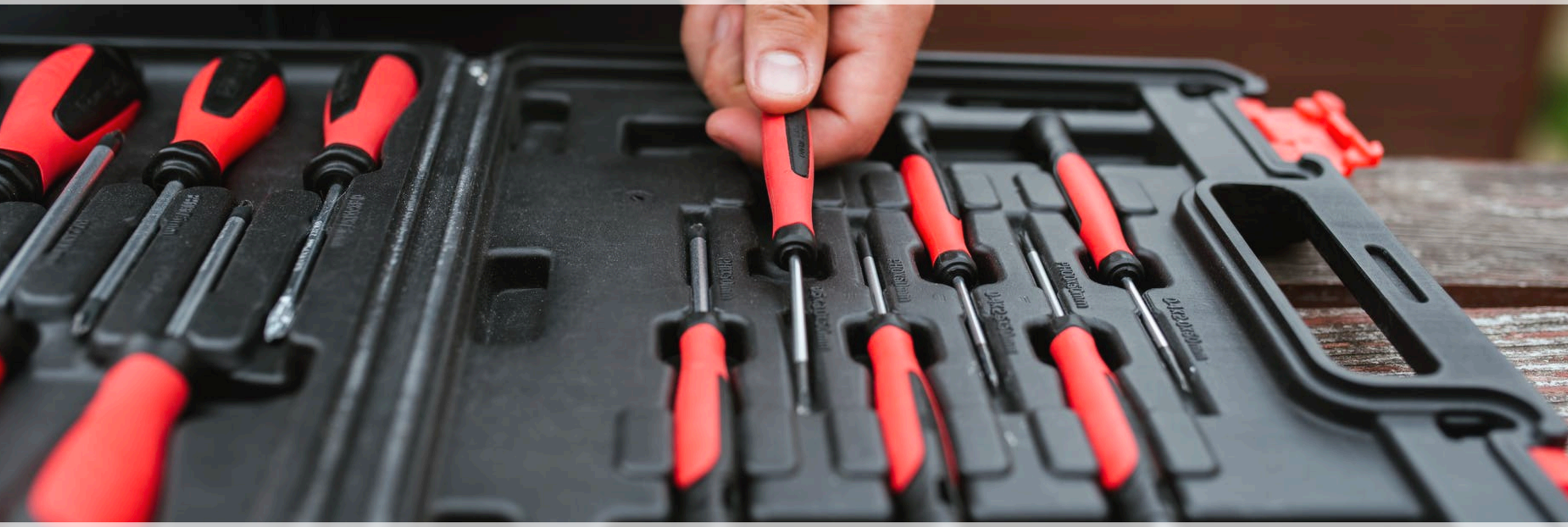
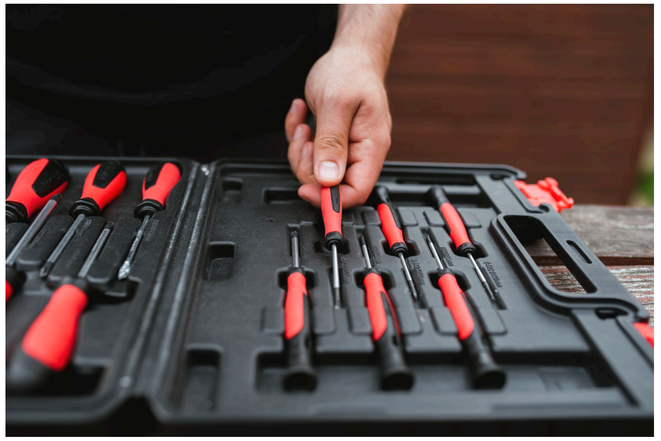
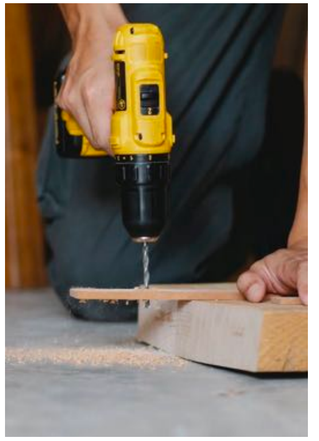
Improving the Usability of Refactoring Tools for Software Change Tasks

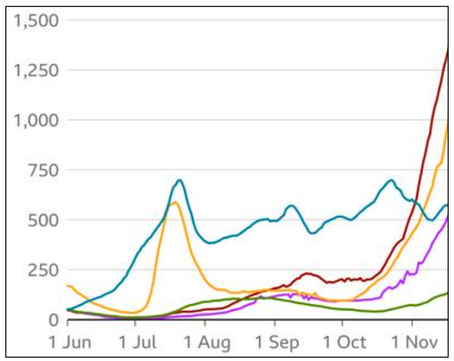


University of Bergen | Institute for Informatics | PhD Defense | Anna Maria Eilertsen | 22. December 2021 | Bergen / Zoom

Tool support

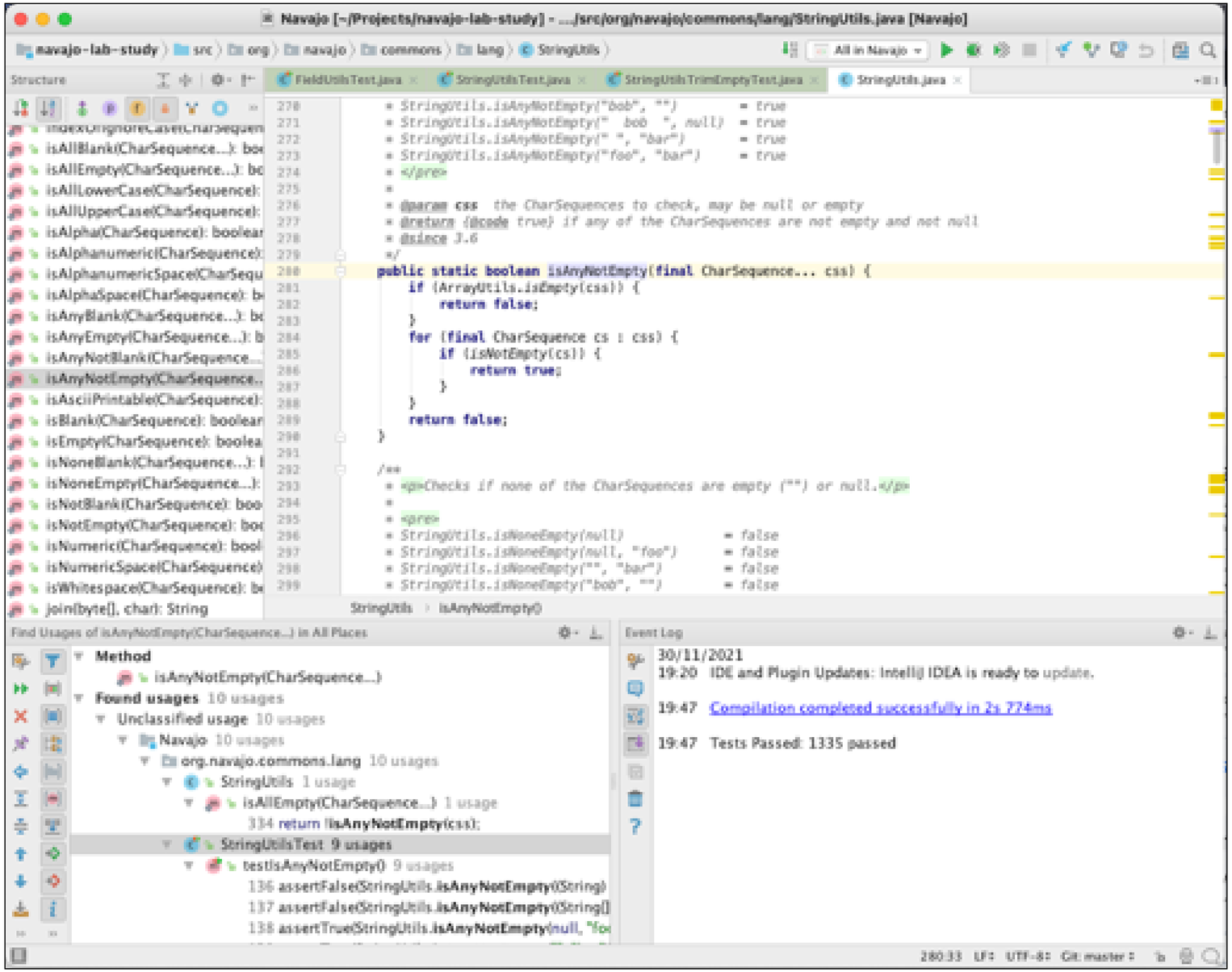




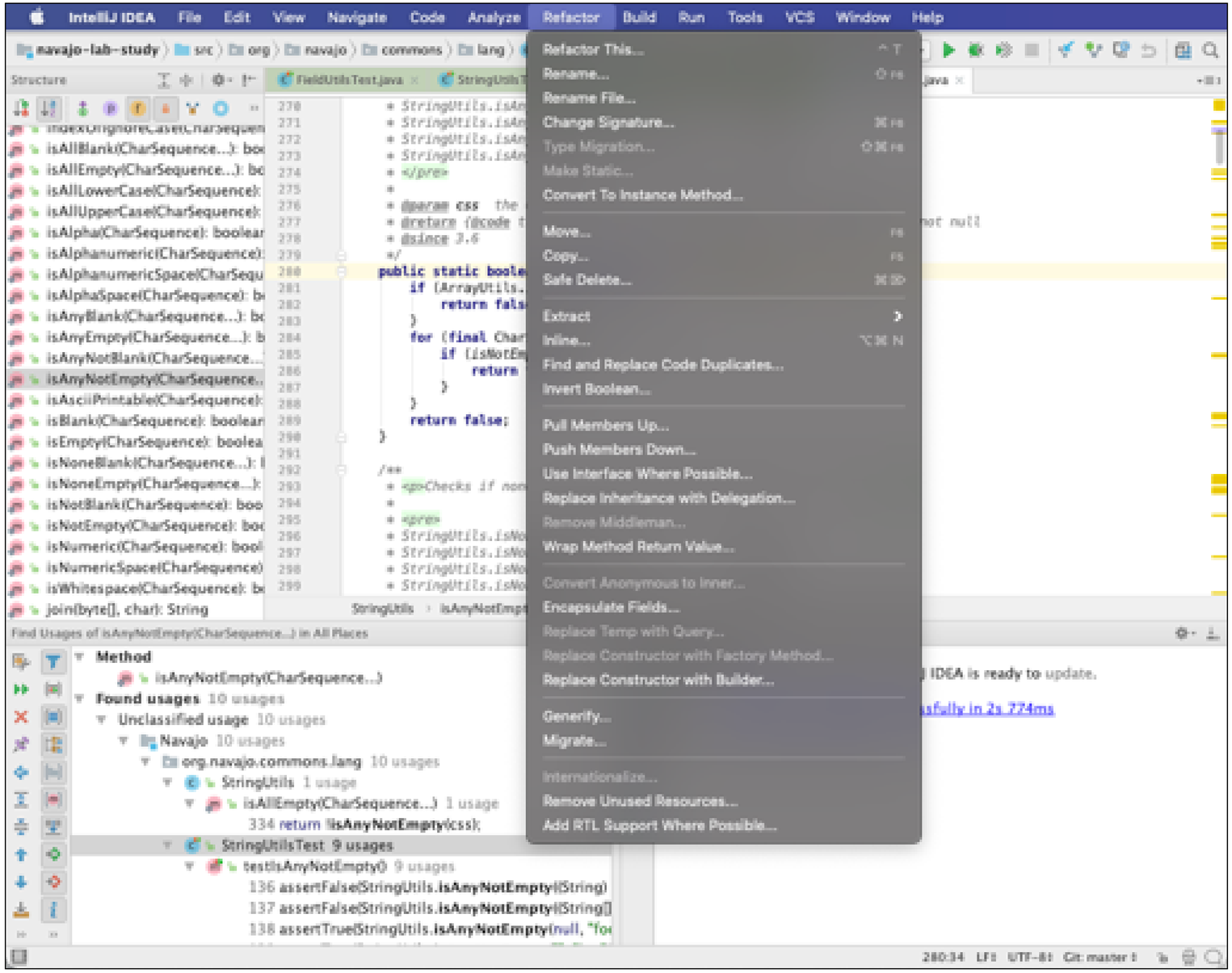

1,500
1,250
1,000
750
500
250
0
1 Jun
1 Jul
1 Aug
1 Sep
1 Oct
1 Nov


Google
www.google.com
Google Search
I'm Feeling Lucky
Advertising Programs
Business Solutions
About Google

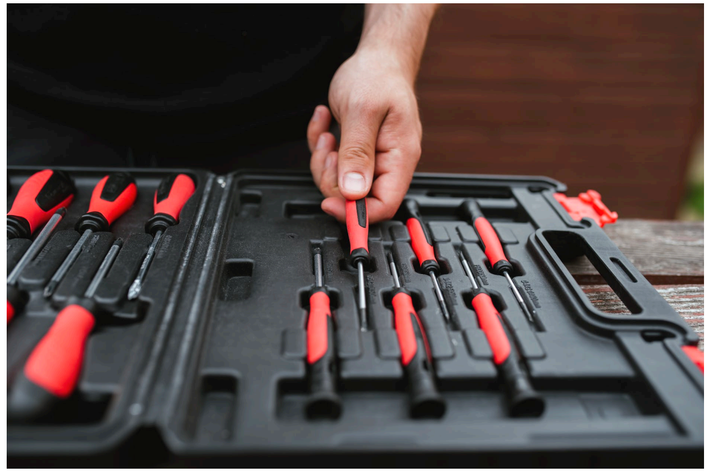
Tool support

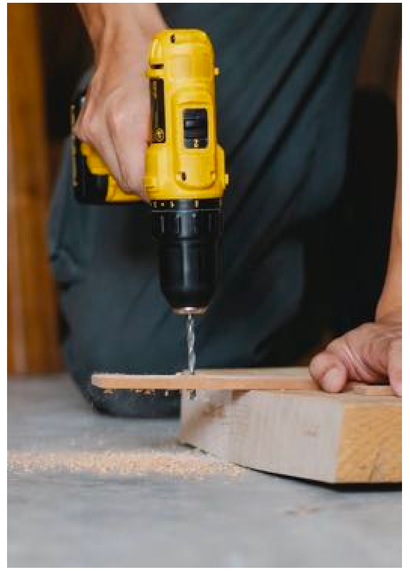


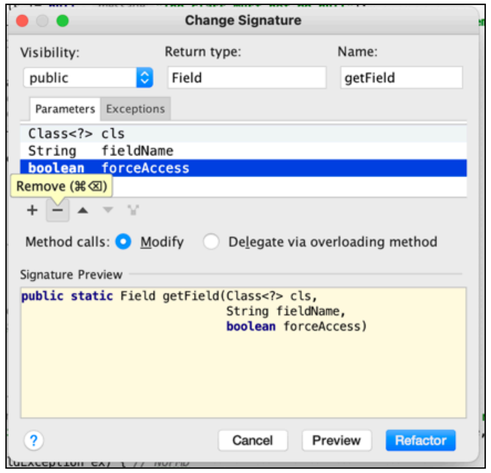
Tool support

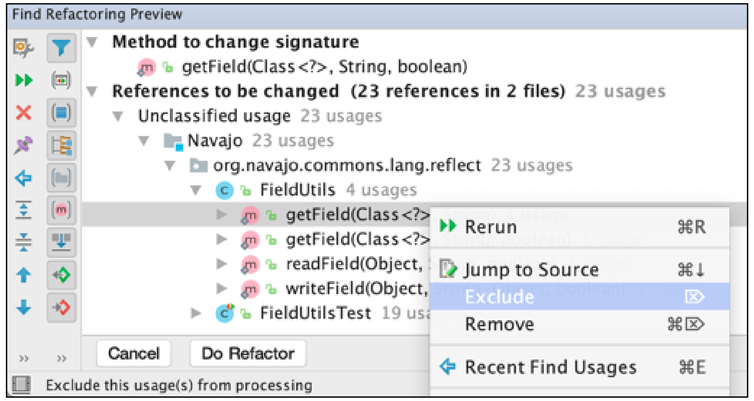


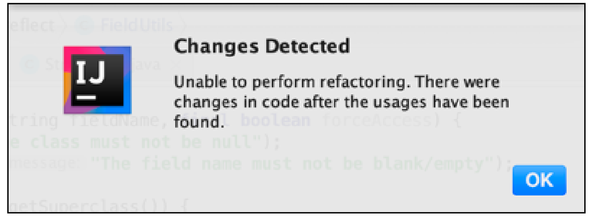
Tool support

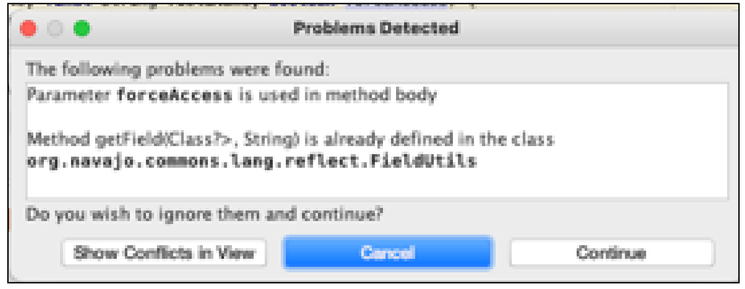




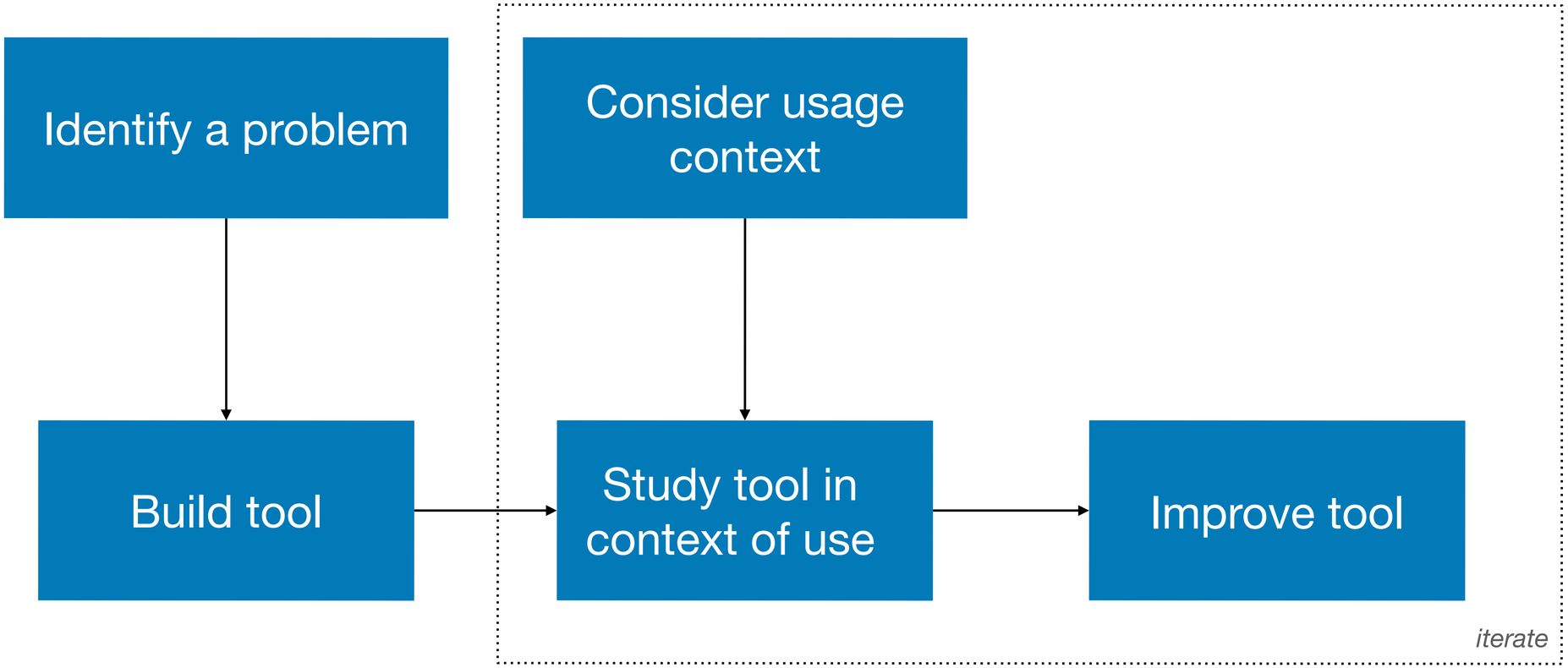




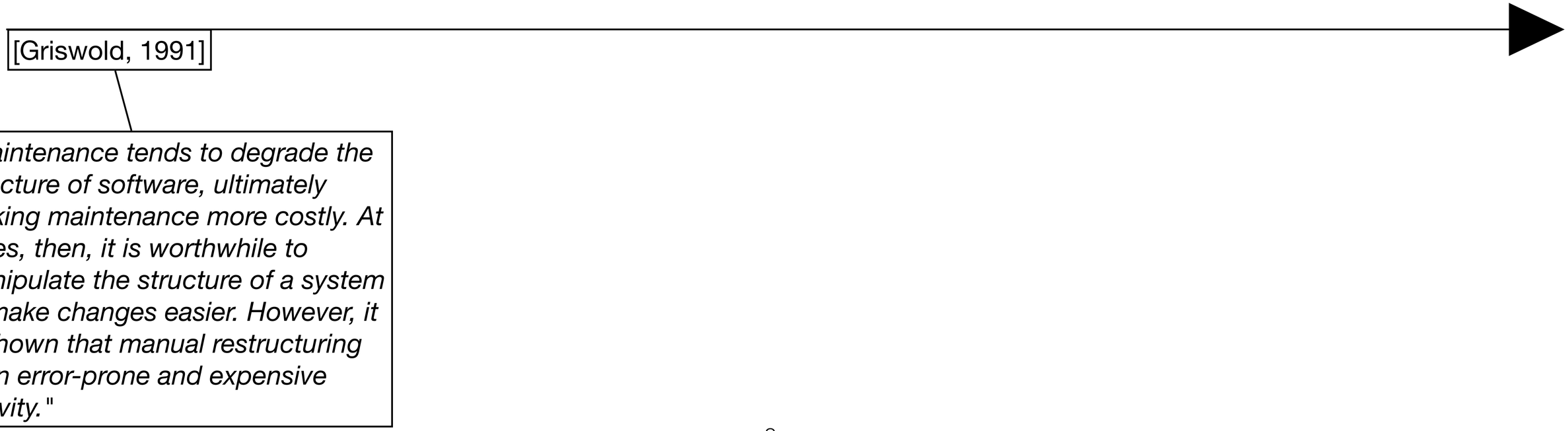




Tool design


Identify a problem
Consider usage context
Build tool
Study tool in context of use
Improve tool
iterate

Refactoring tools



Refactoring tools

"*Refactorings are reorganization plans that support change at an intermediate level.*"

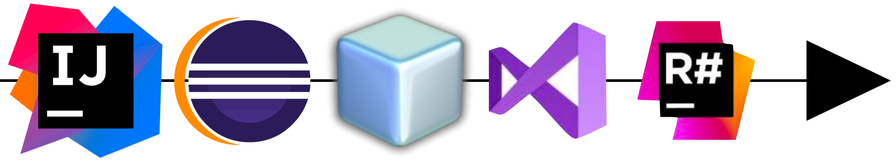
[Opdyke, 1992]

Catalogue of 72 refactorings

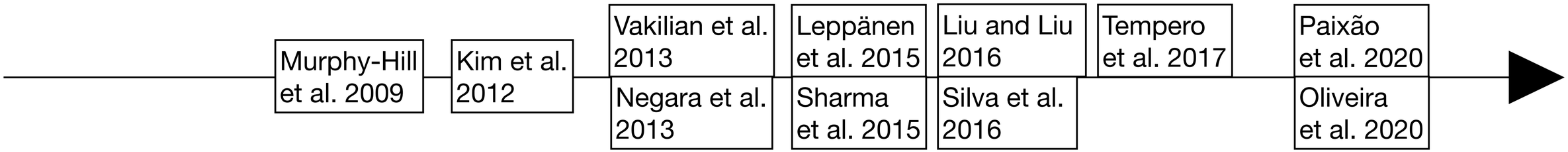
[Fowler, 1999]

[Roberts, 99]

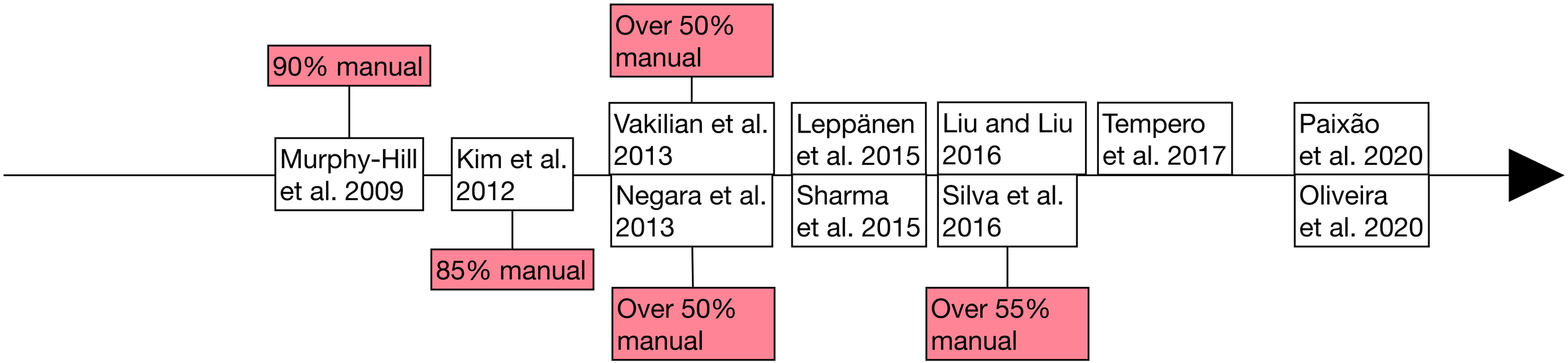
"*It has typically been done manually, or with the help of primitive tools such as text editors with search and replace. (...) Our approach is to create tools to assist a human in refactoring code (..)*"



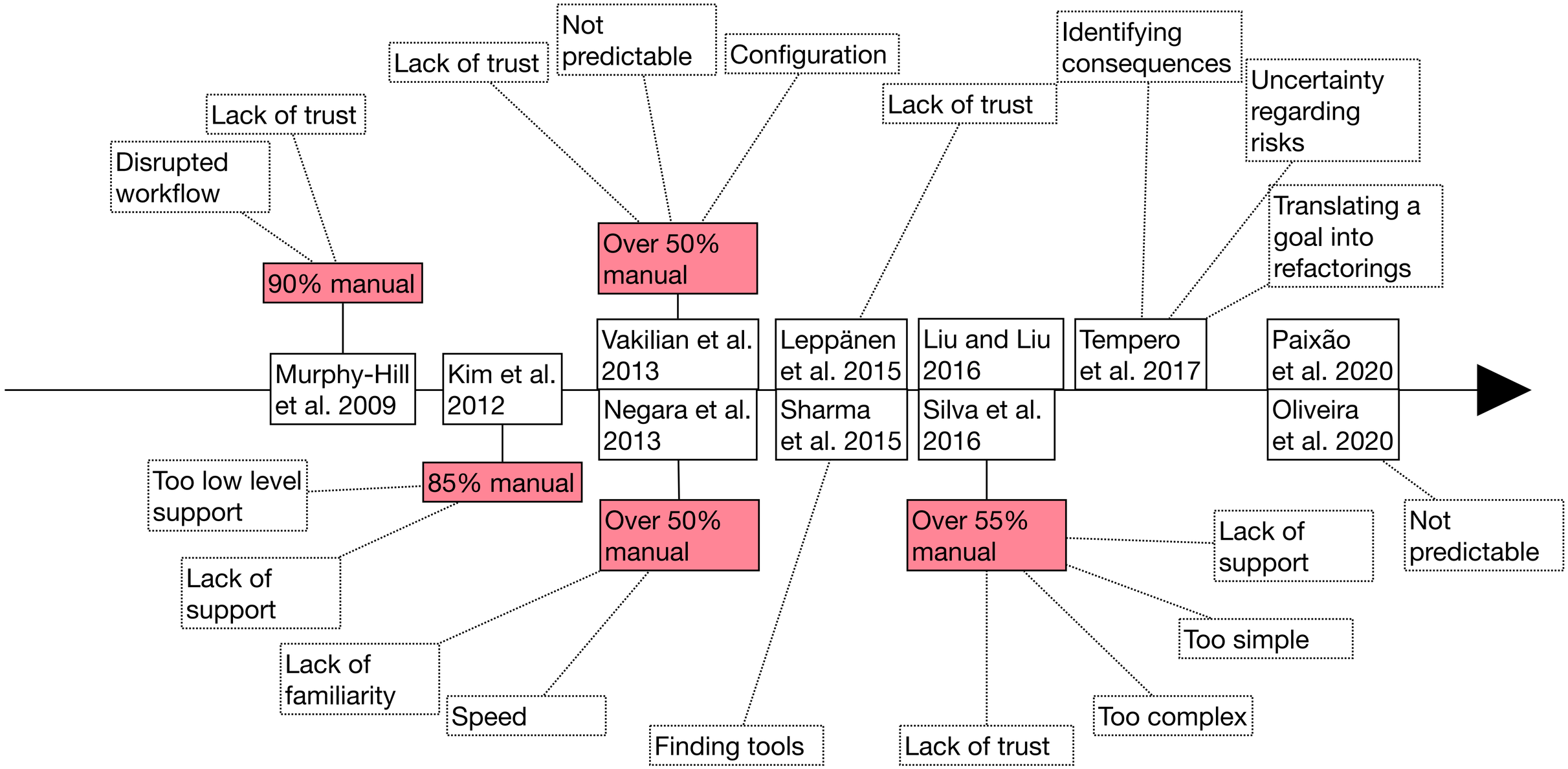
Refactoring tools



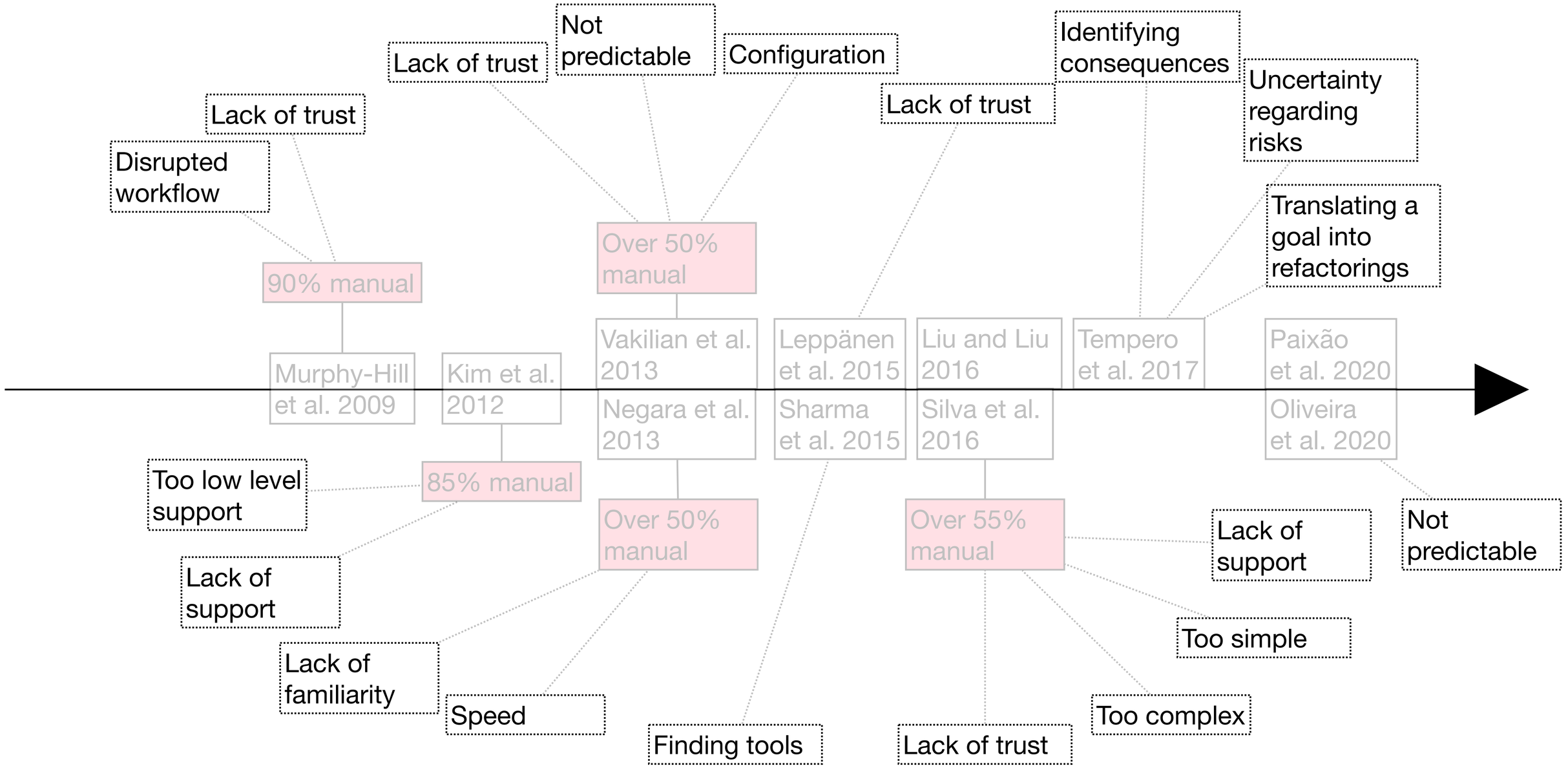
Refactoring tools



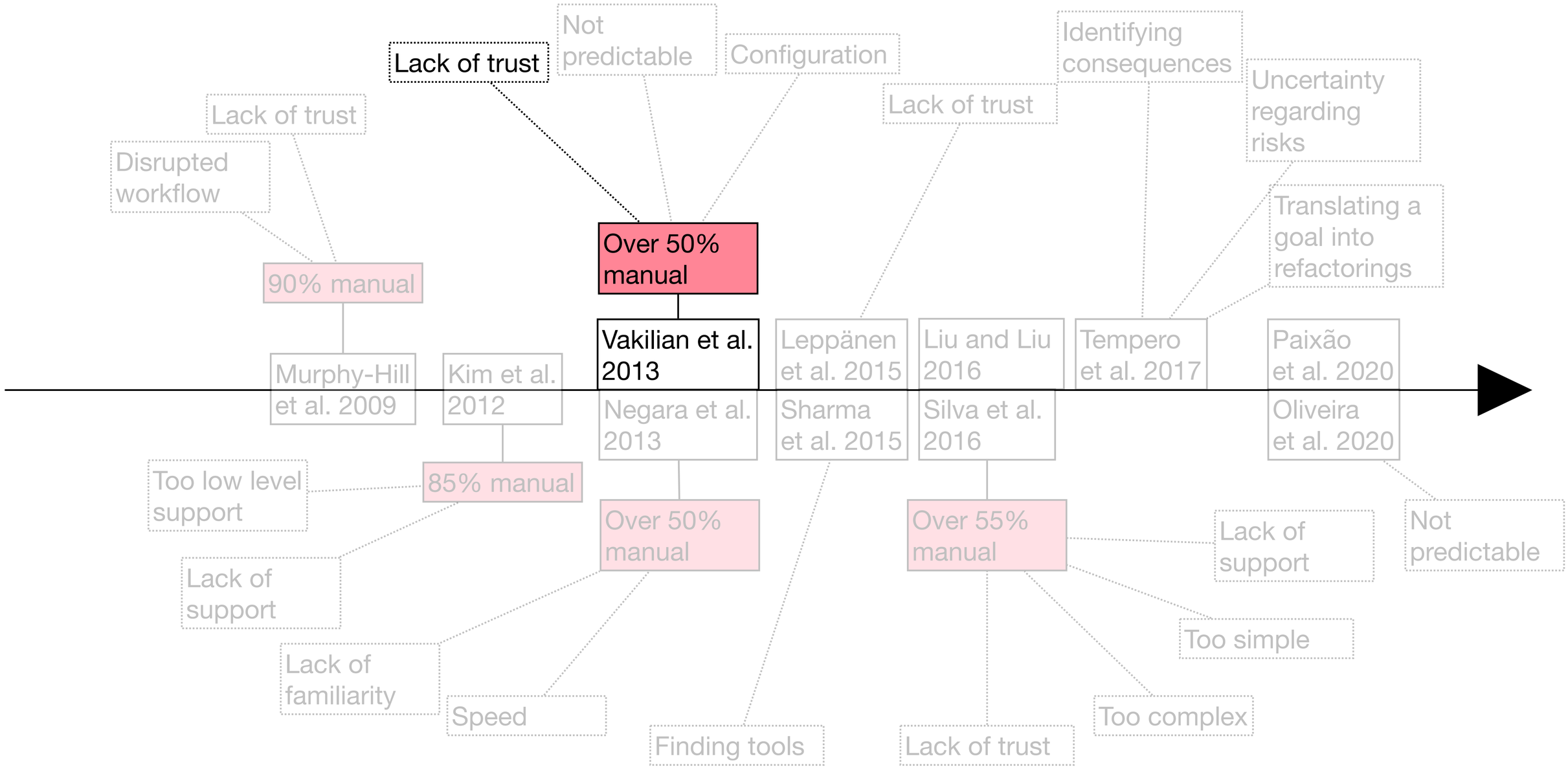
Refactoring tools



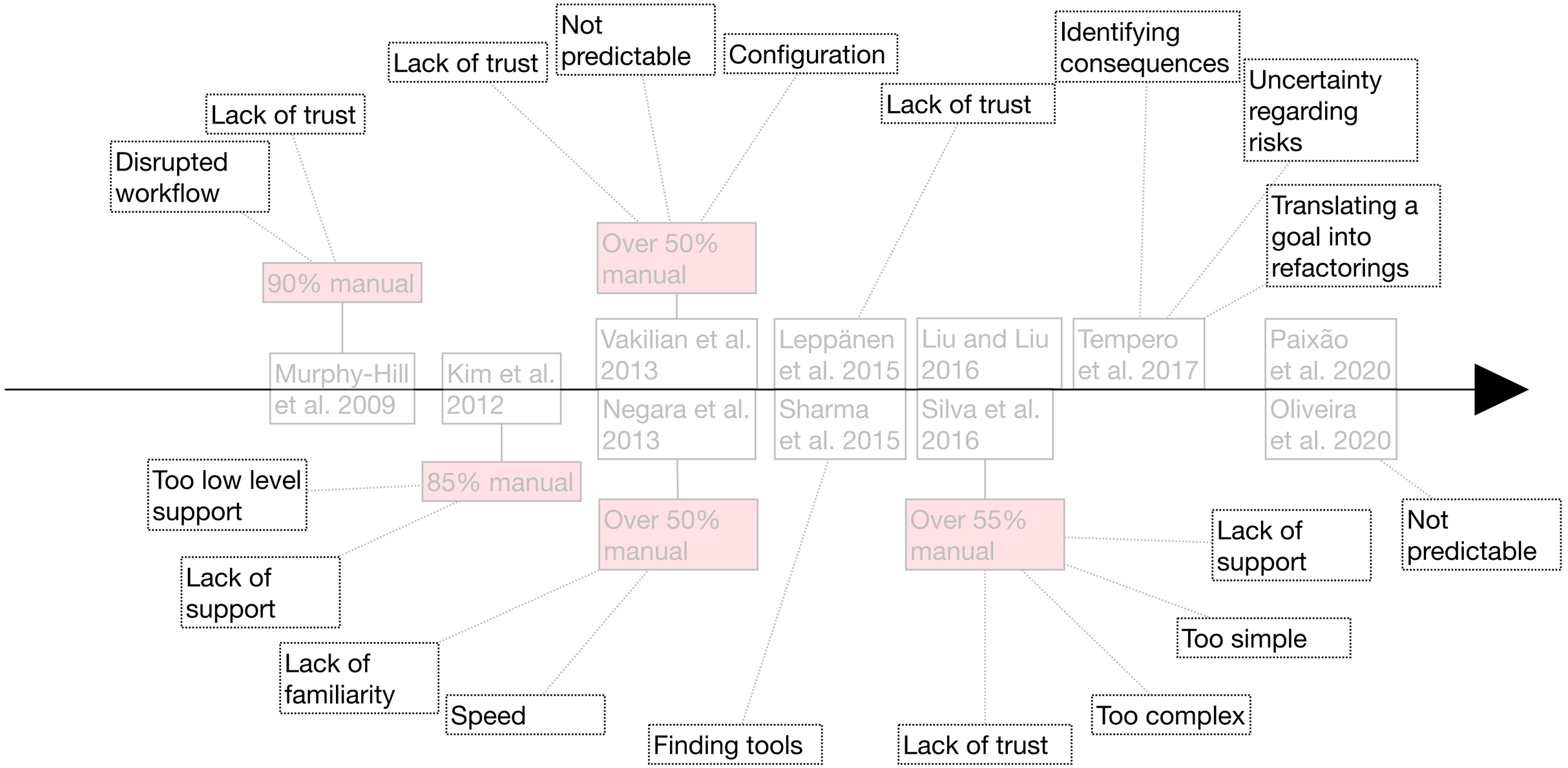
Refactoring tools



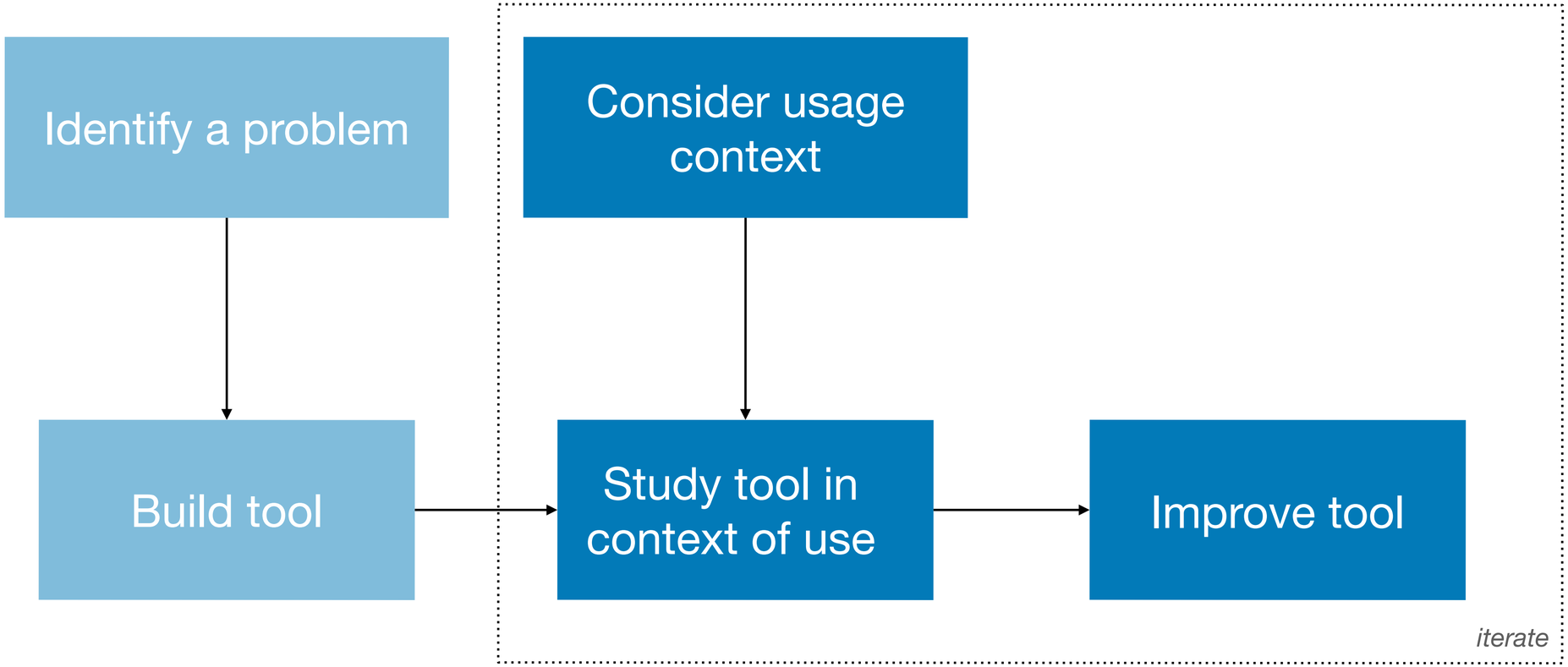
Refactoring tools



Refactoring tools



Improving the Usability of Refactoring Tools for Software Change Tasks



Improving the Usability of Refactoring Tools for Software Change Tasks

Improving the Usability of Refactoring Tools for Software Change Tasks

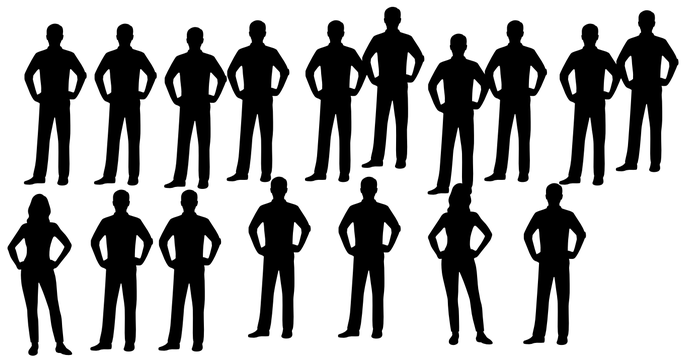
Paper		Pages
I	A Study of Refactorings During Software Change Tasks	31 - 71
II	Refactoring Tool Usability	73 - 88
III	Usability Profiles of Refactoring Tools	91 - 114
IV	Stepwise Refactoring Tools	115 - 140

Study design

Navajo

78K lines (Java version 1.8)

1335 JUnit tests



Move Method 1

Inline Method 2

Remove Parameter 3

- 32 hours recording
- 17 transcripts
- 134,947 words
- 100 refactoring tool invocations
- 385 detected refactorings (manual&tool)

Summary of participant workflow

Intent	"My long-term goal is that I want to inline this method. So by keeping this kind of stub implementation of it - stub isn't the right word - I'm preserving it as long as possible so any tests calling it [are] still working while I do my refactoring. What I want to do is move the implementation into the method I am going to keep and ensure that the tests are all passing and then remove the method I want to remove."
Approach Summary	Forms plan upfront that includes Inline Method. Changes plan when encountering tool error. Isolate changes and keeps tests running by introducing a shared dependency as an intermediate step. Manual inline, keep old code in method to validate (compare). Validates change by running tests and comparing changes in git. Guides steps by Find Usages.
Refactoring Tools	
Mentioned:	Inline Method
Mentioned prompted:	
Tried:	Inline Method
Tried prompted:	
Used to solve task:	None
Other Tools:	Find Usages
Guide changes:	Find Usages, Avoid compiler errors
Verify:	Dynamic after each change, git diff

Figure 3.1: Task analysis schema for P_9 on Task-2.

Summary of participant workflow

Intent	"My long-term goal is that I want to inline this method. So by keeping this kind of stub implementation of it - stub isn't the right word - I'm preserving it as long as possible so any tests calling it [are] still working while I do my refactoring. What I want to do is move the implementation into the method I am going to keep and ensure that the tests are all passing and then remove the method I want to remove."
Approach Summary	Forms plan upfront that includes Inline Method. Changes plan when encountering tool error. Isolate changes and keeps tests running by introducing a shared dependency as an intermediate step. Manual inline, keep old code in method to validate (compare). Validates change by running tests and comparing changes in git. Guides steps by Find Usages.
Refactoring Tools	
Mentioned:	Inline Method
Mentioned prompted:	
Tried:	Inline Method
Tried prompted:	
Used to solve task:	None
Other Tools:	Find Usages
Guide changes:	Find Usages, Avoid compiler errors
Verify:	Dynamic after each change, git diff

Figure 3.1: Task analysis schema for P_9 on Task-2.

Summary of participant workflow

Intent	"My long-term goal is that I want to inline this method. So by keeping this kind of stub implementation of it - stub isn't the right word - I'm preserving it as long as possible so any tests calling it [are] still working while I do my refactoring. What I want to do is move the implementation into the method I am going to keep and ensure that the tests are all passing and then remove the method I want to remove."
Approach Summary	Forms plan upfront that includes Inline Method. Changes plan when encountering tool error. Isolate changes and keeps tests running by introducing a shared dependency as an intermediate step. Manual inline, keep old code in method to validate (compare). Validates change by running tests and comparing changes in git. Guides steps by Find Usages.
Refactoring Tools	
Mentioned:	Inline Method
Mentioned prompted:	
Tried:	Inline Method
Tried prompted:	
Used to solve task:	None
Other Tools:	Find Usages
Guide changes:	Find Usages, Avoid compiler errors
Verify:	Dynamic after each change, git diff

Figure 3.1: Task analysis schema for P_9 on Task-2.

Summary of participant workflow

Intent	"My long-term goal is that I want to inline this method. So by keeping this kind of stub implementation of it - stub isn't the right word - I'm preserving it as long as possible so any tests calling it [are] still working while I do my refactoring. What I want to do is move the implementation into the method I am going to keep and ensure that the tests are all passing and then remove the method I want to remove."
Approach Summary	Forms plan upfront that includes Inline Method. Changes plan when encountering tool error. Isolate changes and keeps tests running by introducing a shared dependency as an intermediate step. Manual inline, keep old code in method to validate (compare). Validates change by running tests and comparing changes in git. Guides steps by Find Usages.
Refactoring Tools	
Mentioned:	Inline Method
Mentioned prompted:	
Tried:	Inline Method
Tried prompted:	
Used to solve task:	None
Other Tools:	Find Usages
Guide changes:	Find Usages, Avoid compiler errors
Verify:	Dynamic after each change, git diff

Figure 3.1: Task analysis schema for P_9 on Task-2.

Summary of participant workflow

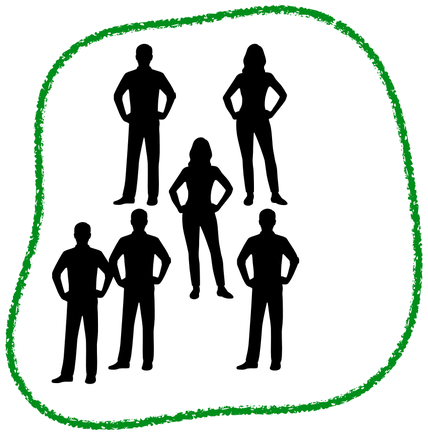
Intent	"My long-term goal is that I want to inline this method. So by keeping this kind of stub implementation of it - stub isn't the right word - I'm preserving it as long as possible so any tests calling it [are] still working while I do my refactoring. What I want to do is move the implementation into the method I am going to keep and ensure that the tests are all passing and then remove the method I want to remove."
Approach Summary	Forms plan upfront that includes Inline Method. Changes plan when encountering tool error. Isolate changes and keeps tests running by introducing a shared dependency as an intermediate step. Manual inline, keep old code in method to validate (compare). Validates change by running tests and comparing changes in git. Guides steps by Find Usages.
Refactoring Tools	
Mentioned:	Inline Method
Mentioned prompted:	
Tried:	Inline Method
Tried prompted:	
Used to solve task:	None
Other Tools:	Find Usages
Guide changes:	Find Usages, Avoid compiler errors
Verify:	Dynamic after each change, git diff

Figure 3.1: Task analysis schema for P_9 on Task-2.

Strategies

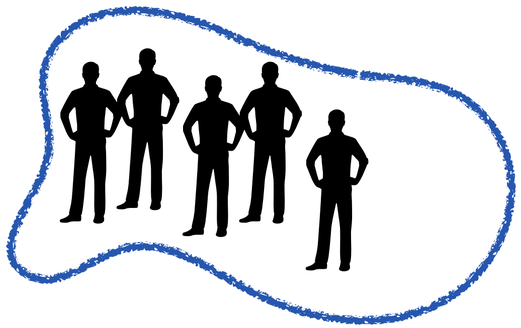
Local

"So what I'm going to do now is rely on my friendly compiler to do this. So now things are going to break"



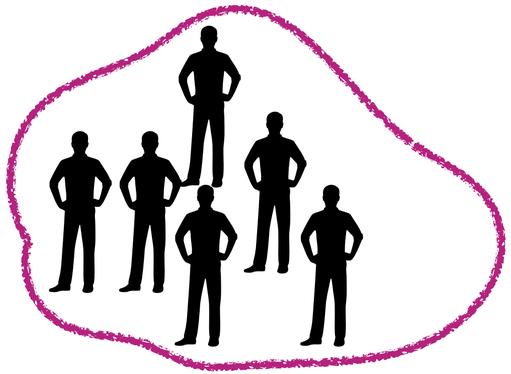
Structure

"Any invocations would need to be dealt with. So what I'd like to do, is look in StringUtils, look where these are used (...)"

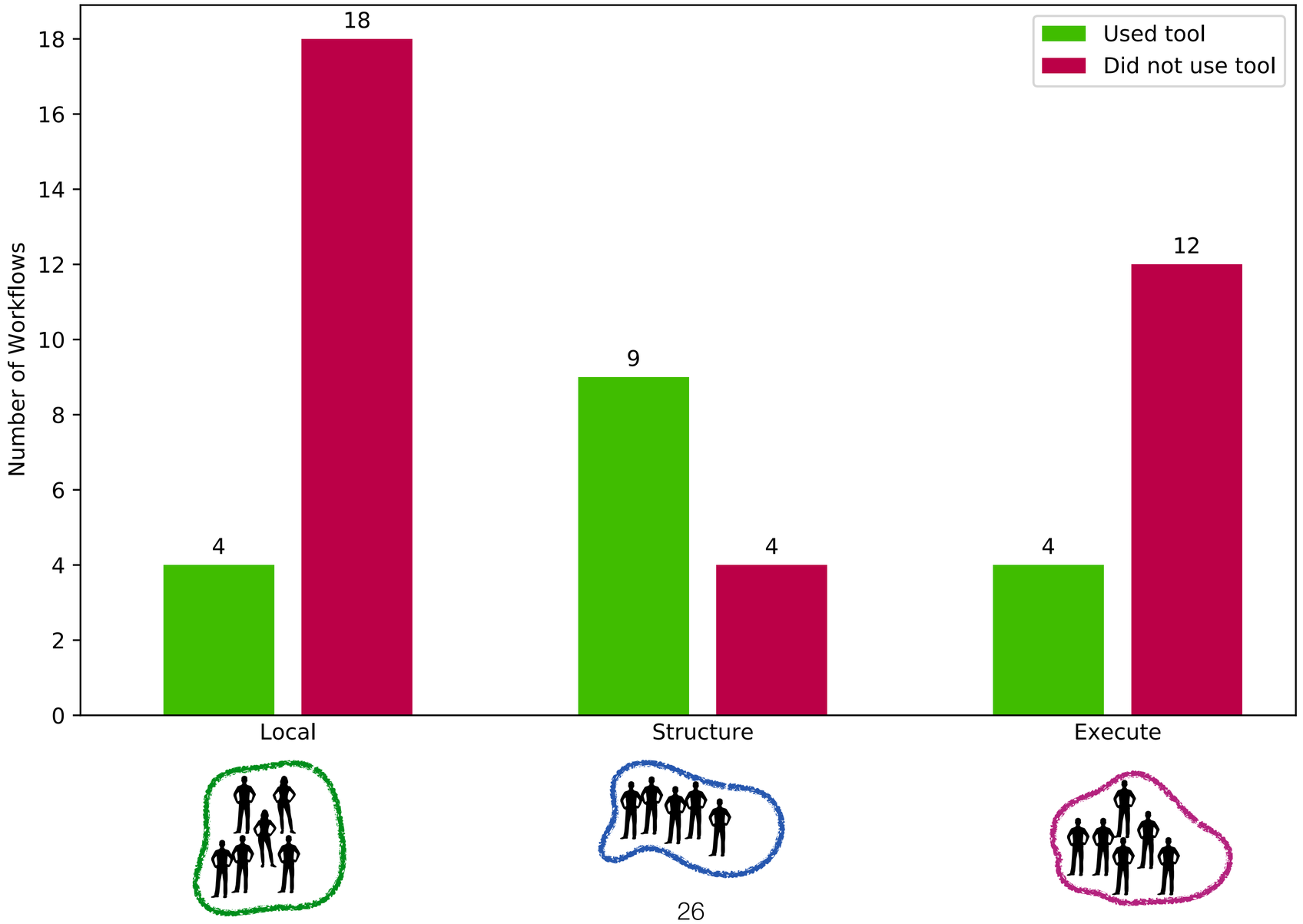


Execute

"(...) change the internal definition of the methods first so that it always executes as if it was false. That will reveal the test cases that I need to look at more closely".



Refactoring tool use



Summary of contributions

- An experimental system that can be reused by other researchers to study realistic software change tasks that contain steps amenable to refactoring tools.
- A dataset comprising 32 hours of screen recordings from 17 experienced practitioners with time stamped transcripts totaling 130k words and 100 refactoring invocations.
- Three strategies that were observed to be used by developers to make progress in functional software change tasks: Local, Structure, and Execute.

Improving the Usability of Refactoring Tools for Software Change Tasks

Usability

"extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"

(ISO 9241-11:2018, Part 11, 3.1.1)

Refactoring Tool Usability

Software developers employ refactoring tools to help prepare or complete functional changes to a software system. Software developers seek these tools to be reasonable to locate and apply for a desired intent (effective), reasonable in terms of time and effort required to use the tool (efficient), and to add value to the development process (satisfying).

Coding of transcripts

Codebook

Label	Pattern
Effective	**User experiences the tool as successfully performing a desired result.**
Efficient	**User experiences the tool as being fast or increasing productivity.**
Satisfactory	**User experiences having their needs or wants met by the use of the tool.**
Trust	**User experiences the tool as trustworthy, safe or reliable.**
Predictable	**User understands up-front what will happen by using the tool or indicates a lack of surprise when using the tool.**

Coding of transcripts

I: So the refactoring did break the code but it was what you expected nonetheless.

Yes it was more helpful for the refactoring to result in code that did not compile than to say I can not do that.

I: how come?

It was more in line with the change I would have done manually. I would make the change, see that there was an in-collision and then resolve the collision. Asking me to modify unrelated code in order to delete the unrelated code feels like an extra step.

- P9, T3

Coding of transcripts

I: So the refactoring did break the code but it was what you expected nonetheless.	
Yes it was more helpful for the refactoring to result in code that did not compile than to say I can not do that.	**Predictable**
I: how come?	
It was more in line with the change I would have done manually. I would make the change, see that there was an in-collision and then resolve the collision. Asking me to modify unrelated code in order to delete the unrelated code feels like an extra step.	

- P9, T3

Coding of transcripts

I: So the refactoring did break the code but it was what you expected nonetheless.	
Yes it was more helpful for the refactoring to result in code that did not compile than to say I can not do that.	**Predictable** **Satisfaction**
I: how come?	
It was more in line with the change I would have done manually. I would make the change, see that there was an in-collision and then resolve the collision. Asking me to modify unrelated code in order to delete the unrelated code feels like an extra step.	

- P9, T3

Coding of transcripts

I: So the refactoring did break the code but it was what you expected nonetheless.	
Yes it was more helpful for the refactoring to result in code that did not compile than to say I can not do that.	**Predictable** **Satisfaction**
I: how come?	
It was more in line with the change I would have done manually. I would make the change, see that there was an in-collision and then resolve the collision. Asking me to modify unrelated code in order to delete the unrelated code feels like an extra step.	**Efficient** (“Asking me to modify unrelated code in order to delete the unrelated code feels like an extra step.”)

- P9, T3

Card sort

256 Cards

Predictable

I: So the refactoring did break the code but it was what you expected nonetheless.

Yes it was more helpful for the refactoring to result in code that did not compile than to say I can not do that.

Satisfaction

I: So the refactoring did break the code but it was what you expected nonetheless.

Yes it was more helpful for the refactoring to result in code that did not compile than to say I can not do that.

Efficient

I: So the refactoring did break the code

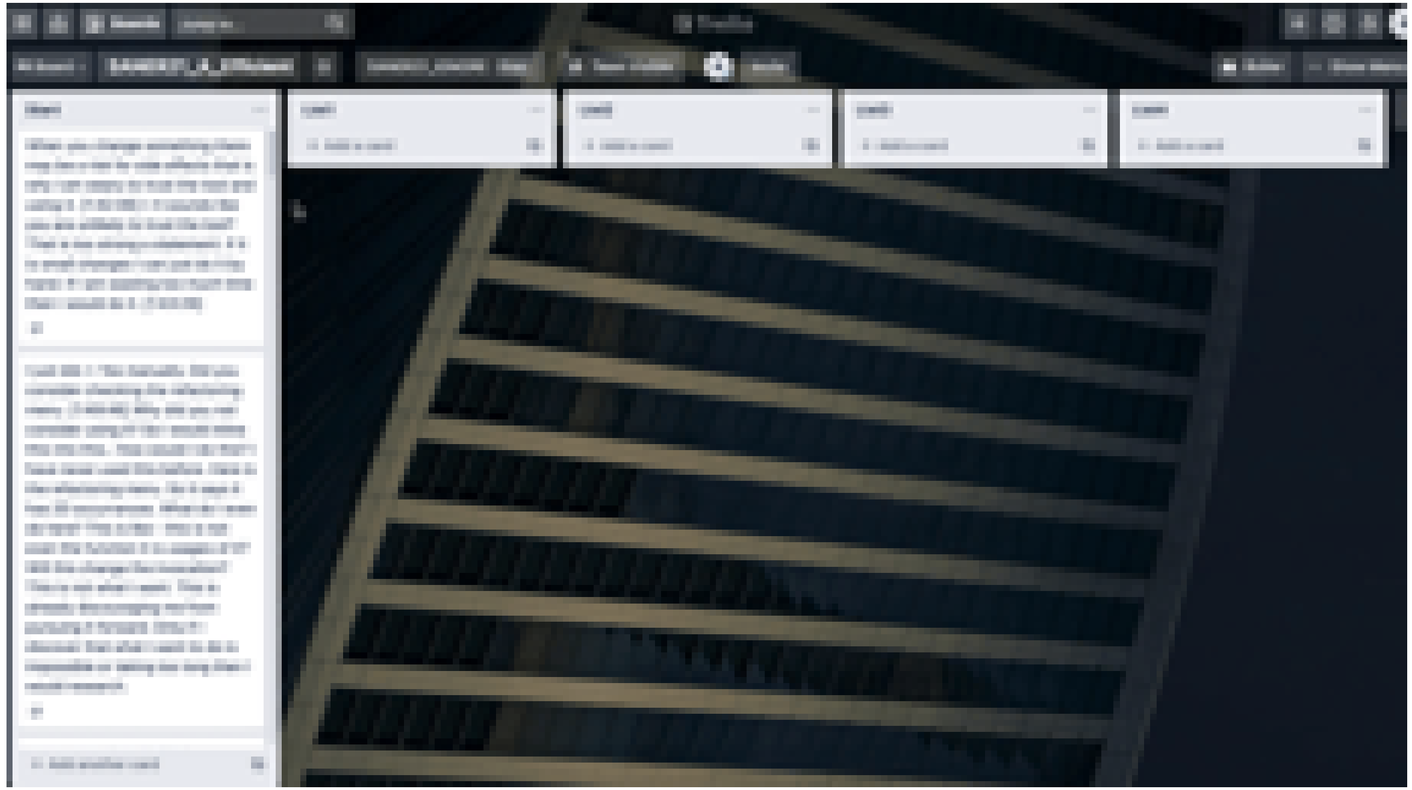
Yes it was more helpful for the refactoring to result in code that did not compile than to say I

I: how come?

It was more in line with the change I would have done manually. I would make the change,

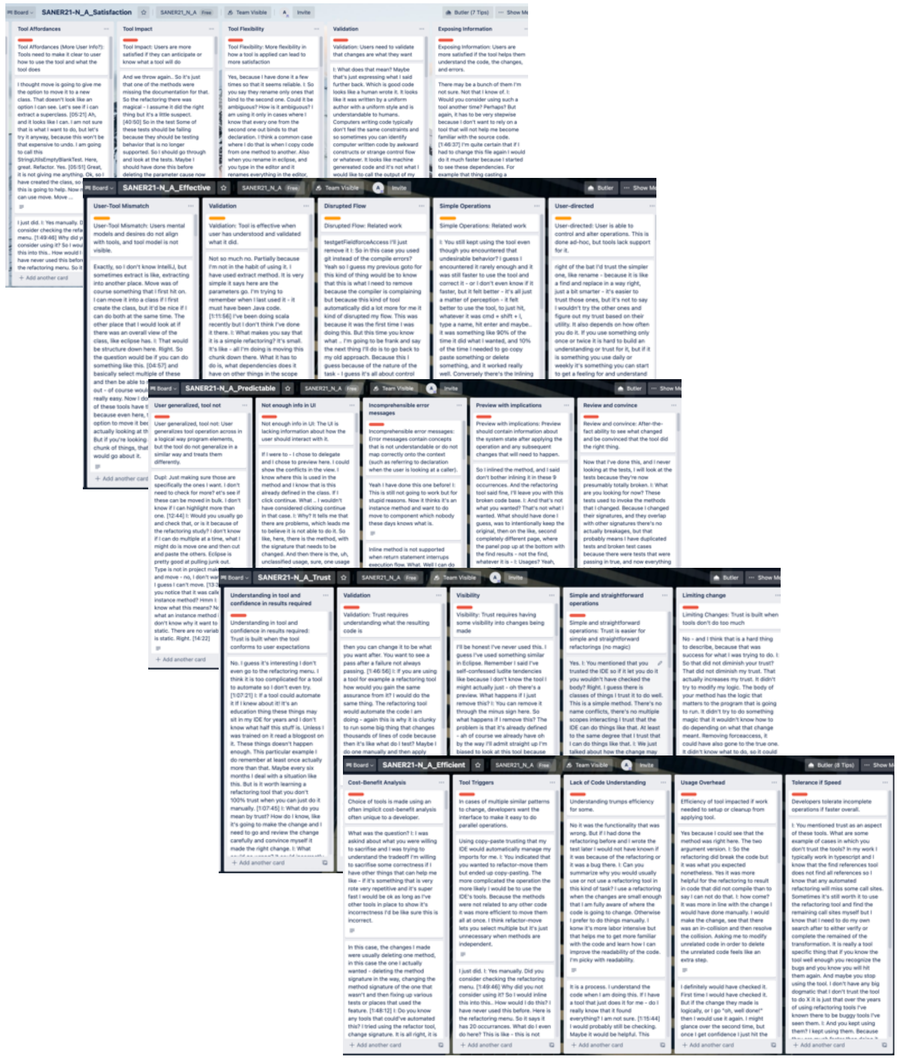
Card sort

Predictable	79
Effective	53
Satisfaction	51
Efficient	42
Trust	31

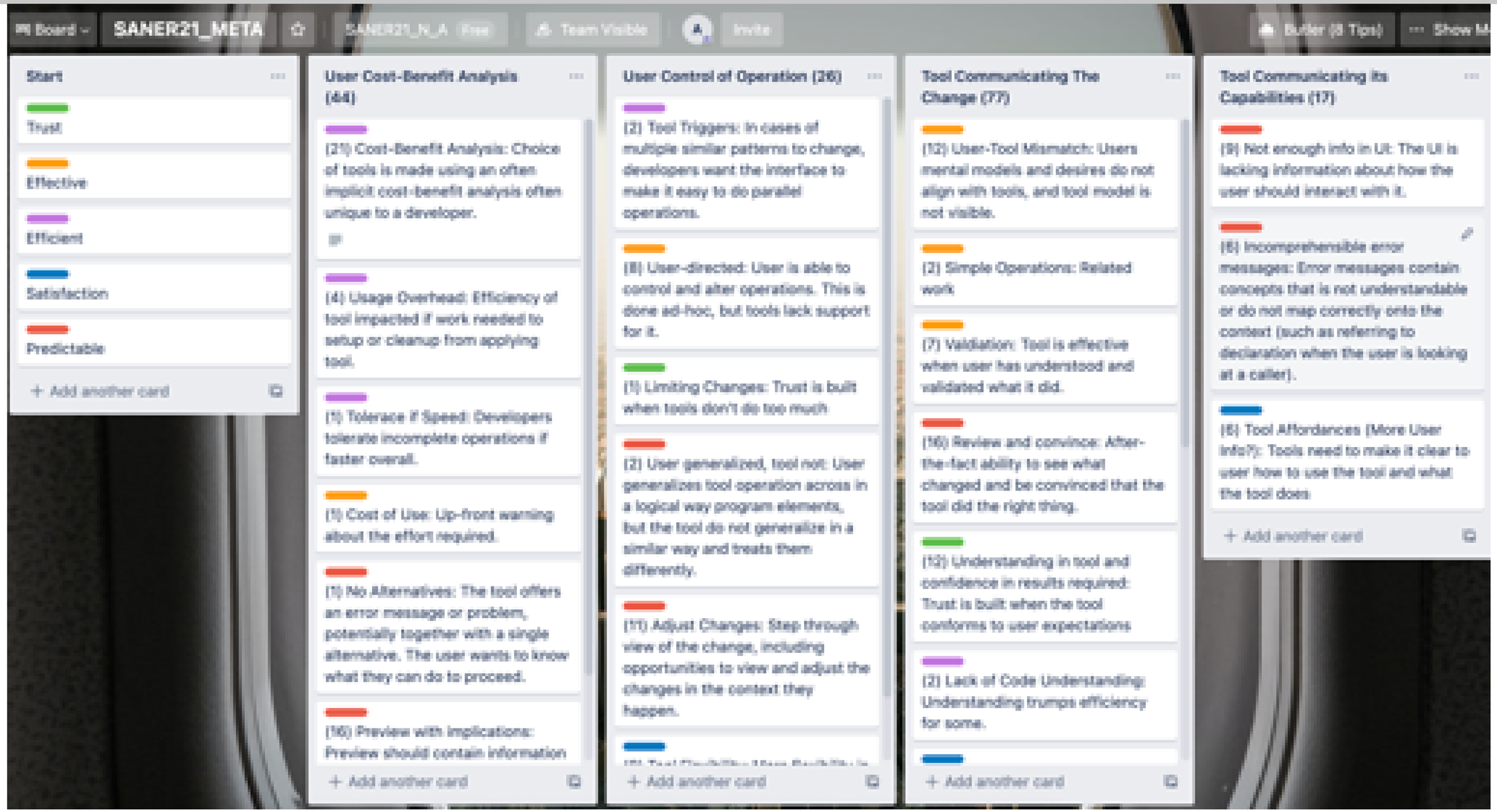


Card sort

	Cards: 256	Themes: 28
Predictable	79	7
Effective	53	6
Satisfaction	51	5
Efficient	42	5
Trust	31	5



Meta card sort



Usability themes

Theme	Description	PRD	EFE	SAT	EFI	TRS
Tool communicates capabilities	A tool's user interface must communicate clearly and directly to a developer in terms familiar to a developer. The tool should guide a developer in its use, including providing intelligible error messages.	✅		✅		
Tool communicates change	A tool should make the changes it will make to code clear before the tool is executed. It should be possible for a developer to inspect the changes that the tool has made.	✅	✅	✅		✅
Developer guides tool	Developers wish to guide how a tool executes in several ways: applying an operation to many elements easily, excluding the application to some locations of code, and altering how particular code is changed.	✅	✅	✅	✅	✅
Developer cost-benefit analysis	Developers assess the cost of applying a tool before they invoke the tool. Developers want to assess whether it is better to proceed with a tool or manually at the start of considering using a tool. The tool should help them with this assessment.	✅	✅		✅	

Usability themes

Theme	Description	PRD	EFE	SAT	EFI	TRS
Tool communicates capabilities	A tool's user interface must communicate clearly and directly to a developer in terms familiar to a developer. The tool should guide a developer in its use, including providing intelligible error messages.	✅		✅		
Tool communicates change	A tool should make the changes it will make to code clear before the tool is executed. It should be possible for a developer to inspect the changes that the tool has made.	✅	✅	✅		✅
Developer guides tool	Developers wish to guide how a tool executes in several ways: applying an operation to many elements easily, excluding the application to some locations of code, and altering how particular code is changed.	✅	✅	✅	✅	✅
Developer cost-benefit analysis	Developers assess the cost of applying a tool before they invoke the tool. Developers want to assess whether it is better to proceed with a tool or manually at the start of considering using a tool. The tool should help them with this assessment.	✅	✅		✅	

Paper II

Refactoring Tool Usability: Refined

Software developers employ refactoring tools to help prepare or complete functional changes to a software system.

Software developers seek these tools to be reasonable to locate, and for the tools to help them assess the efficiency of the tool, in terms of the costs and benefits of the tool, before its use.

To enable effective use in multiple situations, software developers seek to guide how a tool changes the source code for a system; this ability to tailor how a tool works can improve the efficiency of the tool for the developer.

Software developers also seek refactoring tools to explain their impact to source code so that the software developer can understand the effectiveness of the tool.

Software developers also expect tools to communicate clearly and directly in terms that match how software developers perceive refactoring operations.

These characteristics in a refactoring tool increase the satisfaction of the software developer using a refactoring tool.

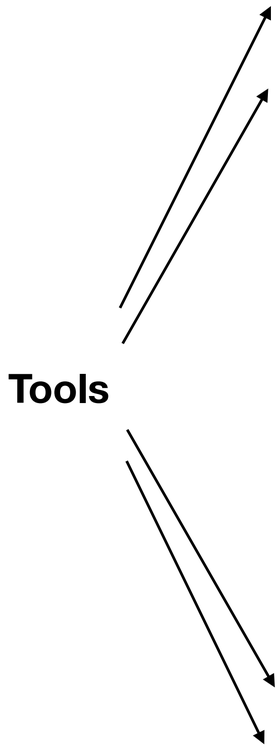
Summary of contributions

- A theory of usability for refactoring tools.
- Four usability themes that illustrate problem areas of refactoring tools in the context of software change tasks.

Improving the Usability of Refactoring Tools for Software Change Tasks

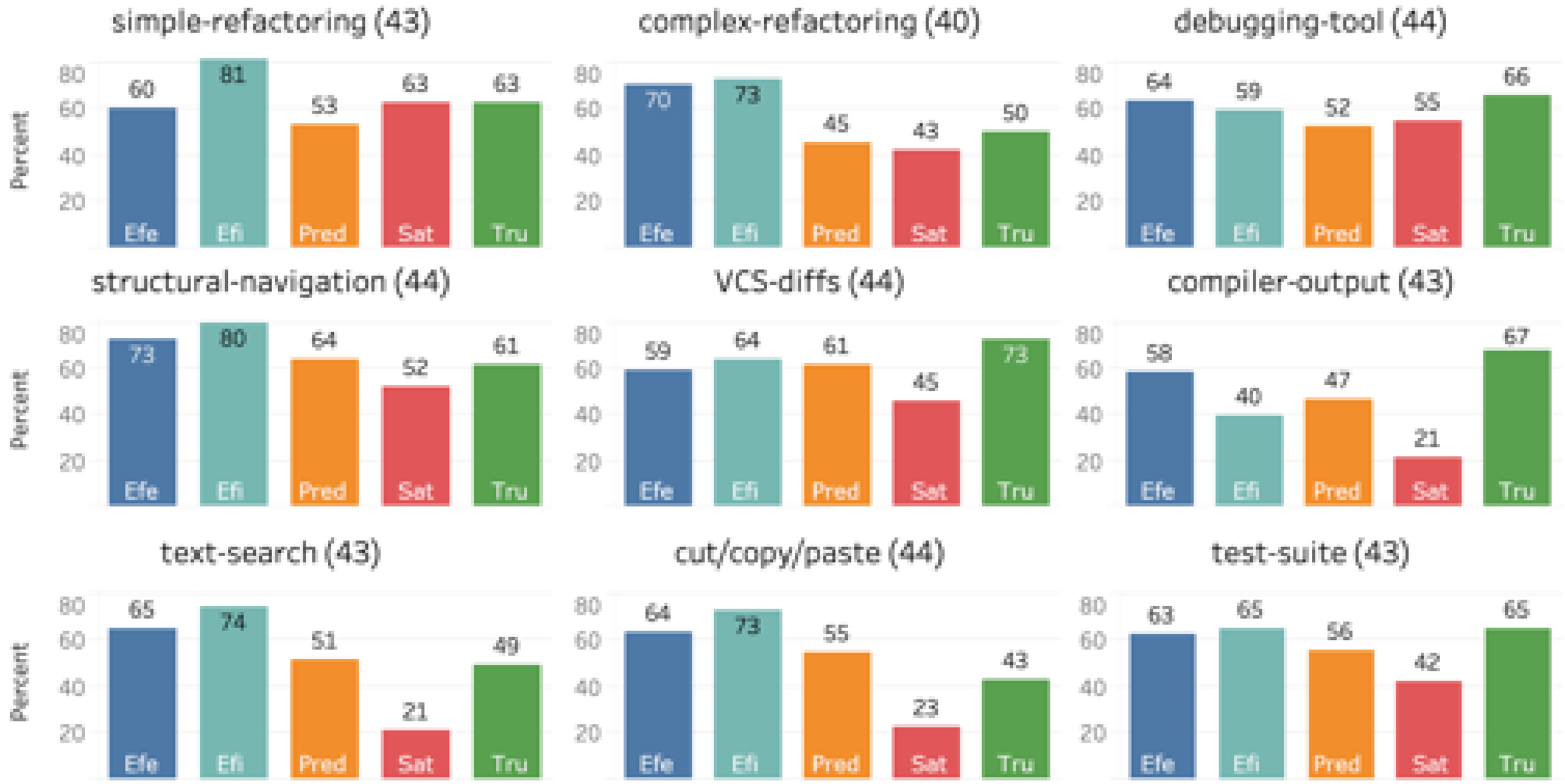
Paper III

Usability factors considered necessary

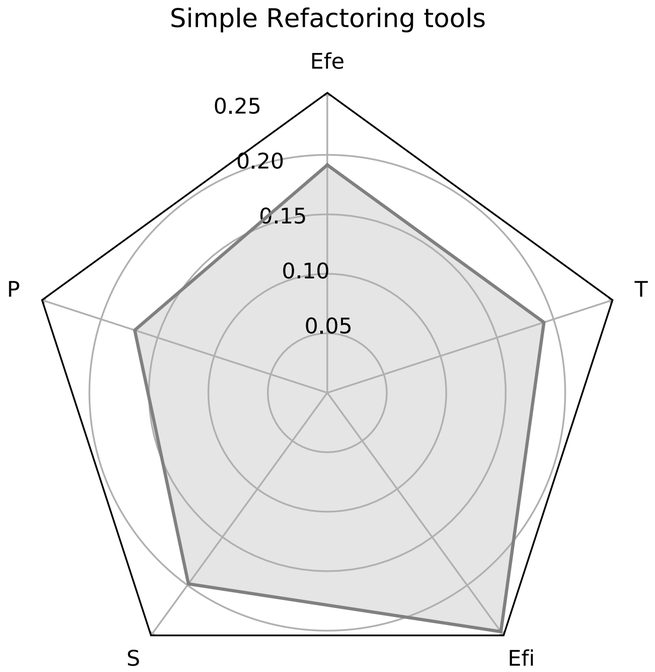


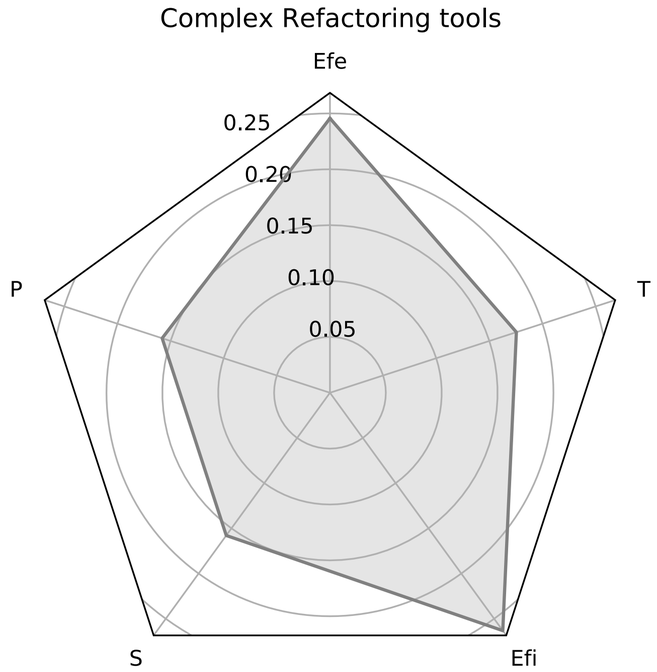
	The tool is effective	The tool saves time or effort	The tool is satisfying to use	The tool is trustworthy	The tool is predictable
Complex Refactoring tools (e.g. Move Method, Extract Class, Introduce Parameter)	☐	☐	☐	☐	☐
Compiler output (e.g. compiler errors)	☐	☐	☐	☐	☐
Version Control Systems "diffs" (e.g. git diff, et.c.)	☐	☐	☐	☐	☐
Structural navigation (e.g. find references, go to declaration)	☐	☐	☐	☐	☐
Debugging tools	☐	☐	☐	☐	☐
Copy/cut and paste code	☐	☐	☐	☐	☐
Test suites (e.g. JUnit, test errors)	☐	☐	☐	☐	☐
Simple Refactoring tools (e.g. Rename, Extract Constant, Inline Method)	☐	☐	☐	☐	☐
Textual search (e.g. grep, find and replace)	☐	☐	☐	☐	☐

Variability between tools

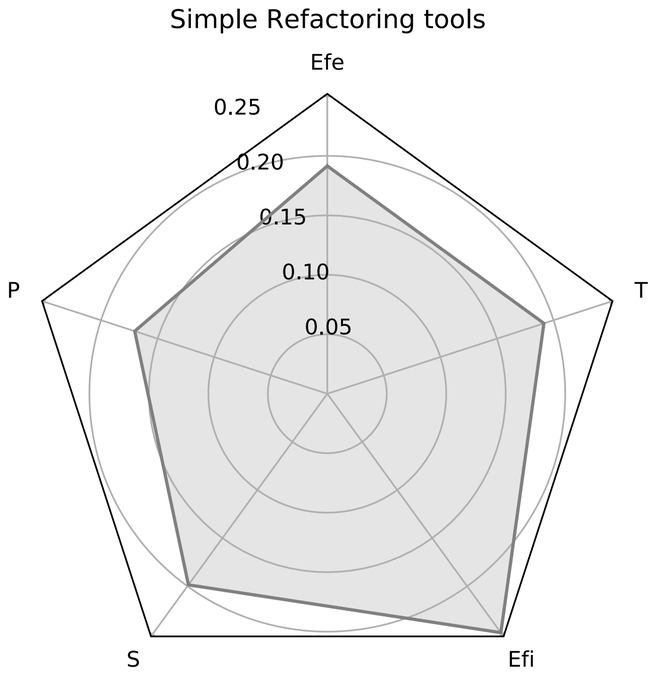


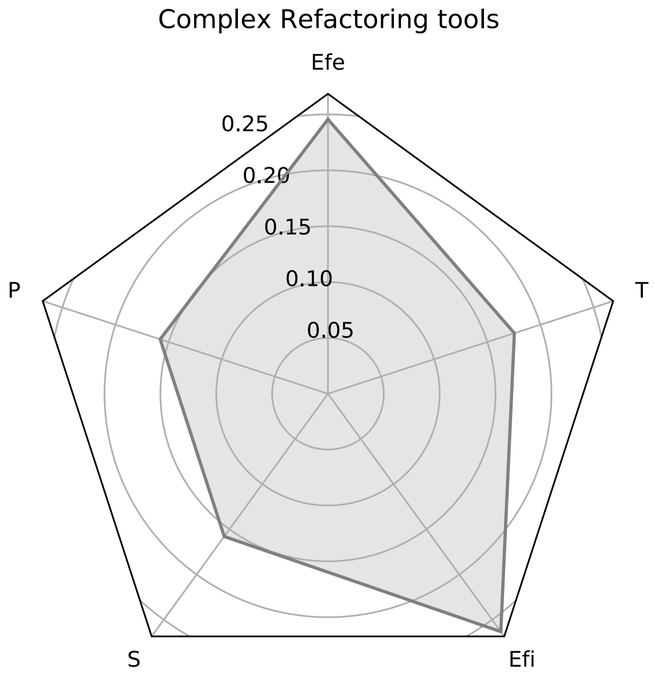
Usability Profiles

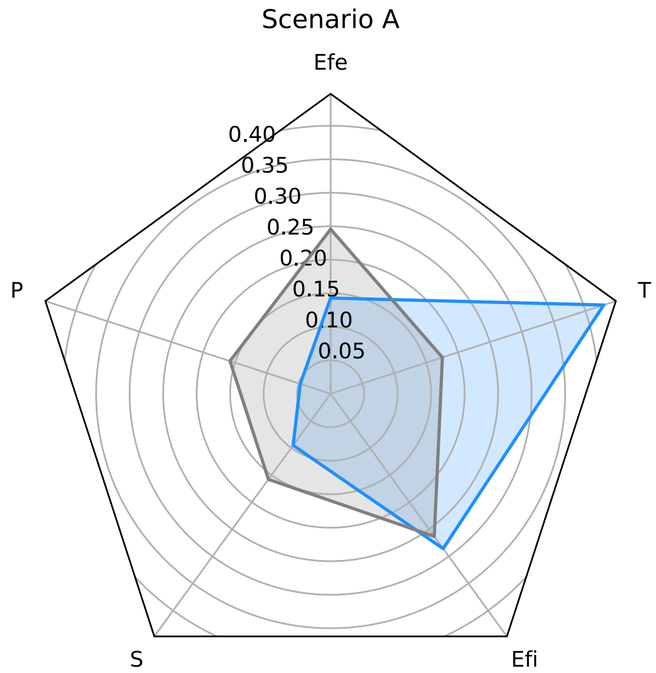




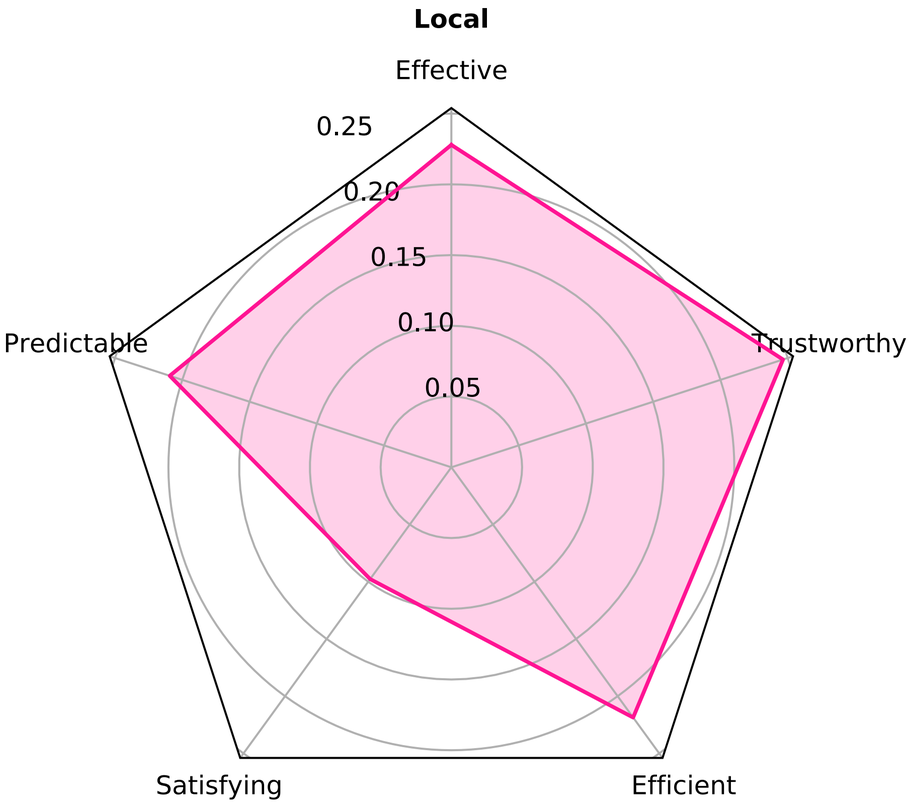
Usability Profiles

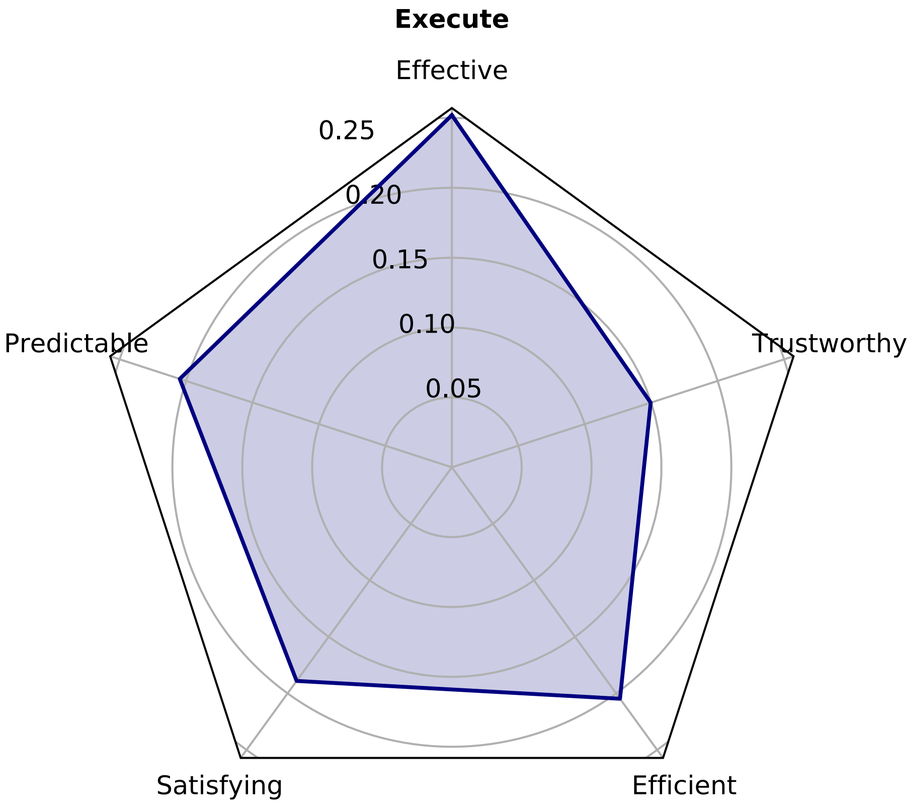






Variability between strategies





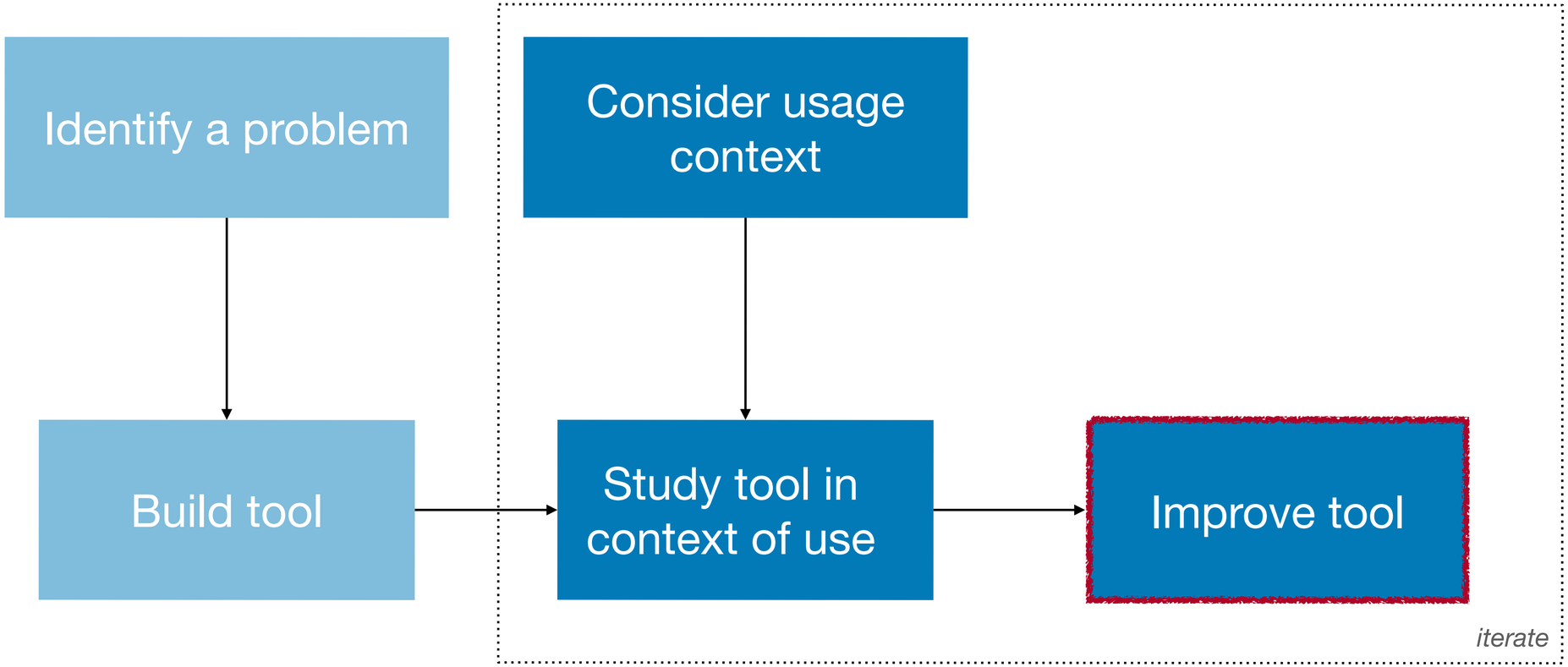
Summary of contributions

- Usability profiles of nine types of tools that developers find useful for approaching software change tasks, including simple and complex refactoring tools.

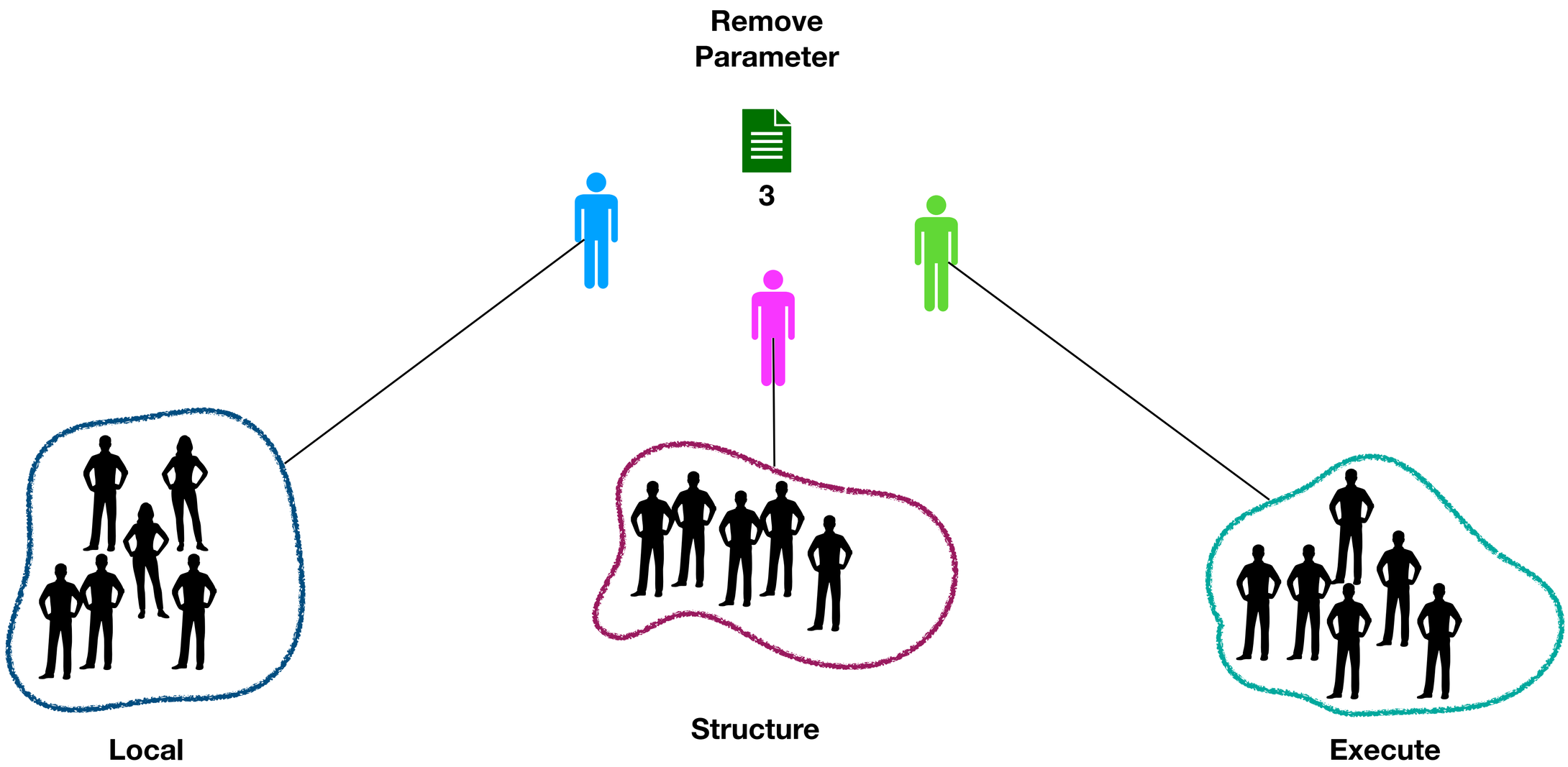
Improving the Usability of Refactoring Tools for Software Change Tasks

Paper IV

Improve refactoring tools

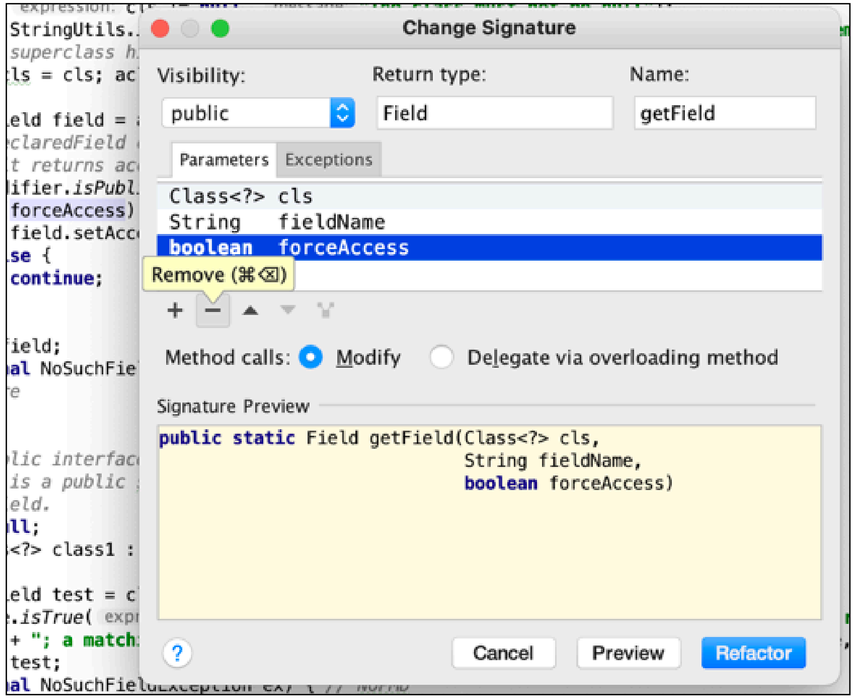


Usability analysis

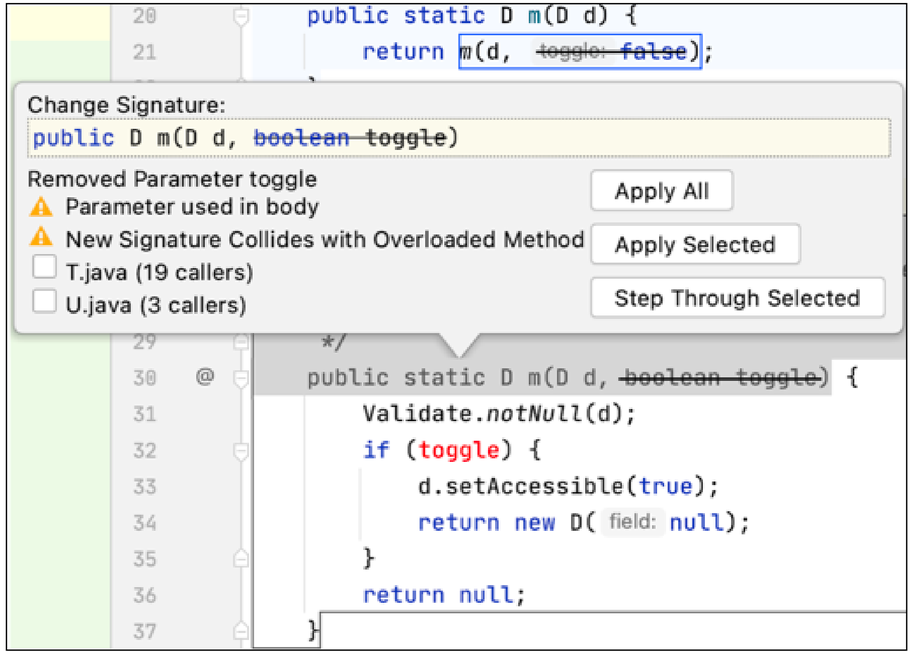


Usability problems

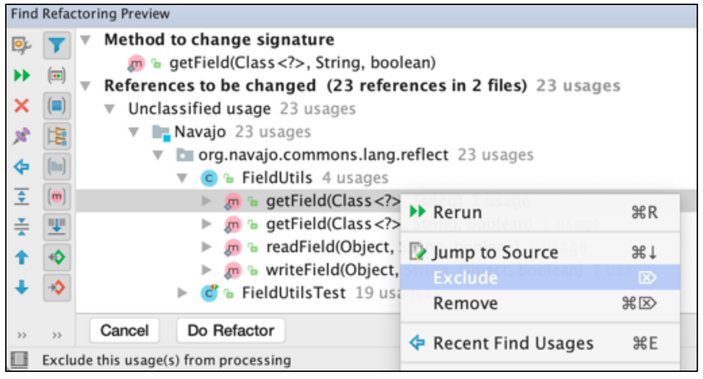
Configuration View



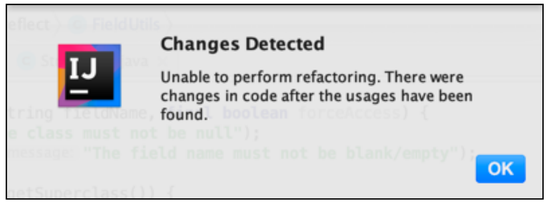
Impacts and additional changes



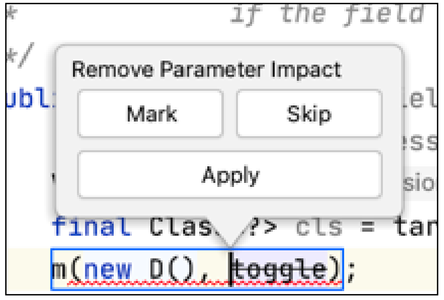
Preview View



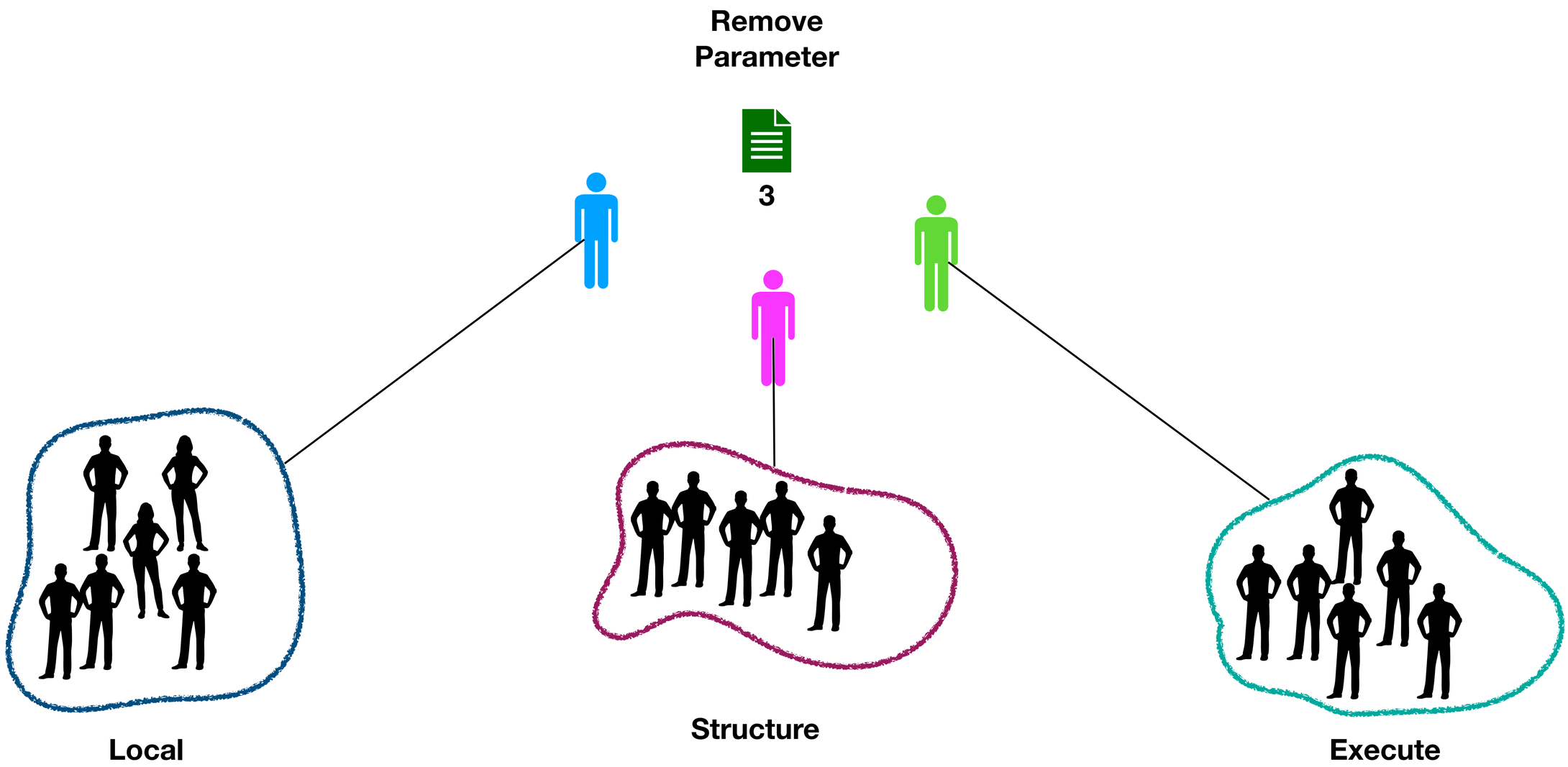
Can not change code



Ability to step through

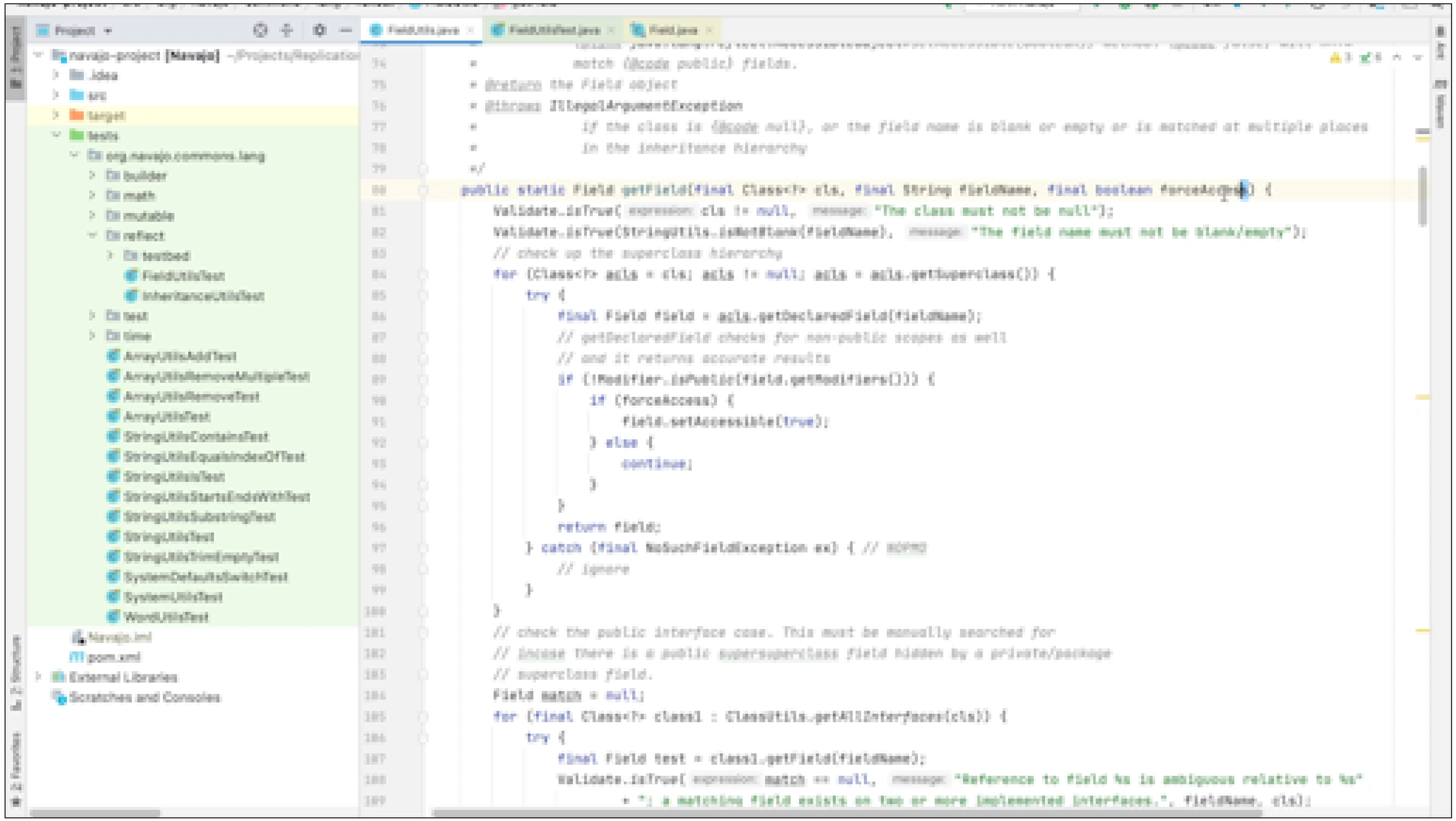


Conceptual validation



Paper IV

Stepwise refactoring tool



Summary of contribution

- An interaction model for a stepwise refactoring tool, and a prototype of such a tool, that showcases ways to address multiple of the usability issues that were previously identified

Future Work

- Is a stepwise refactoring tool more usable?
- How can a stepwise refactoring tool be integrated with other tool types, like "quick-fixes"?
- Do developers exibit these three strategies in other software change tasks?
- Which other usability factors should be considered by toolmakers?

Summary of contributions

- Three strategies: Local, Structure, and Execute
- A theory of usability for refactoring tools
- Four usability themes
- Stepwise refactoring tool