Landing Page

THE UNIVERSITY OF BRITISH COLUMBIA
Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1Z4

Tool Use in Software Change Tasks

Do you wish it was easier to change software? So do we. And we want to make that happen.

First, we need to know how developers use the tools that already exist. The purpose of this survey is to understand developers' experiences with mainstream tools. We focus on development tools for Java and C# programs, because they are popular, mainstream, and allows us to build on others' work.

The survey is expected to take no more than 40 minutes to complete. You participate by answering questions in an anonymous, web-based questionnaire that you can start, pause, and complete at your convenience. Your responses are anonymous: we do not collect identifying information.

If you choose to participate, you will be asked to share your experiences with a few mainstream tools. You will also be asked to answer questions about tool use in the context of a few software change scenarios that have occurred in open-source codebases.

In addition to being a contributor to our research goal, you may find it interesting to learn about alternative ways to solve real software change problems and to reflect on your own tool usage.

This study is conducted by Dr. Gail Murphy, Dr. Anya Bagge, and graduate student Anna Eilertsen. It is funded by the Research Council of Norway under grant number 250683 (Co-Evo).

For more information, continue to the consent form on the next page.

Informed Consent

THE UNIVERSITY OF BRITISH COLUMBIA
Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1Z4

Tool Use in Software Change Tasks

Welcome to our study on tool use in software change tasks. This first page provides survey information and asks for your consent to participate. If you consent, you will be taken to a set of questions that ensure eligibility to participate. If you are eligible, you will be taken to the survey.

Principal Investigator:
Dr. Gail Murphy, Dept. of Computer Science, University of British Columbia (murphy@cs.ubc.ca, +1 604 822 5169)

Co-Investigators:
Anna Maria Eilertsen, graduate student, Institute for Informatics, University of Bergen (anna.eilertsen@uib.no, +47 401 03 368)

Other Investigators:
Dr. Anya Bagge, Institute for Informatics, University of Bergen (anya.bagge@uib.no, +47 482 71 775)

Study Purpose:
The purpose of this survey is to learn about how software developers experience tools that can help them make software changes. We want to gather data about the different tools that developers use when changing software and how they fit into developers' workflows. The data resulting from this study will be used to design tools that better support software evolution and maintenance activities.

What you will be asked to do:

This is a web-based survey. You will be asked to read and respond to a series of questions regarding your experience with a number of tools that developers use when changing software. The survey is estimated to take 40 minutes or less. You will be asked to read descriptions of software change scenarios and answer questions about how the steps and tools that you would approach them with. You will be asked qualifying questions to ensure eligibility. You will be asked non-identifying demographic questions.

Known Risks:
The main risk is the time required to answer the survey. This amounts to the time it takes you to respond to the questions in this web-based survey. We are mitigating this risk by distributing the survey such that you may respond at a time that is convenient for you. Also, you can terminate your participation in the survey at any point in time without providing any reason. Otherwise, the risks involved in this study are minimal and are those commonly associated with the use of computers, such as potential eye or wrist strain.

Potential benefits:
You may find it beneficial to learn about different tools for software changes. You may also find it interesting to reflect on your own tool use. Direct benefits can arise if our findings lead to automated tools that can better support you in your daily software evolution and maintenance tasks.

Compensation:
You will not be compensated for participating in this survey.

Data, Storage & Confidentiality:

Survey responses will be stored on password-protected and encrypted devices. No personal information will be collected and as a result all data is anonymous. This means that once you complete and submit your survey responses, we can no longer remove data if you choose to withdraw your consent.

You will be identified by number or pseudonyms in any internal or academic research publication or presentation. If we choose to use some of your comments, they will be attributed to a participant number or pseudonym. At no point in time will your employer have access to the identifying information.

The anonymous data may be seen by other researchers for educational purposes or the application of further scientific methods. Papers published on this data may also require the processed data to be submitted to an online repository or database where other researchers and members of the public will be able to access it. Note that you will not identifiable in this data.

The survey responses will be stored for at least five years and may exist longer, in open repositories, to enable other researchers to benefit from the data. All of the data collected and stored is anonymous.

Use of the Data:

The results of this study will potentially appear in both internal and external academic research presentations and publications, such as academic journals and conference proceedings. Note that you will not identifiable in this data.

The data collected in this study may be useful for designing better tools for software developers or benchmarking tools. We may wish to use the collected data in the future for this purpose. Please note that this data does not require any information that can be traced back to you or your personal data.

Contact for information about the study:

If you have any questions or desire further information with respect to the study, you may contact Dr. Gail Murphy (murphy@cs.ubc.ca, +1 604-822-5169).

Who can you contact if you have complaints or concerns about the study?

If you have any concerns or complaints about your rights as a research participant and/or your experiences while participating in this study, contact the Research Participant Complaint Line in the UBC Office of Research Ethics at 604-822-8598 or in long distance email RSIL@ors.ubc.ca or call toll free 1-877-822-8598.

Study Ethics ID H20-03787

Consent

Your participation in this survey is entirely voluntary. You are free to withdraw your participation at any point without providing any reason. Any information you contribute up to your withdrawal will be retained and included in the dataset unless you request otherwise.

By clicking "I consent" below and providing your name and the date, you confirm that you:

1. *understand what is required based on reading the information provided above,*
2. *understand that your participation is voluntary and you are free to withdraw at any time,*
3. *understand the provisions of confidentiality,*

4. *consent to participate in the study.*

○ I consent and wish to participate
○ I do not consent, I do not wish to participate

DeclinedBlock

You declined the consent form.

You are not eligible to participate in this survey. In order to ensure that the collected information is relevant and up to date, we require participants to have two years of professional experience with Java or C# programming from within the last ten years.

Thank you for your consideration. Click the "next" button to end this survey.

Background Questions

Background Questions

On this page, we present you with the terminology we use in this survey. Please read it carefully.

> **Software change tasks** refer to activities where developers apply changes to existing software for purposes such as adding or refining a feature, correcting a mistake, or improving code quality.
>
> When applying changes, developers may also need to take actions like inspecting or validating changes they have already made and actions that help them understand the code, such as navigating or reading source code, test code, and documentation.
>
> We use the term **tools** to refer to programs that help support or automate the types of software change activities previously outlined: applying changes, inspecting changes, validating changes, and comprehending source code, test code, and documentation.
>
> Developers typically perform one or more of these activities in an integrated development environment (IDE) or a text editor. We refer to these collectively as **programming environments**. Tools may be accessed both from the IDE or editor or from other sources such as, e.g., a console window.

Now we will ask a few background questions. Your responses will help us ensure that your experiences are within the scope of this study and will be kept confidential.

How many years have you worked in the software industry?

0 4 8 12 16 20 24 28 32 36 40

The following questions will refer to "changing software in Java or C#". You may answer every question based on the language you have used the most or are most proficient with. For example, if you have spent 1 year maintaining software in Java and 10 other years in C#, answer 11 years. If you spent 1 year maintaining both Java and C# software, answer 1 year.

How many years have you spent developing or maintaining software in Java or C# professionally?

0 4 8 12 16 20 24 28 32 36 40

How many of **the last ten years** (2010-2020) have you spent developing or maintaining software in Java or C# professionally?

0 1 2 3 4 5 6 7 8 9 10

For questions relating to programming environments or tools, you may answer based on both your Java and C# experience. For example, if you use the environment *foo* for Java code and *bar* for C# code, you may select both *foo* and *bar*.

Which of the following programming environments do you use when working on software in Java or C#?

□ JetBrains IntelliJ

□ Eclipse
□ Netbeans
□ Visual Studio
□ Visual Studio Code
□ JetBrains Rider
□ Vim
□ EMACS
□ Other, please specify

Please rank your general proficiency with changing software in Java and C#.
(One star indicates no proficiency and five stars indicate expert proficiency.)

Java

C#

Demographic Questions

Demographic Questions

Based on your answers to the background questions, we welcome your participation in this study, **Tool Use in Software Change Tasks**.

On this page, we ask a few demographic questions. Your answers will allow us to compare data across different groups of respondents and will be kept confidential.

Which job title best represents your responsibilities right now?

○ Programmer / Software Developer / Software Engineer
○ System Administrator / Network Engineer
○ Project Manager
○ Technical Lead / Team Leader
○ Researcher
○ Other, please specify
○ Prefer not to answer

What is the highest level of school you have completed or the highest degree you have received?

○ Less than high school degree
○ High school graduate (high school diploma or equivalent including GED)
○ Some college or university but no degree
○ Bachelor's degree
○ Master's degree
○ Doctoral degree
○ Professional degree (JD, MD)
○ Prefer not to answer

What is your age?

○ Under 18
○ 18 - 24
○ 25 - 34
○ 35 - 44
○ 45 - 54
○ 55 - 64
○ 65 - 74
○ 75 - 84

○ 85 or older
○ Prefer not to answer

What is your gender?

○ Male
○ Female
○ Other
○ Prefer not to answer

Part 1 - Tool Experience

Tool Experiences

We will now ask about your experiences with using tools when changing software written in Java and C#.

We briefly reiterate the survey terminology.

> **Software change tasks** refer to applying changes to existing software for purposes such as adding or refining a feature, correcting a mistake, or improving code quality.
>
> We use the term **tools** to refer to programs that help support or automate the types of software change activities previously outlined: applying changes, inspecting changes, validating changes, and comprehending source code, test code, and documentation.

Please answer the following questions based on your experiences within the last ten years of changing software written in Java or C#.

The following is a list of tools that software developers may find useful during software change tasks. Please mark all the tools that you find useful during software change tasks.

□ Complex Refactoring tools (e.g. Move Method, Extract Class, Introduce Parameter)
□ Textual search (e.g. grep, find and replace)
□ Debugging tools
□ Compiler output (e.g. compiler errors)
□ Structural navigation (e.g. find references, go to declaration)
□ Version Control Systems "diffs" (e.g. git diff, et.c.)
□ Simple Refactoring tools (e.g. Rename, Extract Constant, Inline Method)
□ Test suites (e.g. JUnit, test errors)
□ Copy/cut and paste code

Part 1 - Tool Experience - ranking

Here we list some statements that may or may not be true about the different tools.

For each tool, please mark **the statements that are necessary for that tool to be useful**.

You should select statements that are necessary, regardless of whether they are currently true or not. The following list contains a description of each statement.

- The tool is effective if it is reasonable to locate and apply for a desired intent.
- The tool saves time or effort if it is reasonable in terms of time and effort required to use.
- The tool is satisfying to use if it adds value to the development process.
- The tool is trustworthy if it feels safe or reliable to use, for example by rarely or never making errors or mistakes.
- The tool is predictable if it behaves consistently and predictably each time you use it.

	The tool is effective	The tool saves time or effort	The tool is satisfying to use	The tool is trustworthy	The tool is predictable
» Simple Refactoring tools (e.g. Rename, Extract Constant, Inline Method)	□	□	□	□	□
» Complex Refactoring tools (e.g. Move Method, Extract Class, Introduce Parameter)	□	□	□	□	□

	The tool is effective	The tool saves time or effort	The tool is satisfying to use	The tool is trustworthy	The tool is predictable
» Version Control Systems "diffs" (e.g. git diff, et.c.)	□	□	□	□	□
» Compiler output (e.g. compiler errors)	□	□	□	□	□
» Textual search (e.g. grep, find and replace)	□	□	□	□	□
» Structural navigation (e.g. find references, go to declaration)	□	□	□	□	□
» Copy/cut and paste code	□	□	□	□	□
» Test suites (e.g. JUnit, test errors)	□	□	□	□	□
» Debugging tools	□	□	□	□	□

Part 2 - Scenario - intro

You will now be presented with three **software change scenarios**.

Each scenario follows the following format: you will be given a description of a **task** you need to solve, three **approaches** you may want to use, and a question about **tool use** in this task.

For each scenario, please select the approach that is most similar to the approach you are likely yo take. If none of the approaches apply to you, select None of the above, and give a textual brief description of what you would do.

For the purpose of these three scenarios, put yourself in the following position.

You are acting as one of three developers of a utility library for Java core types. The source code is a Maven project comprising 78K lines of Java code and 1335 JUnit tests. It mainly consists of classes with static utility methods.

All public methods have one or more tests that specify their behavior. When considering the tasks you should consider changes to source code and test code but you may assume that appropriate release notes or other client-related actions has been handled.

For example, if your task is to remove a method **foo** from the library and replace it with a method **bar** that has somewhat different behavior, you need to remove **foo**, add **bar**, and update any internal references to foo, including JUnit tests like **testFoo**. You do NOT need to deprecate **foo**, consider release documents, client code, or other artifacts as long as other code in the library functions as before. You will be given descriptions about internal usages as test code as part of the task or approach descriptions.

If you wish to look at the source code or even attempt any of the tasks yourself, you may browse or download it from a GitHub repository here. It will open in a new window. **It is not necessary to do so in order to solve the tasks.**

Part 2 - Scenario - Remove Methods

Remove methods scenario

In this task, you should remove a few methods that are no longer useful. These methods are declared in a file of 8000 lines consisting solely of static methods and test methods are declared in its own (JUnit) test file.

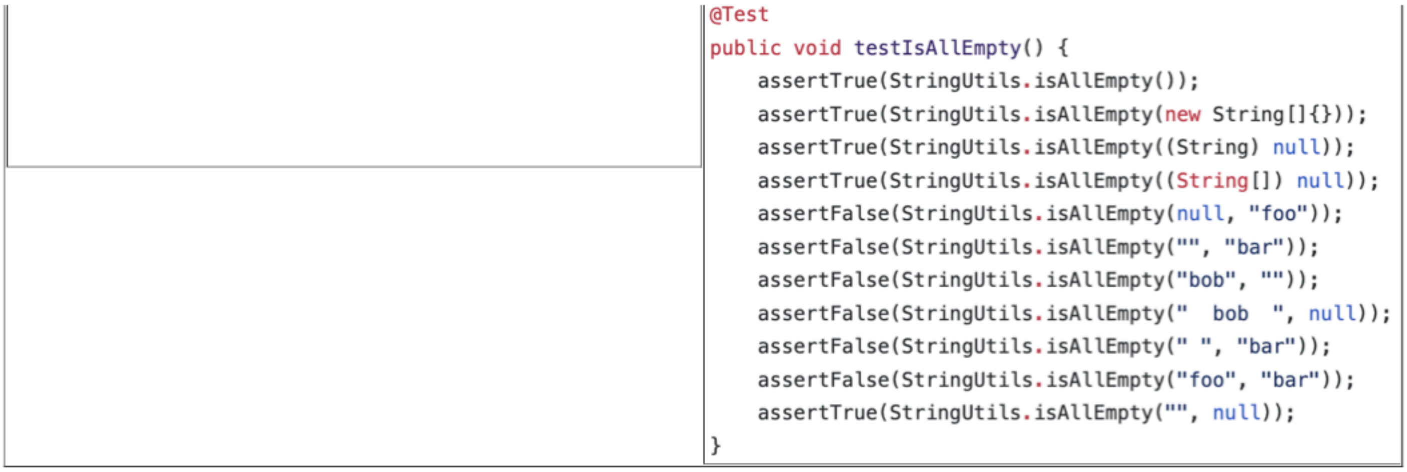
Each method has one in-class caller and one test method. They all follow this pattern. Here we present an excerpt of the functional code to be changed (left) and the tests to be changed (right). In this example, isAnyNotEmpty should be removed.

Functional Code: StringUtils.java

```java
public static boolean isAnyNotEmpty(final CharSequence... css) {
        if (ArrayUtils.isEmpty(css)) {
                return false;
        }
        for (final CharSequence cs : css) {
                if (isNotEmpty(cs)) {
                        return true;
                }
        }
        return false;
}
...
public static boolean isAllEmpty(final CharSequence... css) {
        return !isAnyNotEmpty(css);
}
```

Test Code: StringUtilsTest.java

```java
@Test
public void testIsAnyNotEmpty() {
    assertFalse(StringUtils.isAnyNotEmpty((String) null));
    assertFalse(StringUtils.isAnyNotEmpty((String[]) null));
    assertTrue(StringUtils.isAnyNotEmpty(null, "foo"));
    assertTrue(StringUtils.isAnyNotEmpty("", "bar"));
    assertTrue(StringUtils.isAnyNotEmpty("bob", ""));
    assertTrue(StringUtils.isAnyNotEmpty("  bob  ", null));
    assertTrue(StringUtils.isAnyNotEmpty(" ", "bar"));
    assertTrue(StringUtils.isAnyNotEmpty("foo", "bar"));
    assertFalse(StringUtils.isAnyNotEmpty("", null));
}
```



```
@Test
public void testIsAllEmpty() {
    assertTrue(StringUtils.isAllEmpty());
    assertTrue(StringUtils.isAllEmpty(new String[]{}));
    assertTrue(StringUtils.isAllEmpty((String) null));
    assertTrue(StringUtils.isAllEmpty((String[]) null));
    assertFalse(StringUtils.isAllEmpty(null, "foo"));
    assertFalse(StringUtils.isAllEmpty("", "bar"));
    assertFalse(StringUtils.isAllEmpty("bob", ""));
    assertFalse(StringUtils.isAllEmpty("  bob  ", null));
    assertFalse(StringUtils.isAllEmpty(" ", "bar"));
    assertFalse(StringUtils.isAllEmpty("foo", "bar"));
    assertTrue(StringUtils.isAllEmpty("", null));
}
```

If you wish to see all the code in detail, click anywhere on the code or here to open it in a new window as a GitHub repository.

In this example, isAnyNotEmpty should be removed and isAllEmpty should stay. Then, you need to perform the same change to other methods.

Please select the workflow **is most similar to what you would do** in your day-to-day work to approach this scenario.

○ **I would:**

1. Start by removing isAnyNotEmpty (e.g. by deleting or commenting out) to make compiler errors appear.
2. Use the compiler errors to navigate to the code that is impacted (i.e., testIsAnyNotEmpy and isAllEmpty).
3. Fix compiler errors (e.g. by removing testIsAnyNotEmpty and implementing isAllEmpty by undoing and moving code).
4. Validate the changes by running tests.

○ **I would:**

1. Start by moving the implementation of isAnyNotEmpty into isAllEmpty.
2. Validate this change by running testIsAllEmpty.
3. If the test pass, I will remove testIsAnyNotEmpty and isAnyNotEmpty (e.g. by deleting or commenting out).
4. Finally, I validate the changes by running tests.

○ **I would:**

1. I would start by introducing a temporary new method that isAnyNotEmpty delegate to.
2. I would validate this change by running tests.
3. Then I would replace the call to isAnyNotEmpty to the temporary method. (e.g. return !isAnyNotEmpty(..) is replaced by return !tempMethod(..))
4. I would validate this change by running tests.
5. If the tests pass, I would remove testIsAnyNotEmpty and isAnyNotEmpty (e.g. by deleting or commenting out)
6. Then I would validate the change by running tests.
7. If the tests pass, I would inline the temporary method into isAllEmpty and remove it.

○ **None of the above**, please specify

You chose ${q://QID84/ChoiceGroup/SelectedChoices}

You chose ${q://QID95/ChoiceTextEntryValue/10}

Which of the following tools would you use in this scenario?

□ Copy/cut and paste code

□ Textual search (e.g. grep, find and replace)

□ Compiler output (e.g. compiler errors)

□ Test suite (e.g. test failures)

□ Move Method

□ Extract Constant

□ Extract Method

□ Rename

□ Remove Parameter

□ Version Control Systems "diffs" (e.g. git diff, et.c.)

□ Inline Constant

□ Extract Class

□ Change Signature

□ Inline Method

□ Structural navigation (e.g. find references, go to declaration)

□ Safe Delete

□ Other, please specify

Please select all the the statements that best match your reason for choosing this approach.

If none apply to you, please specify your reason.

□ This approach lets me keep the tests running so I can validate changes.

□ This approach lets me make the change stepwise.

□ This approach lets me rely on the compiler to show me the steps.

□ This approach lets me automate as much as possible.

□ None of the above, please specify..

One way to solve this task would be to use the refactoring tool Inline Method to move the implementation of isAnyNotEmpty to isAllEmpty.

You did not select the Inline Method refactoring. Why would you not use the Inline Method refactoring in this scenario?

□ The tool is not effective (it is not reasonable to locate and apply for a desired intent)

□ The tool does not saves time or effort (it is not reasonable in terms of time and effort required to use)

□ The tool is not satisfying to use (it does not add value to the development process)

□ The tool is not trustworthy (it does not feel safe or reliable to use)

□ The tool is not predictable (it does not behave consistently and predictably each time you use it)

□ Other, please specify...

Part 2 - Scenario - Reorganize Methods

Reorganize test methods scenario

There is a large class (StringUtils, ~8000 LOC) with static helper methods for strings (e.g. isAllEmpty, isAllBlank). Each method in this file has one or more JUnit tests (e.g. testIsAllEmpty, testIsAllBlank), distributed across two test files: StringUtilsTest and StringUtilStrimEmptyTest.

Your task is to reorganize the tests such that test methods related to a particular functionality (Empty and Blank) is found in its own class such that it is easier to locate and run tests related to only this functionality.

You need to take around 8 methods from one class (StringUtilsTest) and 4 methods from another test class (StringUtilStrimEmptyTest) and put them into a new test file (new file, e.g. StringUtilsEmptyBlankTest). If there is any setup, variables, et.c., they should be brought along.

Please select the workflow **is most similar to what you would do** in your day-to-day work to approach this scenario.

○ **I would:**

1. Start with creating a new empty class.
2. Then I would move all the methods into the new class.
3. I would fix any compiler errors (e.g. from any additional code).
4. Then I would validate the change.

○ **I would:**

1. Start with duplicating one of the classes (e.g. by copying and pasting).
2. Then I would validate this change.
3. Then I would remove the appropriate code from the original and the duplicate, so they each contain only the right code elements.
4. Then I would validate this change.
5. Once one class is finished, I would duplicate code from the other file into the new class (e.g. by copying and pasting).
6. Then I would validate this change.
7. Finally, I would remove code until all classes contain only the code I want and validate continously.

○ **I would:**

1. Start with extracting the code I want and any necessary additional code from one of the files into a new class.
2. Then I would extract the code I want and any necessary additional code from the other file into the new class.
3. Then I would validate the change.

○ **None of the above**, please specify

You chose ${q://QID95/ChoiceGroup/SelectedChoices}

You chose ${q://QID95/ChoiceTextEntryValue/10}

Which of the following tools would you use in this scenario?

□ Rename

□ Remove Parameter

□ Copy/cut and paste code

□ Extract Class

□ Extract Method

□ Inline Constant

□ Version Control Systems "diffs" (e.g. git diff, et.c.)

□ Inline Method

□ Compiler output (e.g. compiler errors)

□ Safe Delete

□ Test suite (e.g. test failures)

□ Textual search (e.g. grep, find and replace)

□ Extract Constant

□ Change Signature

□ Structural navigation (e.g. find references, go to declaration)

□ Move Method

□ Other, please specify

Please select all the the statements that best match your reason for choosing this approach. If none apply to you, please specify your reason.

□ This approach lets me keep the tests running so I can validate changes.

□ This approach lets me rely on the compiler to show me the steps.

□ This approach lets me automate as much as possible.

□ This approach lets me make the change stepwise.

□ None of the above, please specify..

One way to solve this task would be to use the refactoring tool Move Method to move methods from one class to another. You did not select the Move Method refactoring. Why would you not use the Move Method refactoring in this scenario?

□ The tool is not effective (it is not reasonable to locate and apply for a desired intent)

□ The tool does not saves time or effort (it is not reasonable in terms of time and effort required to use)

□ The tool is not satisfying to use (it does not add value to the development process)

□ The tool is not trustworthy (it does not feel safe or reliable to use)

□ The tool is not predictable (it does not behave consistently and predictably each time you use it)

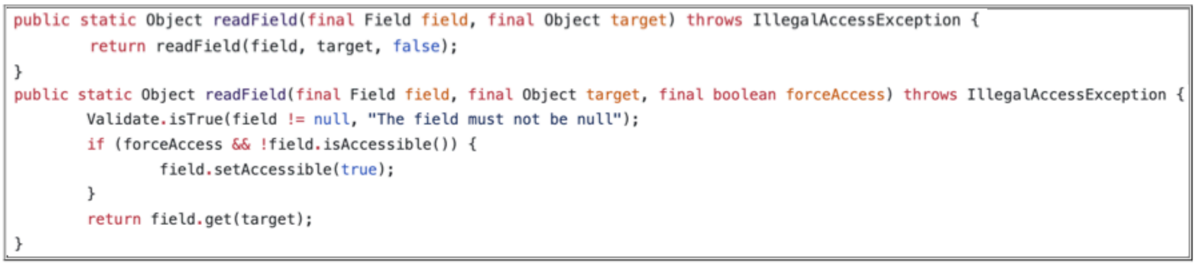
□ Other, please specify...

Part 2 - Scenario - Parameter Removal

Parameter Removal Scenario

In this task, you need to make changes to a class FieldUtils, comprising 16 static helper methods for reading and writing fields, and to their test methods. Each method in the class has 1 or more JUnit tests. All methods in the class are only called by other methods in the class or by the test code.

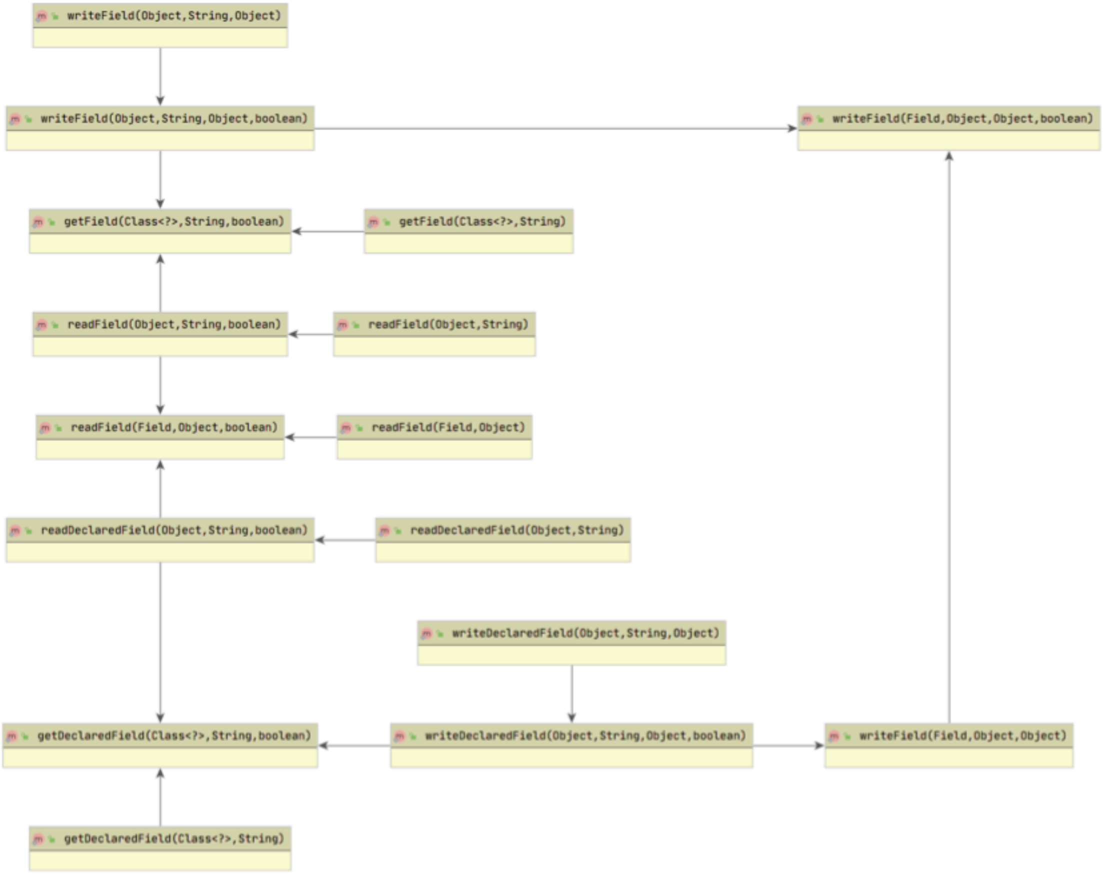
The class consists of 8 method "pairs". Where the method without the forceAccess parameter, or flag exposes a default behavior, and the method with the parameter allows to access a special-case behavior. The default method delegates to the special-case method by calling it with the parameter set to false, as can be seen below.



```
public static Object readField(final Field field, final Object target) throws IllegalAccessException {
        return readField(field, target, false);
}
public static Object readField(final Field field, final Object target, final boolean forceAccess) throws IllegalAccessException {
        Validate.isTrue(field != null, "The field must not be null");
        if (forceAccess && !field.isAccessible()) {
                field.setAccessible(true);
        }
        return field.get(target);
}
```

Your task is to remove the parameter. In order to do so, **all methods should be declared without the forceAccess parameter. Consequently, in-class callers must be updated, as must the tests**. (There are no callers outside the class).

In addition to the delegating calls between method "pairs", there are dependencies between the methods in the class, as can be seen in the following diagram.


writeField(Object,String,Object)
writeField(Object,String,Object,boolean)
writeField(Field,Object,Object,boolean)
getField(Class<?>,String,boolean)
getField(Class<?>,String)
readField(Object,String,boolean)
readField(Object,String)
readField(Field,Object,boolean)
readField(Field,Object)
readDeclaredField(Object,String,boolean)
readDeclaredField(Object,String)
writeDeclaredField(Object,String,Object)
getDeclaredField(Class<?>,String,boolean)
writeDeclaredField(Object,String,Object,boolean)
writeField(Field,Object,Object)
getDeclaredField(Class<?>,String)

The ability to force access should be removed. Consequently, the forceAccess parameter representing this flag should be removed and all callers and overloaded methods should be updated accordingly. During this process all 16 methods must be changed at least once

Please select the workflow **is most similar to what you would do** in your day-to-day work to approach this scenario.

○ **I would:**

1. Go through the file top-down and move the implementation from each forceAccess-method into the default method (i.e. for all 8 methods before addressing compiler errors).
2. Go through the file top-down again, and remove all usages of forceAccess in the method body (e.g. by locating compiler errors)
3. Then I would update all callers and tests (e.g. by relying on compiler errors)
4. Finally, I would validate the change by running the tests.

○ **I would:**

1. Start by assigning the default value of false to all forceAccess parameters inside all the special-case methods, so the class only performs the default functionality.
2. Then I would run the tests to locate the ones that break and fix them (e.g. by deleting the ones that are no longer necessary).
3. Once all tests pass, I would remove the forceAccess functionality (that is now unused) from all 8 methods.
4. Then I validate the change using tests.
5. Finally, I move all implementations into the default methods' bodies.
6. Then I validate the change using tests.

○ **I would:**

1. Start by exploring the code structure and get an overview of callers and parameter references.
2. Change one method and its callers at a time, attempting to resolve any compiler errors that appear, (e.g. by removing tests, or delete the default method or update code that references forceAccess, or consolidate the overloaded methods) before moving to the next method.
3. Repeat for each method until all 8 methods are updated.
4. Validate the change by running tests.

○ **None of the above**, please specify

You chose ${q://QID108/ChoiceGroup/SelectedChoices}

You chose ${q://QID108/ChoiceTextEntryValue/10}

Which of the following tools would you use in this scenario?

□ Copy/cut and paste code
□ Inline Constant
□ Compiler output (e.g. compiler errors)
□ Test suite (e.g. test failures)
□ Inline Method
□ Rename
□ Change Signature
□ Safe Delete
□ Textual search (e.g. grep, find and replace)
□ Move Method
□ Version Control Systems "diffs" (e.g. git diff, et.c.)
□ Extract Class
□ Extract Constant
□ Structural navigation (e.g. find references, go to declaration)
□ Extract Method
□ Remove Parameter
□ Other, please specify

Please select all the the statements that best match your reason for choosing this approach. If none apply to you, please specify your reason.

□ This approach lets me automate as much as possible.
□ This approach lets me make the change stepwise.
□ This approach lets me keep the tests running so I can validate changes.
□ This approach lets me rely on the compiler to show me the steps.
□ None of the above, please specify..

One way to solve this task would be to use the refactoring tool Change Signature to remove the forceAccess parameter. You did not select the Change Signature refactoring. Why would you not use the Change Signature refactoring in this scenario?

□ The tool is not effective (it is not reasonable to locate and apply for a desired intent)
□ The tool does not saves time or effort (it is not reasonable in terms of time and effort required to use)
□ The tool is not satisfying to use (it does not add value to the development process)
□ The tool is not trustworthy (it does not feel safe or reliable to use)

□ The tool is not predictable (it does not behave consistently and predictably each time you use it)

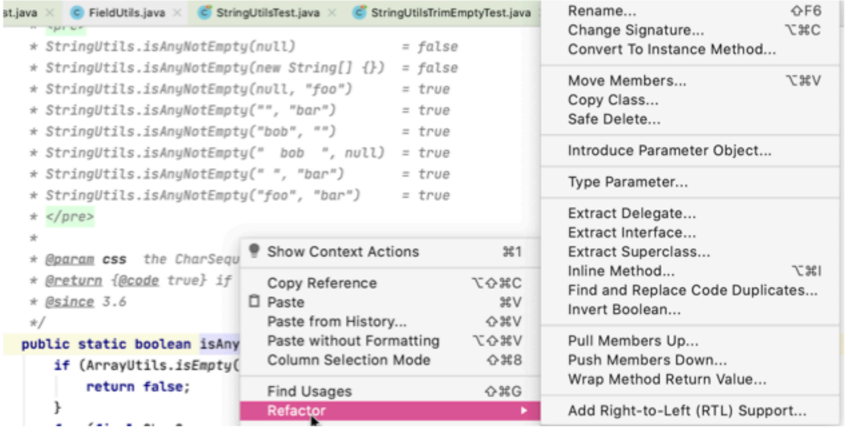
□ Other, please specify...

Refactoring Usability Issues

Refactoring Tool Usability

IDEs usually support numerous refactoring operations. On this page, we are interested in any problems you may have encountered when attempting to use such refactoring tools. We first provide a brief description of how they are invoked, before posing some problem statements that may or may not apply to your experience.

A refactoring operation can be accessed by selecting a *program element* (like a method, a variable, or a code selection) and invoking the refactoring either from the menu bar, by right-clicking on code, or using *hotkeys* (e.g. cmd+alt+R to Rename in Eclipse).



```
 * StringUtils.isAnyNotEmpty(null)              = false
 * StringUtils.isAnyNotEmpty(new String[] {})   = false
 * StringUtils.isAnyNotEmpty(null, "foo")       = true
 * StringUtils.isAnyNotEmpty("", "bar")         = true
 * StringUtils.isAnyNotEmpty("bob", "")         = true
 * StringUtils.isAnyNotEmpty("  bob  ", null)   = true
 * StringUtils.isAnyNotEmpty(" ", "bar")        = true
 * StringUtils.isAnyNotEmpty("foo", "bar")      = true
 * </pre>
 *
 * @param css  the CharSequ
 * @return {@code true} if
 * @since 3.6
 */
public static boolean isAny
    if (ArrayUtils.isEmpty(
        return false;
    }
```

Once a refactoring operation is invoked on the program element, most IDEs will open an interactive wizard that lets you provide additional arguments (e.g. the name of an extracted method), *preview* the change (e.g. see all locations that will be changed), and a *problem* view that presents errors and warnings, and finally, a "*Continue*" or "Do Refactor" button that applies the refactoring operation to the source code.

With this workflow in mind, please consider the following statements describing usability problems that can occur during these steps. Please check any statements that describe situations you have encountered during your work.

□ **The tool makes me lose control of my code or workflow.** This may be due to, for example, changing code you did not intend, making larger changes than you intended, hiding information that you will need later, etc.

□ **The tool prevents me from validating the change.** This may be due to, for example, changing too many things at the same time, changing test code and source code simultaneously, not presenting a way to review changes, etc.

□ **The tool does not help me find the next step.** This may be due to, for example, using terminology that you do not understand, not offering alternative steps when presenting problems or options, not communicating the implications of choices you make, etc.

□ Prefer not to answer

If you use automated refactoring tools in any of the following programming environments, please rank your proficiency with the refactoring tools that are available in that programming environment.
(One star indicates no proficiency and five stars indicate expert proficiency.)

» JetBrains IntelliJ

» Eclipse

» Netbeans

» Visual Studio

» Visual Studio Code

» JetBrains Rider

» Vim

» EMACS

» Other, please specify

When you perform refactorings using an **automated tool**, which of the following tools do you use to **verify** that the change is correct? (Select all that applies. If you do nothing to verify, select "I do not verify the change")

□ Version Control Systems "diffs" (e.g. git diff, et.c.)
□ Compiler output (e.g. compiler errors)
□ Textual search (e.g. grep, find and replace)
□ Structural navigation (e.g. find references, go to declaration)
□ Copy/cut and paste code
□ Test suites (e.g. JUnit, test errors)
□ Debugging tools
□ Other, please specify..
□ I do not verify the change

Please indicate the percentage of refactoring operations like the ones referred to in this study that you perform manually (e.g. using cut-and-paste or other simple tools) instead of using refactoring tools that are available to you?

	0	10	20	30	40	50	60	70	80	90	100
The percentage of % of refactoring operations performed manually instead of using available tools.											

When you choose to perform refactorings **manually** instead of using refactoring tools that are available to you, which of the following reasons impacts that choice?

□ The tool is not effective (it is not reasonable to locate and apply for a desired intent)
□ The tool does not saves time or effort (it is not reasonable in terms of time and effort required to use)
□ The tool is not satisfying to use (it does not add value to the development process)
□ The tool is not trustworthy (it does not feel safe or reliable to use)
□ The tool is not predictable (it does not behave consistently and predictably each time you use it)
□ Other, please specify...

When you perform refactorings **manually**, which of the following tools do you use to **verify** that the change is correct? (Select all that applies. If you do nothing to verify, select "I do not verify the change")

□ Version Control Systems "diffs" (e.g. git diff, et.c.)
□ Compiler output (e.g. compiler errors)
□ Textual search (e.g. grep, find and replace)
□ Structural navigation (e.g. find references, go to declaration)
□ Copy/cut and paste code
□ Test suites (e.g. JUnit, test errors)
□ Debugging tools
□ Other, please specify..
□ I do not verify the change

When you were asked which statements was necessary for **Simple Refactoring tools (e.g. Rename, Extract Constant, Inline Method)** to be useful, these are the statements you selected.

Now, please select the statements **that you agree are true**. If you think none of these statements are true, select None.

None

	None
Simple Refactoring tools (e.g. Rename, Extract Constant, Inline Method)	□

When you were asked which statements was necessary for **Complex Refactoring tools (e.g. Move Method, Extract Class, Introduce Parameter)** to be useful, these are the statements you selected.

Now, please select the statements **that you agree are true**. If you think none of these statements are true, select None.

	None
Complex Refactoring tools (e.g. Move Method, Extract Class, Introduce Parameter)	□

If you have encountered any difficult situations when attempting to use refactoring tools in software change tasks, other than the ones we have asked about so far, please give a brief description here.

Prototype block

Here we present three types of functionality that could be added to refactoring tools.

For each type, please read the description and indicate whether it would make you more likely to use refactoring tools, it would have no change, or it would make you less likely to use tools.

When you apply the refactoring, the tool offers the ability to **step through** each change in the code editor as it happens, so that in each step you may view or edit the code in that location before continuing, somewhat akin to stepping through a program execution with a debugger tool.

○ This would make me **more** likely to use a refactoring tool.

○ This would make me **no more or less** likely to use a refactoring tool.

○ This would make me **less** likely to use a refactoring tool.

When you apply the refactoring, the tool offers the ability to **review a summary** after changing the code so that you may learn about the code that was changed and see why it was changed, somewhat akin to reviewing a project's version control history.

○ This would make me **more** likely to use a refactoring tool.

○ This would make me **no more or less** likely to use a refactoring tool.

○ This would make me **less** likely to use a refactoring tool.

When you apply the refactoring, the tool offers the ability to **revert or alter a refactoring you previously applied** at a later point in your workflow, even after applying other code changes. This is so that you may change, for example, configuration arguments or the type of refactoring to reflect information you garnered since applying it, somewhat akin to rebasing a project's version control history.

○ This would make me **more** likely to use a refactoring tool.

○ This would make me **no more or less** likely to use a refactoring tool.

○ This would make me **less** likely to use a refactoring tool.

When presented with the ability to **step through** each change in the code editor as it happens, you answered: ${q://QID139/ChoiceGroup/SelectedChoices} Please briefly describe why.

When presented with the ability to **review a summary after changing the code**, you answered: ${q://QID140/ChoiceGroup/SelectedChoices} Please briefly describe why.

When presented with the ability to **revert or alter a refactoring you previously applied at** a later point in your workflow, you answered: ${q://QID142/ChoiceGroup/SelectedChoices} Please briefly describe why.

End Questions

The questions on this page are optional. Once you click next, the survey will end.

Use this field to provide us with any comments or extra information that you think is relevant.

Use this field to provide us with feedback on your experience with this survey.

Powered by Qualtrics