**
mRemoteNG	Rdp_D	5	0	0	42	30	1	**78**
		4530	**32**	**44**	**10003**	**3007**	**2103**	

Problematic couplings are software dependencies that interfere with achieving a specific goal.

Source: All project data from github.com/open-source

Early Refactoring Recommendations



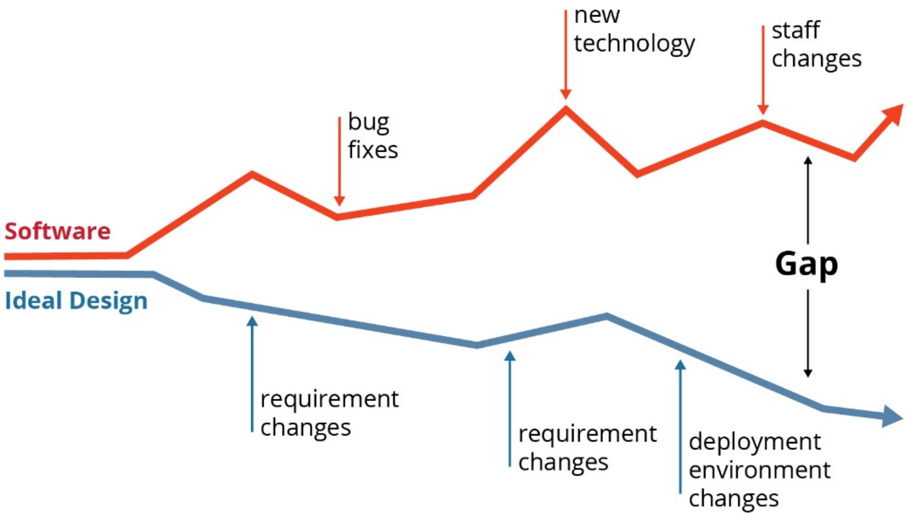


Change	RefactoringApplied	Target
0	None	None
1	MoveInterface	Duplicati.Server.Serialization.Interface.ISetting
2	MoveClass	Duplicati.Server.Strings.Program.LogfileCommandDescription
3	MoveClass	Duplicati.Server.Database.Backup.ID
4	MoveClass	Duplicati.Server.Database.TempFile.ID
5	MoveClass	Duplicati.Library.Interface.CommandLineArgument.CommandLineAr
6	MoveClass	Duplicati.Library.AutoUpdater.AutoUpdateSettings.AppName
7	MoveClass	Duplicati.Library.Localization.Short.LC.L
8	MoveClass	Duplicati.Library.Utility.WorkerThread<>.Resume
9	MoveInterface	Duplicati.Library.Localization.ILocalizationService.Localize
10	MoveClass	Duplicati.Server.Database.Notification.Type
11	MoveInterface	Duplicati.Server.Serialization.Interface.IBackup
12	MoveInterface	Duplicati.Library.Interface.ICommandLineArgument.DeprecationMes
13	MoveClass	Duplicati.Server.WebServer.RESTMethods.RequestInfo.ReportClient
14	MoveInterface	Duplicati.Server.Serialization.Interface.ISchedule.Time
15	MoveClass	Duplicati.Library.Interface.Strings.DataTypes.Flags
16	MoveClass	Duplicati.Server.EventPollNotify.SignalNewEvent
17	MoveClass	Duplicati.Library.Common.Platform.IsClientWindows
18	MoveInterface	Duplicati.Server.Serialization.Interface.IFilter
19	MoveInterface	Duplicati.Server.WebServer.RESTMethods.IRESTMethodDocumente
20	MoveStaticProperty	Duplicati.Library.Utility.Utility.ClientFilenameStringComparison
21	MoveClass	Duplicati.Server.Database.Schedule.ID
22	MoveClass	Duplicati.Server.WebServer.BodyWriter.OutputOK
23	MoveClass	Duplicati.Server.LiveControls.LiveControls
24	MoveStaticField	Duplicati.Library.Utility.Utility.EPOCH
25	MoveStaticMethod	Duplicati.Library.Main.DatabaseLocator.GenerateRandomName

Local search with a single fitness function

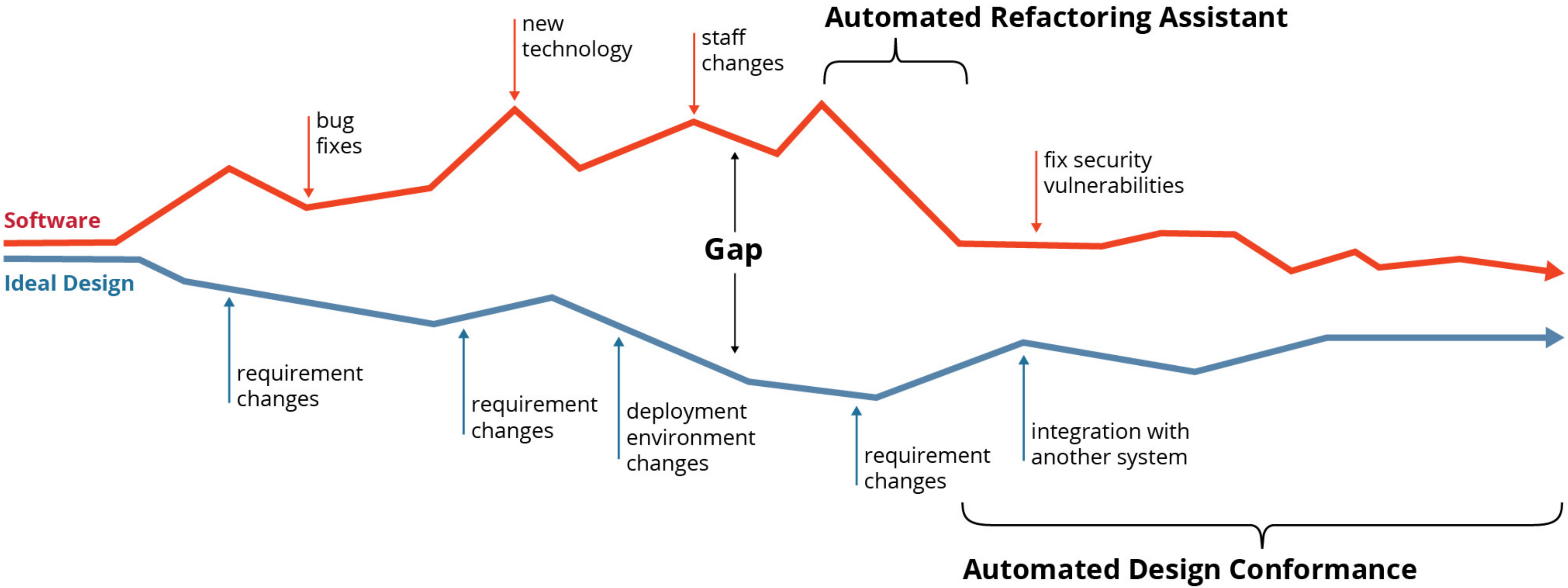
- Illustrative of what we're working towards
- Not yet what we'd consider a "good" solution, but encouraging

Over Time, Gaps Seem Inevitable



When software structure differs significantly from what is needed, the pace of change and innovation slows down.

Different AI Approaches Solve Complementary Problems



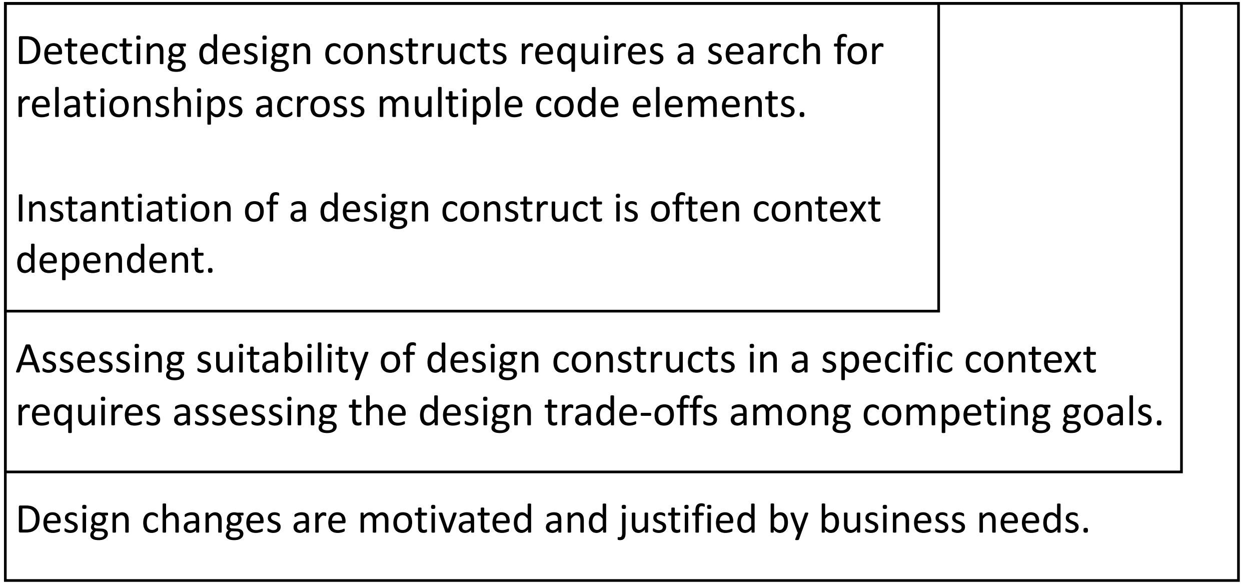
Automated Refactoring Assistant
new technology
staff changes
bug fixes
Software
Ideal Design
Gap
fix security vulnerabilities
requirement changes
requirement changes
deployment environment changes
requirement changes
integration with another system
Automated Design Conformance

Deeper Integration Seems Promising, As Well

Our vision is to combine these two ideas to

- Search for refactorings that correct detected non-conformances (new trigger for search with a narrower scope)
- Preserve existing design abstractions during refactoring (new search constraints based on abstractions)
- Search for opportunities to introduce abstractions (new fitness functions, likely requiring greater developer interaction)

Layers of Challenges



Core Technical Challenge

Detecting design constructs requires a search for relationships across multiple code elements.

Instantiation of a design construct is often context dependent.

System Context Influences Use

Assessing suitability of design constructs in a specific context requires assessing the design trade-offs among competing goals.

Business Context Influences Priorities

Design changes are motivated and justified by business needs.