Program Transformation

CS 695 / SWE 699: Programming Tools

Fall 2023

GEORGE MASON UNIVERSITY

Today

- Part 1 (Lecture)(~60 mins)
- 10 min break!
- Part 2 (In-Class Activity)(50 mins)
- Part 3 (Group work time)

Logistics

- HW 2 due next week

Overview

- What is program transformation
- Applications of program transformation
  - Improving the design of code
  - Editable programming views
  - API & language migration
  - Security fixes

Program transformation

- Problem
    - You already have some existing code
    - You want to change it
    - Your change impacts several parts of the code

- --> Program Transformation

Key Ideas

- Transformations are repetitive edits
  - Need to change all call sites to add new parameter to signature
  - Need to change all call sites to respond to new API change
  - Need to change code snippets of a specific type to make them more secure
- Doing this work manually is tedious and error prone
  - Might miss something
- Programming tools can do some of this work for the developer

Some Key Design Dimensions

- What types of transformations are possible?
  - What types of changes can be described?
- How much control does a developer have over the transformation?
  - All or nothing? Change the behavior of a transformation that doesn't work right?
- What can a developer see before or after a change has been made?
  - # of place changed? Diff of each change? Errors where the change didn't work?

Challenges

- What if a transformation might sometimes insert a defect?

- What if there's more than one way to do the transformation, which a developer might want to choose between?

Improving the design of code

Copy & paste code reuse

- A very common way to edit code is by copying existing code. —> copy & paste reuse
- Creates code duplication
  - But… ok if this code duplication does not represent new abstraction

- Studies have attempted to understand when code duplication introduced by copy & paste is bad

- Many tools to detect code clones introduced by copy & paste

Slides for this section adapted from 05-899D Human Aspects of Software Development Spring 2011, “Software Evolution” by YoungSeok Yoon

Why do developers copy & paste code?

- structural template (the most common intention)
  - relocate, regroup, reorganize, restructure, refactor
- semantic template
  - design pattern
  - usage of a module (following a certain protocol)
  - reuse a definition of particular behavior
  - reuse control structure (nested if~else or loops)

M. Kim, L. Bergman, T. Lau, and D. Notkin (2004), “An ethnographic study of copy and paste programming practices in OOPL,” in *Proceedings of International Symposium on Empirical Software Engineering (ISESE’04)*, pp. 83-92.

Why do developers copy & paste?

- Forking
  - Hardware variations
  - Platform variation
  - Experimental variation
- Templating
  - Boiler-plating due to language in-expressiveness
  - API/Library protocols
  - General language or algorithmic idioms
- Customization
  - Bug workarounds
  - Replicate and specialize

C. Kapser and M. W. Godfrey (2006), "'Cloning Considered Harmful' Considered Harmful," in *13th Working Conference on Reverse Engineering (WCRE '06)*, 2006, pp. 19-28.

Properties of copy & paste reuse

- Unavoidable duplicates (e.g., lack of multiple inheritance)

- Programmers use their memory of C&P history to determine when to restructure code
  - delaying restructuring helps them discover the right level of abstraction

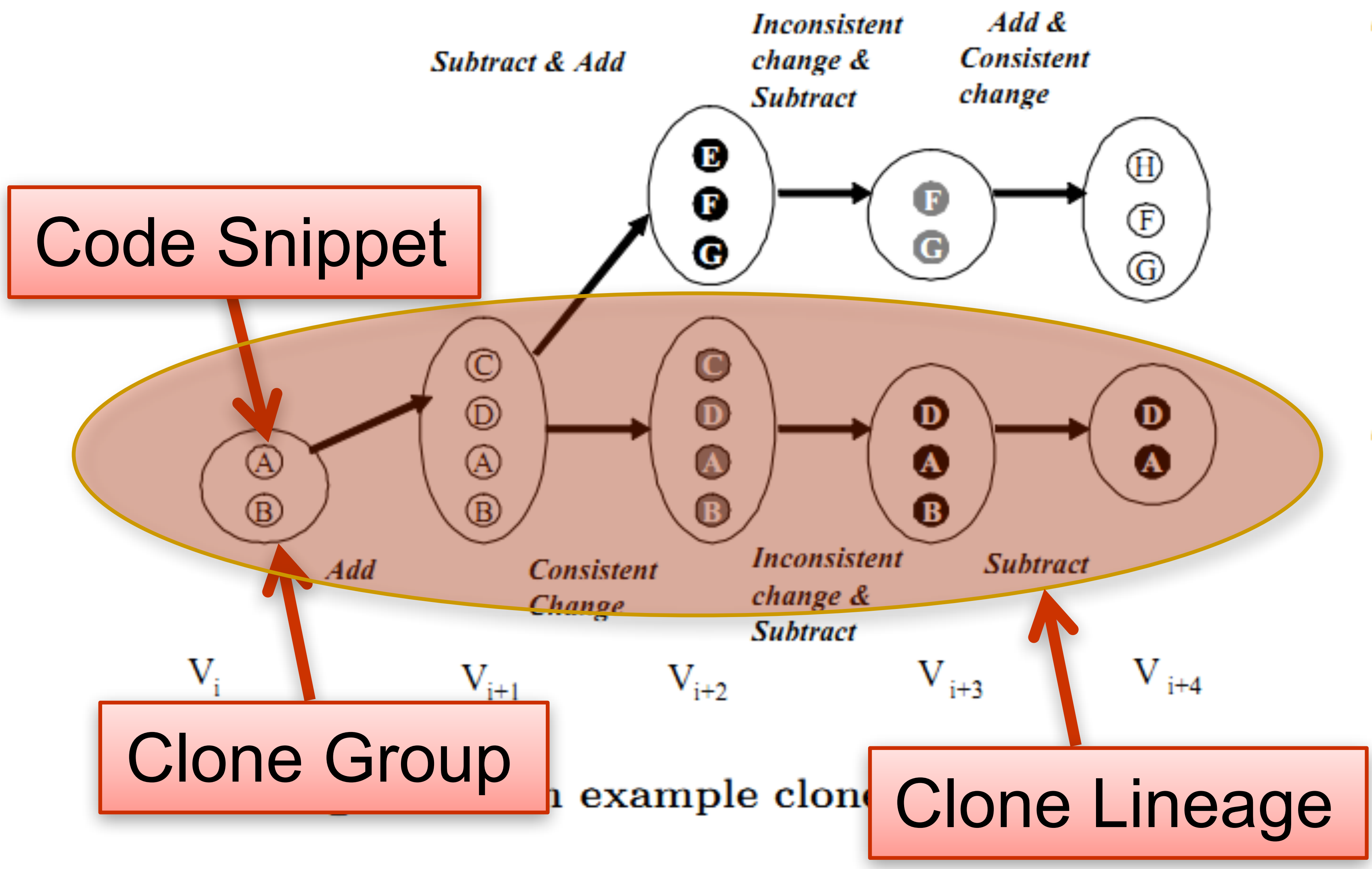
- C&P dependencies are worth observing and maintaining

M. Kim, L. Bergman, T. Lau, and D. Notkin (2004), “An ethnographic study of copy and paste programming practices in OOPL,” in *Proceedings of International Symposium on Empirical Software Engineering (ISESE’04)*, pp. 83-92.

Code clone genealogies

- Investigates the validity of the assumption that code clones are bad
- Defines clone evolution model

- Built an automatic tool to extract the history of code clones from a software repository



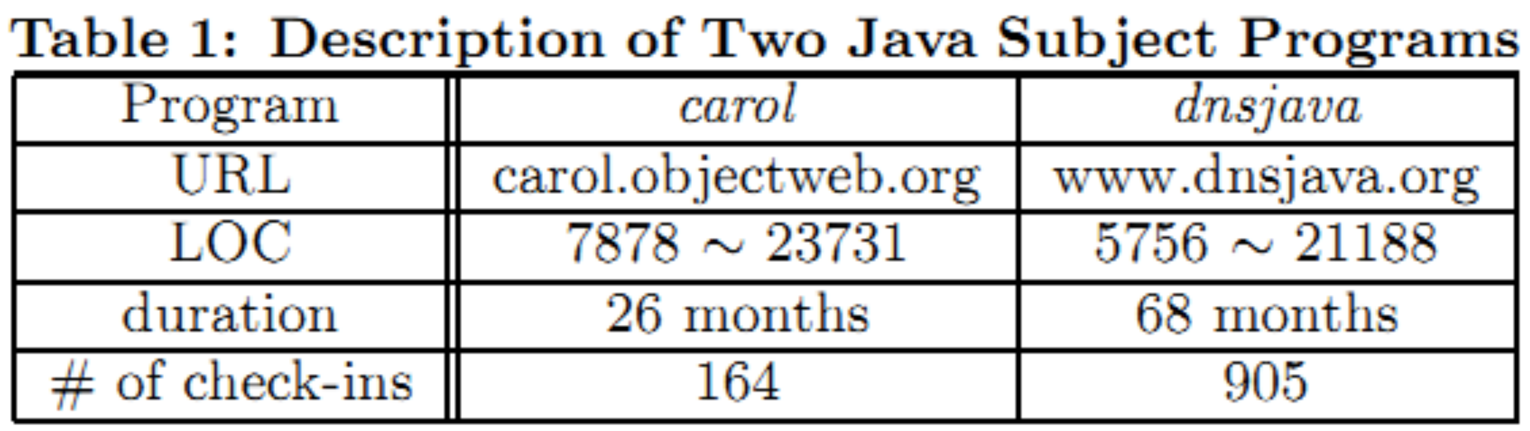


Table 1: Description of Two Java Subject Programs

Program	*carol*	*dnsjava*
URL	carol.objectweb.org	www.dnsjava.org
LOC	7878 ~ 23731	5756 ~ 21188
duration	26 months	68 months
# of check-ins	164	905

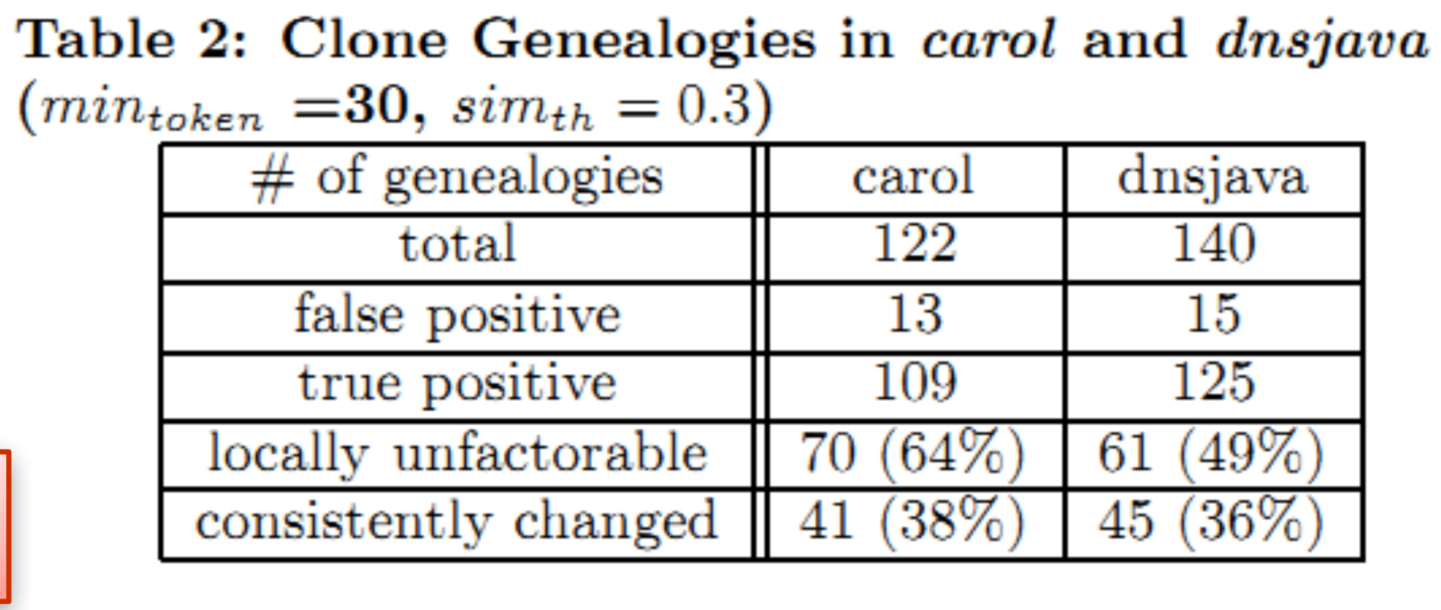


Table 2: Clone Genealogies in *carol* and *dnsjava* (min_{token} =30, sim_{th} = 0.3)

# of genealogies	carol	dnsjava
total	122	140
false positive	13	15
true positive	109	125
locally unfactorable	70 (64%)	61 (49%)
consistently changed	41 (38%)	45 (36%)

11

M. Kim, V. Sazawal, D. Notkin, and G. Murphy (2005), "An empirical study of code clone genealogies," in *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-13).*

Fixing code duplication

- Code duplication happens because there is a missing abstraction.
  - Instead of one piece of code being called 10 times with different parameters to achieve different behaviors, have 10 copies of code with behavior hardcoded

- How can we make it easier to redesign code to create the abstractions that we just realized we needed?

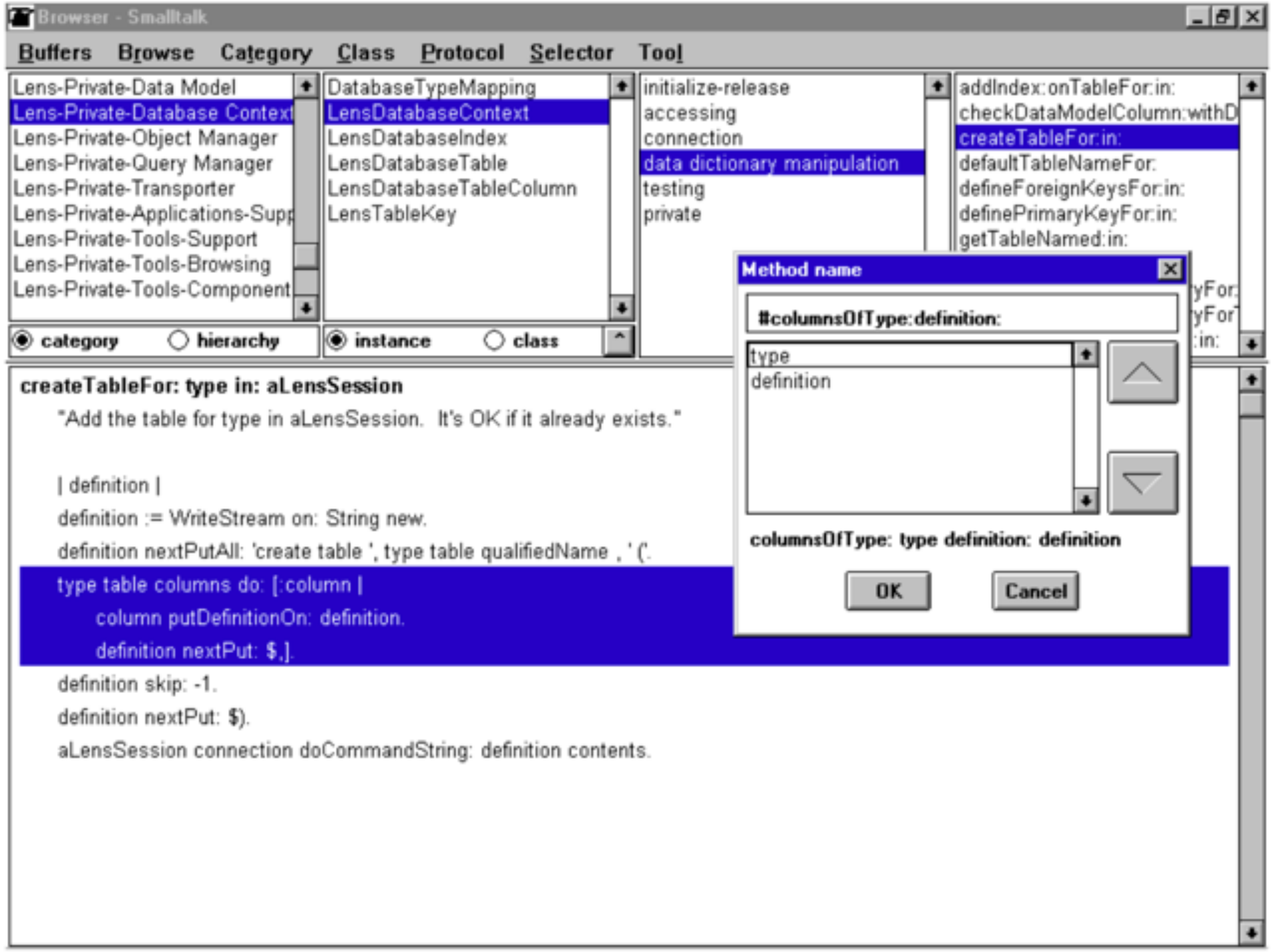
Refactoring: Motivation

"Refactoring is the process of changing a software system in such a way that it ***does not alter the external behavior*** of the code yet ***improves its internal structure***." [Fowler 1999]

M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts (1999), "*Refactoring: Improving the Design of Existing Code"*, 1st ed. Addison-Wesley Professional.

Slides for this section adapted from 05-899D Human Aspects of Software Development Spring 2011, "Software Evolution" by YoungSeok Yoon

First tool: A Refactoring Tool for Smalltalk



```
createTableFor: type in: aLensSession
    "Add the table for type in aLensSession.  It's OK if it already exists."

    | definition |
    definition := WriteStream on: String new.
    definition nextPutAll: 'create table ', type table qualifiedName , ' ('.
    type table columns do: [:column |
        column putDefinitionOn: definition.
        definition nextPut: $,].
    definition skip: -1.
    definition nextPut: $).
    aLensSession connection doCommandString: definition contents.
```

#columnsOfType:definition:

columnsOfType: type definition: definition

Figure 2 - Screenshot of Refactoring Browser during extract code as method refactoring

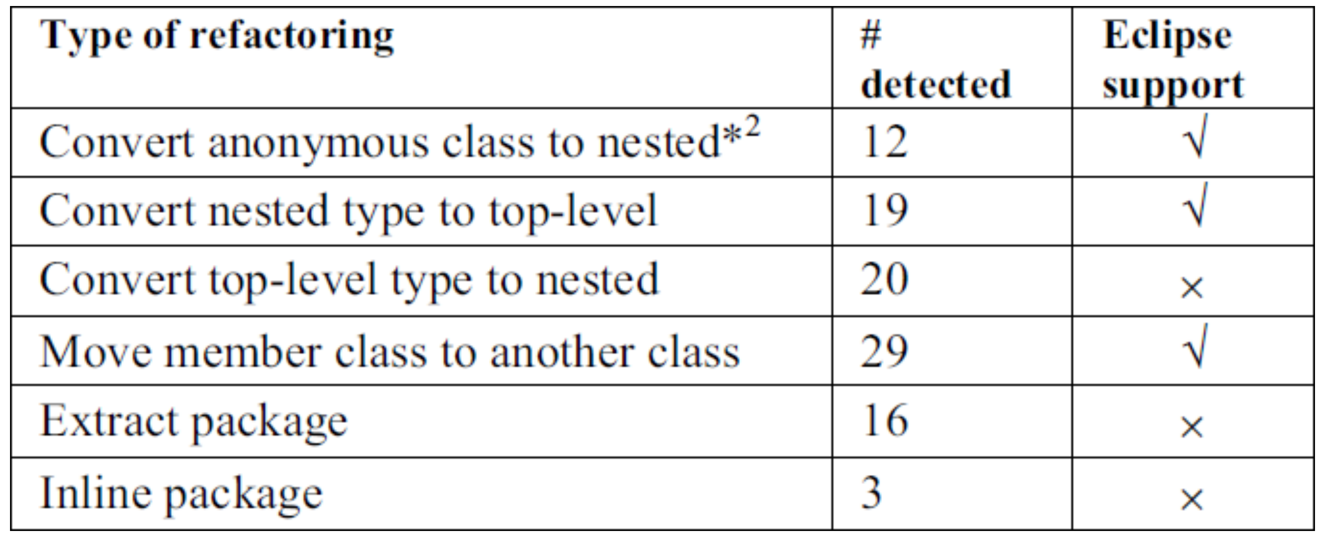
D. Roberts, J. Brant, and R. Johnson (1997), “A refactoring tool for smalltalk,” *Theory and Practice of Object Systems*, vol. 3, no. 4, pp. 253-263.

(Very) brief story of refactoring

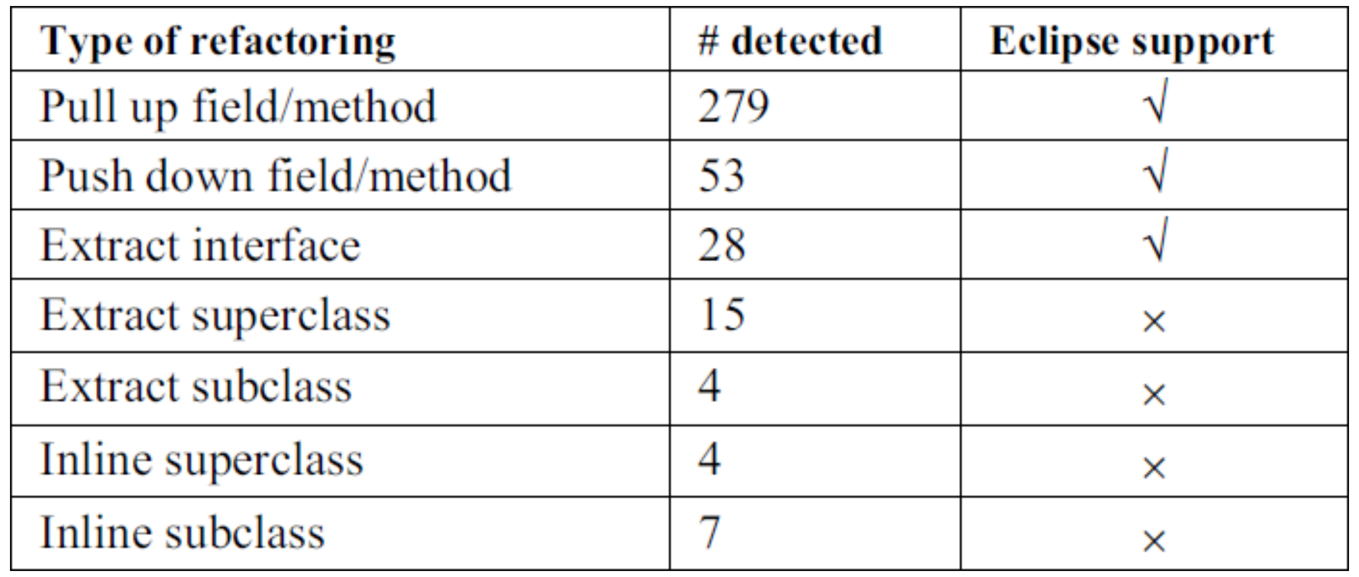
- Started with academic work defining idea of refactoring
  - William F. Opdyke. Refactoring Object-Oriented Frameworks. PhD thesis, University of Illinois, 1992.
- Academic work for tools quickly followed (e.g., [Brant TPOS97])
  - Built in real IDE for Smalltalk from beginning
- Disseminated by agile thought leaders like Martin Fowler
- Adopted into mainstream IDEs like Eclipse, Visual Studio
- Became standard accepted feature of IDES
- Research continued
  - Do developers use refactoring tools?
  - Could they use them more?
  - How could refactoring tools better support developers?

Developers manually perform refactorings not yet supported by tools

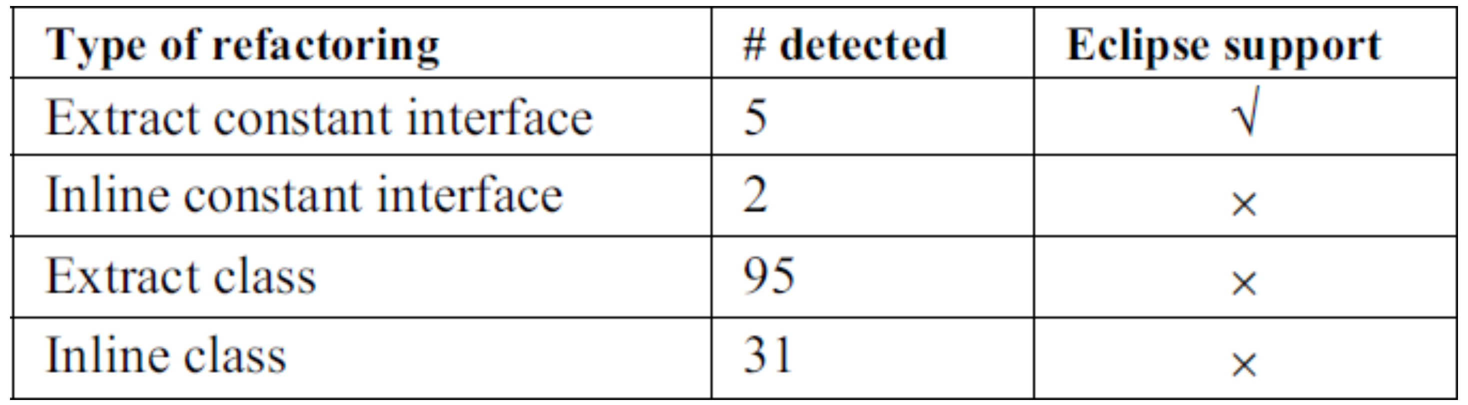
- About 70% of structural changes may be due to refactorings
- About 60% of these changes, the references to the affected entities in a component-based application can be automatically updated
- State-of-the-art IDEs only support a subset of common low-level refactorings, and lack support for more complex ones



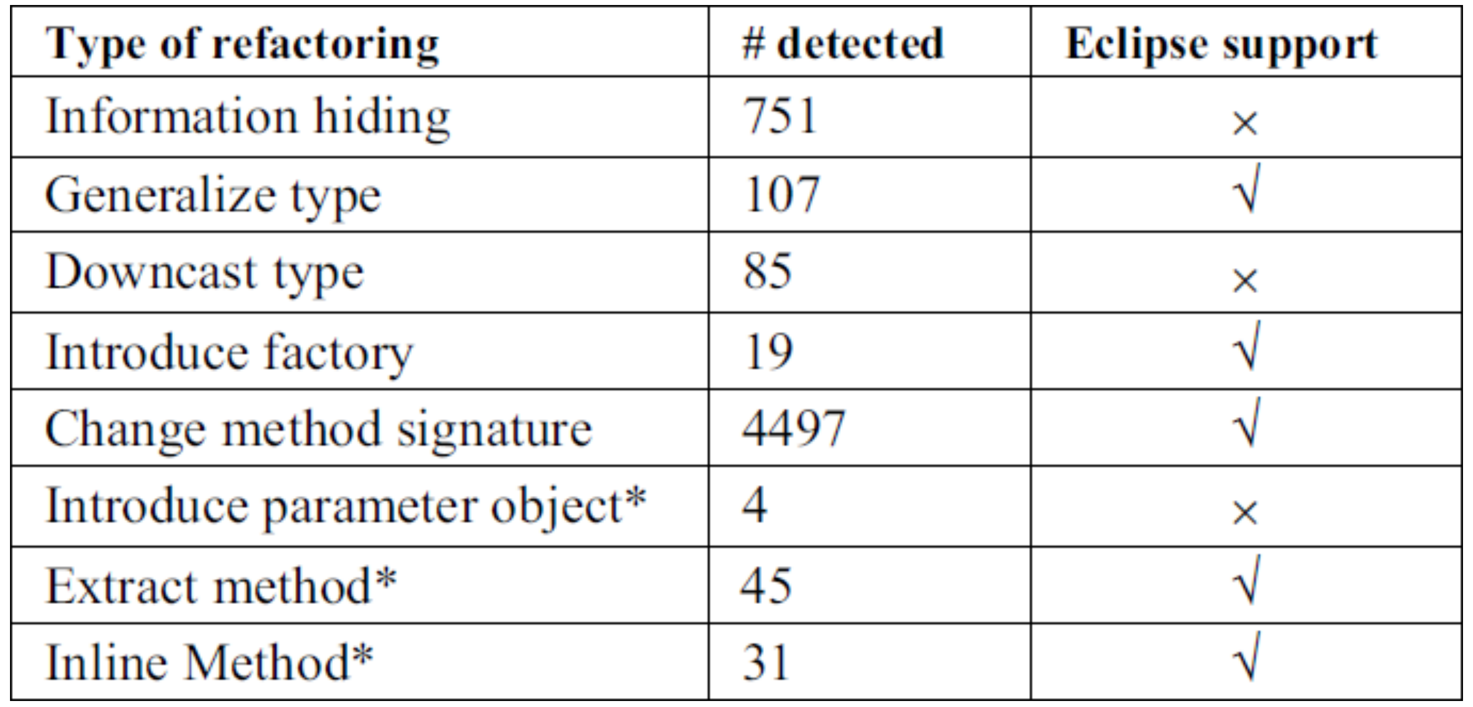
Type of refactoring	# detected	Eclipse support
Convert anonymous class to nested*[2]	12	√
Convert nested type to top-level	19	√
Convert top-level type to nested	20	×
Move member class to another class	29	√
Extract package	16	×
Inline package	3	×



Type of refactoring	# detected	Eclipse support
Pull up field/method	279	√
Push down field/method	53	√
Extract interface	28	√
Extract superclass	15	×
Extract subclass	4	×
Inline superclass	4	×
Inline subclass	7	×



Type of refactoring	# detected	Eclipse support
Extract constant interface	5	√
Inline constant interface	2	×
Extract class	95	×
Inline class	31	×



Type of refactoring	# detected	Eclipse support
Information hiding	751	×
Generalize type	107	√
Downcast type	85	×
Introduce factory	19	√
Change method signature	4497	√
Introduce parameter object*	4	×
Extract method*	45	√
Inline Method*	31	√

Z. Xing and E. Stroulia (2006), “Refactoring Practice: How it is and How it Should be Supported - An Eclipse Case Study,” in *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM ‘06)*, 2006, pp. 458-468.

How developers refactor

- The RENAME refactoring tool is used much more frequently by ordinary programmers than by the developers of refactoring tools
- About 40 percent of refactorings performed using a tool occur in batches
- About 90 percent of configuration defaults in refactoring tools are not changed when programmers use the tools
- Programmers frequently floss refactor, that is, they interleave refactoring with other types of programming activity
- About half of refactorings are not high level, so refactoring detection tools that look exclusively for high-level refactorings will not detect them
- Refactorings are performed frequently
- Close to 90 percent of refactorings are performed manually, without the help of tools

Editable progam views

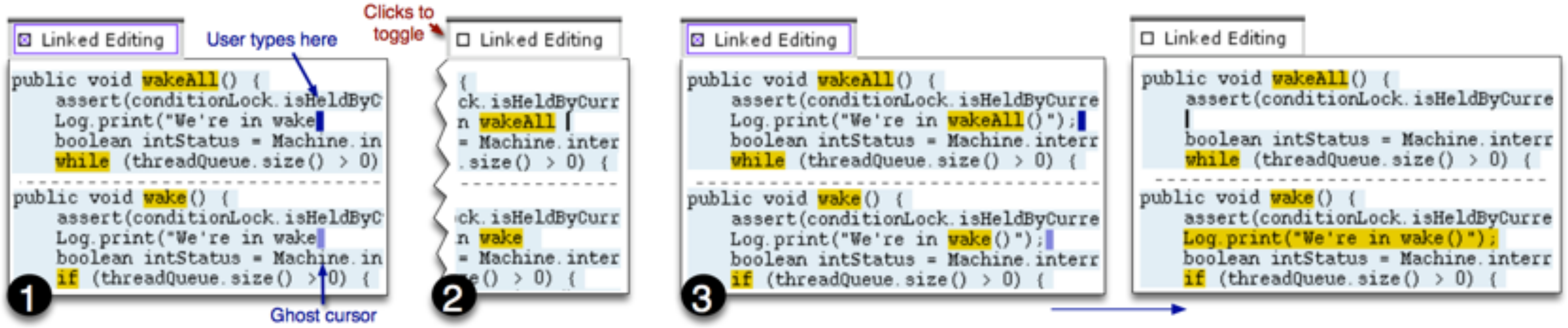
Editable program views

- Expressing code edits through textual changes can be time consuming
  - extra boilerplate, code duplication, etc.

- Key idea: Enable developers to instead interact with abstracted view of code
  - Use edits to abstract view to edit underlying code

- More control than a traditional refactoring tool --> transformation to be done controlled by the developer

Linked Editing



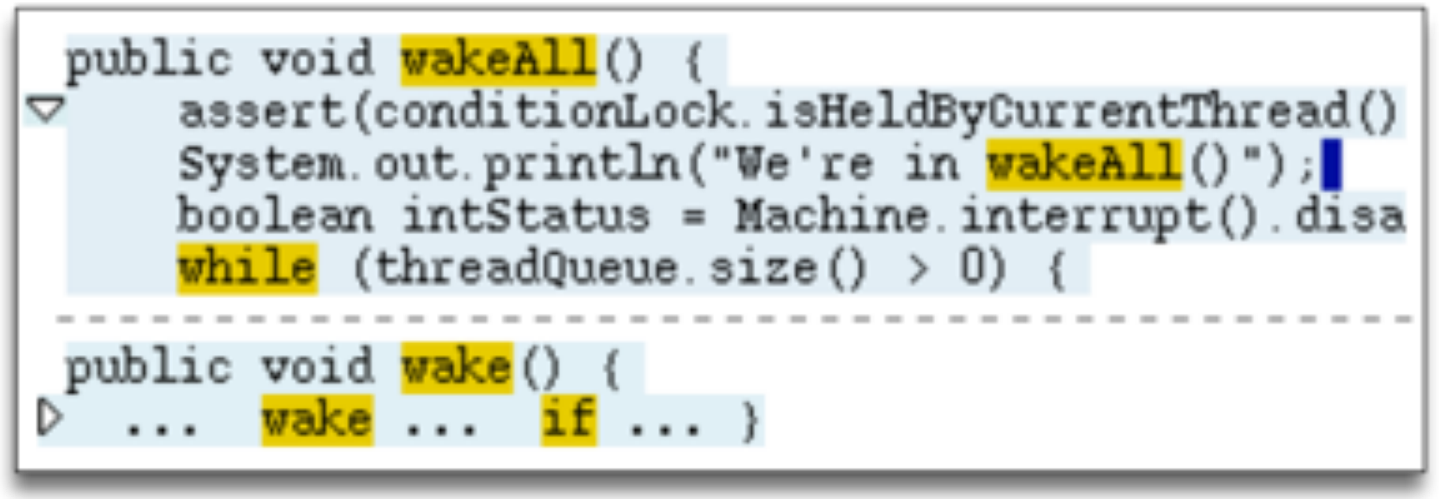
```
public void wakeAll() {
    assert(conditionLock.isHeldByC
    Log.print("We're in wake
    boolean intStatus = Machine.in
    while (threadQueue.size() > 0)
- - - - - - - - - - - - - - - - - -
public void wake() {
    assert(conditionLock.isHeldByC
    Log.print("We're in wake
    boolean intStatus = Machine.in
    if (threadQueue.size() > 0) {
```

```
{
ck.isHeldByCurr
n wakeAll
= Machine.inter
.size() > 0) {
- - - - - - - -
ck.isHeldByCurr
n wake
= Machine.inter
e() > 0) {
```

```
public void wakeAll() {
    assert(conditionLock.isHeldByCurre
    Log.print("We're in wakeAll()");
    boolean intStatus = Machine.interr
    while (threadQueue.size() > 0) {
- - - - - - - - - - - - - - - - - - - -
public void wake() {
    assert(conditionLock.isHeldByCurre
    Log.print("We're in wake()");
    boolean intStatus = Machine.interr
    if (threadQueue.size() > 0) {
```

```
public void wakeAll() {
    assert(conditionLock.isHeldByCurre

    boolean intStatus = Machine.interr
    while (threadQueue.size() > 0) {
- - - - - - - - - - - - - - - - - - - -
public void wake() {
    assert(conditionLock.isHeldByCurre
    Log.print("We're in wake()");
    boolean intStatus = Machine.interr
    if (threadQueue.size() > 0) {
```

Figure 2. (1) Adding a line to two clones. (2) Modifying one instance. (3) Deleting line in one instance.

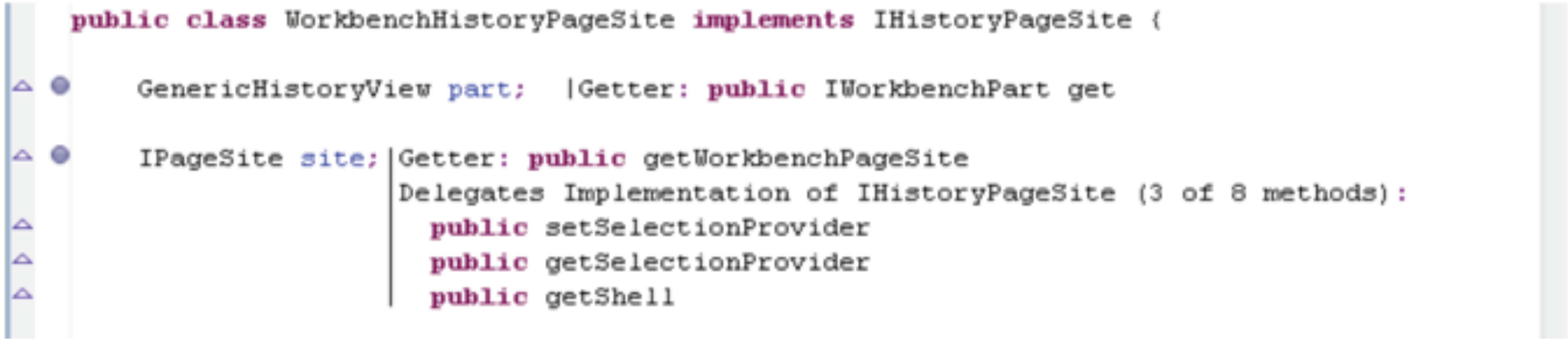


```
public void wakeAll() {
▽   assert(conditionLock.isHeldByCurrentThread()
    System.out.println("We're in wakeAll()");
    boolean intStatus = Machine.interrupt().disa
    while (threadQueue.size() > 0) {
- - - - - - - - - - - - - - - - - - - - - - - - -
public void wake() {
▷  ... wake ... if ... }
```

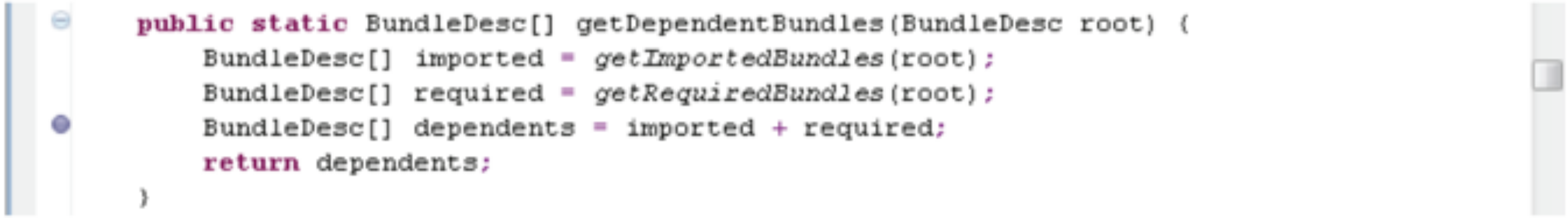
Figure 3. An elided clone looks similar to a function definition and use

Michael Toomim, Andrew Begel, and Susan L. Graham. 2004. Managing Duplicated Code with Linked Editing. In Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04). IEEE Computer Society, Washington, DC, USA, 173-180.

Editable views

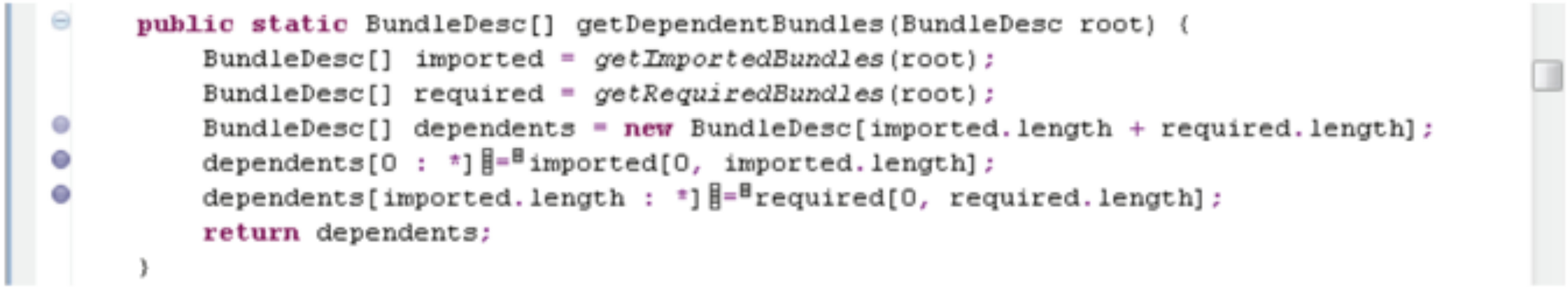


```
public class WorkbenchHistoryPageSite implements IHistoryPageSite {

    GenericHistoryView part;   |Getter: public IWorkbenchPart get

    IPageSite site; |Getter: public getWorkbenchPageSite
                    |Delegates Implementation of IHistoryPageSite (3 of 8 methods):
                    |  public setSelectionProvider
                    |  public getSelectionProvider
                    |  public getShell
```



```
public static BundleDesc[] getDependentBundles(BundleDesc root) {
    BundleDesc[] imported = getImportedBundles(root);
    BundleDesc[] required = getRequiredBundles(root);
    BundleDesc[] dependents = imported + required;
    return dependents;
}
```

(a) An array-concatenation registration. The presentation uses an overloaded "+" to indicate the concatenation of two arrays through calls to `System.arraycopy`.



```
public static BundleDesc[] getDependentBundles(BundleDesc root) {
    BundleDesc[] imported = getImportedBundles(root);
    BundleDesc[] required = getRequiredBundles(root);
    BundleDesc[] dependents = new BundleDesc[imported.length + required.length];
    dependents[0 : *]▯=▯imported[0, imported.length];
    dependents[imported.length : *]▯=▯required[0, required.length];
    return dependents;
}
```

(b) Two arraycopy registrations. The notation "0 : *" indicates that the elements are copied into the indices starting at 0. An icon is used to disambiguate the syntax, by making it clear that the dependents array is not truncated to the length of the copied elements.

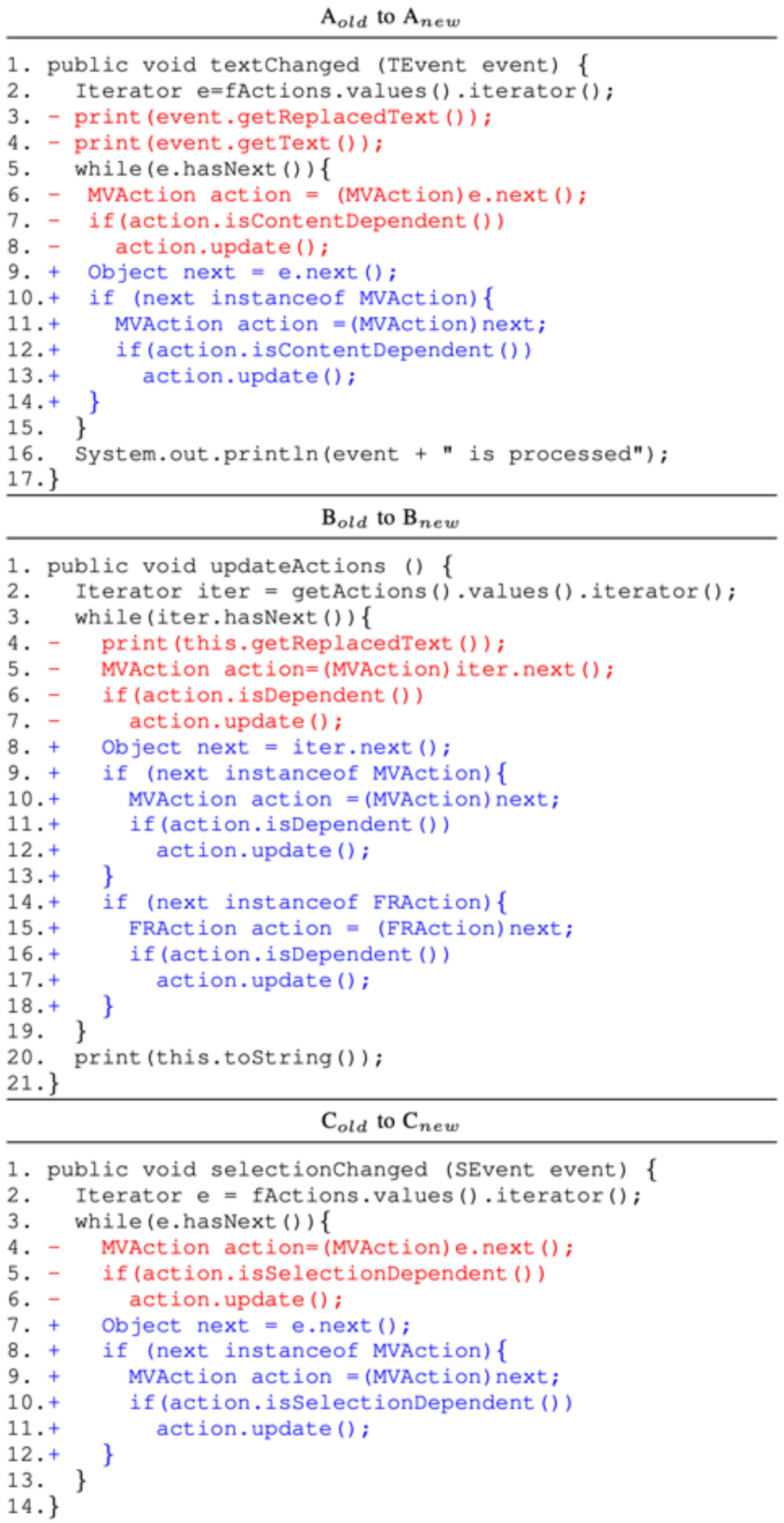
Samuel Davis and Gregor Kiczales. 2010. Registration-based language abstractions. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '10). ACM, New York, NY, USA, 754-773.

Supporting systematic edits

- Developers sometimes make edits to multiple files that are very similar

- Tool idea: find commonality in edits between 2 or more examples, generalize to others

Example



A_{old} to A_{new}

```
public void textChanged (TEvent event) {
  Iterator e=fActions.values().iterator();
- print(event.getReplacedText());
- print(event.getText());
  while(e.hasNext()){
-  MVAction action = (MVAction)e.next();
-  if(action.isContentDependent())
-    action.update();
+  Object next = e.next();
+  if (next instanceof MVAction){
+    MVAction action =(MVAction)next;
+    if(action.isContentDependent())
+      action.update();
+  }
  }
  System.out.println(event + " is processed");
}
```

B_{old} to B_{new}

```
public void updateActions () {
  Iterator iter = getActions().values().iterator();
  while(iter.hasNext()){
-   print(this.getReplacedText());
-   MVAction action=(MVAction)iter.next();
-   if(action.isDependent())
-     action.update();
+   Object next = iter.next();
+   if (next instanceof MVAction){
+     MVAction action =(MVAction)next;
+     if(action.isDependent())
+       action.update();
+   }
+   if (next instanceof FRAction){
+     FRAction action = (FRAction)next;
+     if(action.isDependent())
+       action.update();
+   }
  }
  print(this.toString());
}
```

C_{old} to C_{new}

```
public void selectionChanged (SEvent event) {
  Iterator e = fActions.values().iterator();
  while(e.hasNext()){
-   MVAction action=(MVAction)e.next();
-   if(action.isSelectionDependent())
-     action.update();
+   Object next = e.next();
+   if (next instanceof MVAction){
+     MVAction action =(MVAction)next;
+     if(action.isSelectionDependent())
+       action.update();
+   }
  }
}
```

Fig. 1. A systematic edit to three methods based on revisions from 2007-04-16 and 2007-04-30 to org.eclipse.compare

Na Meng, Miryung Kim, and Kathryn S. McKinley. 2013. LASE: locating and applying systematic edits by learning from examples. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, 502-511.

Locating and applying systematic edits

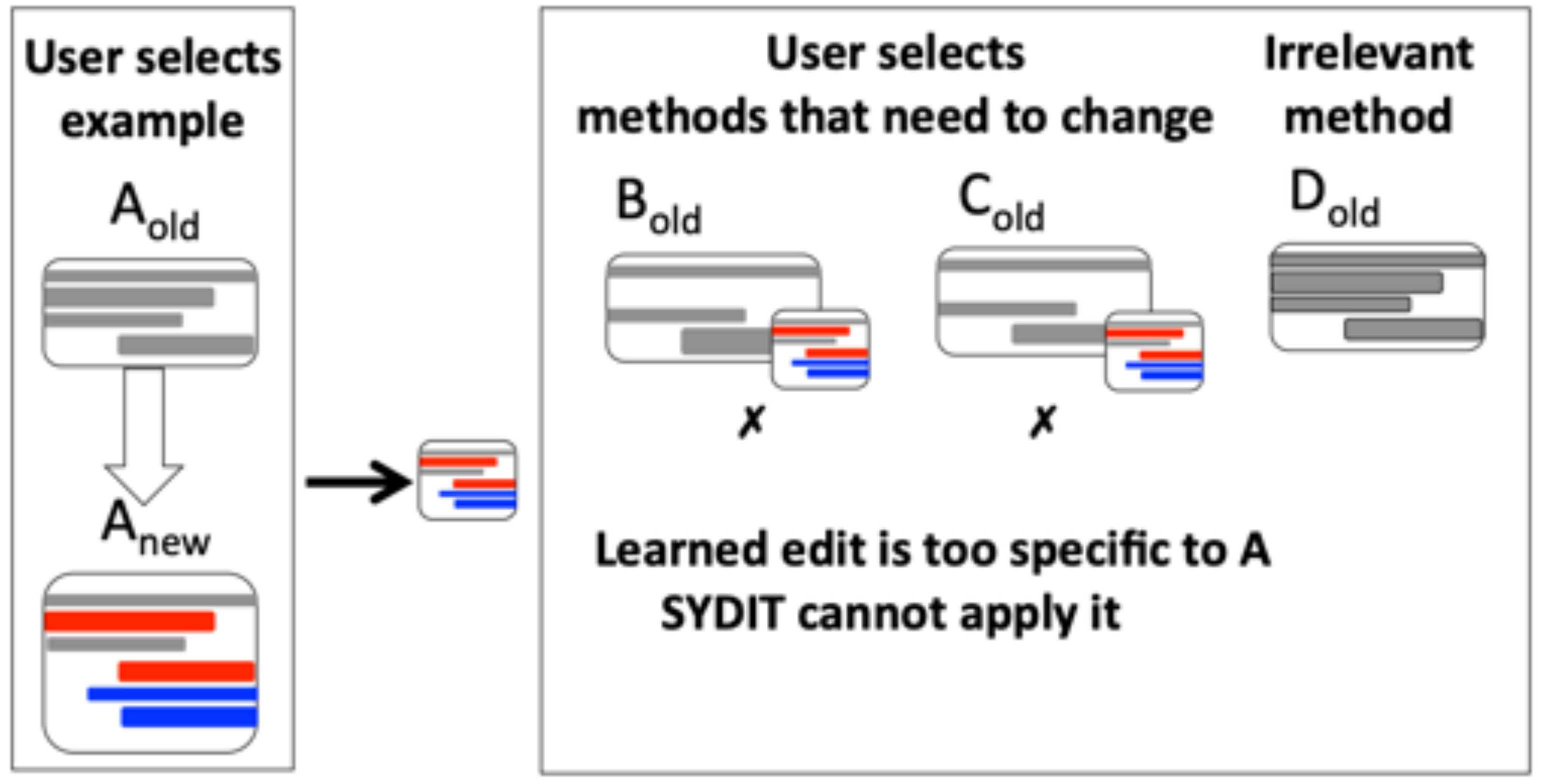


Fig. 2. SYDIT learns an edit from one example. A developer must locate and specify the other methods to change.

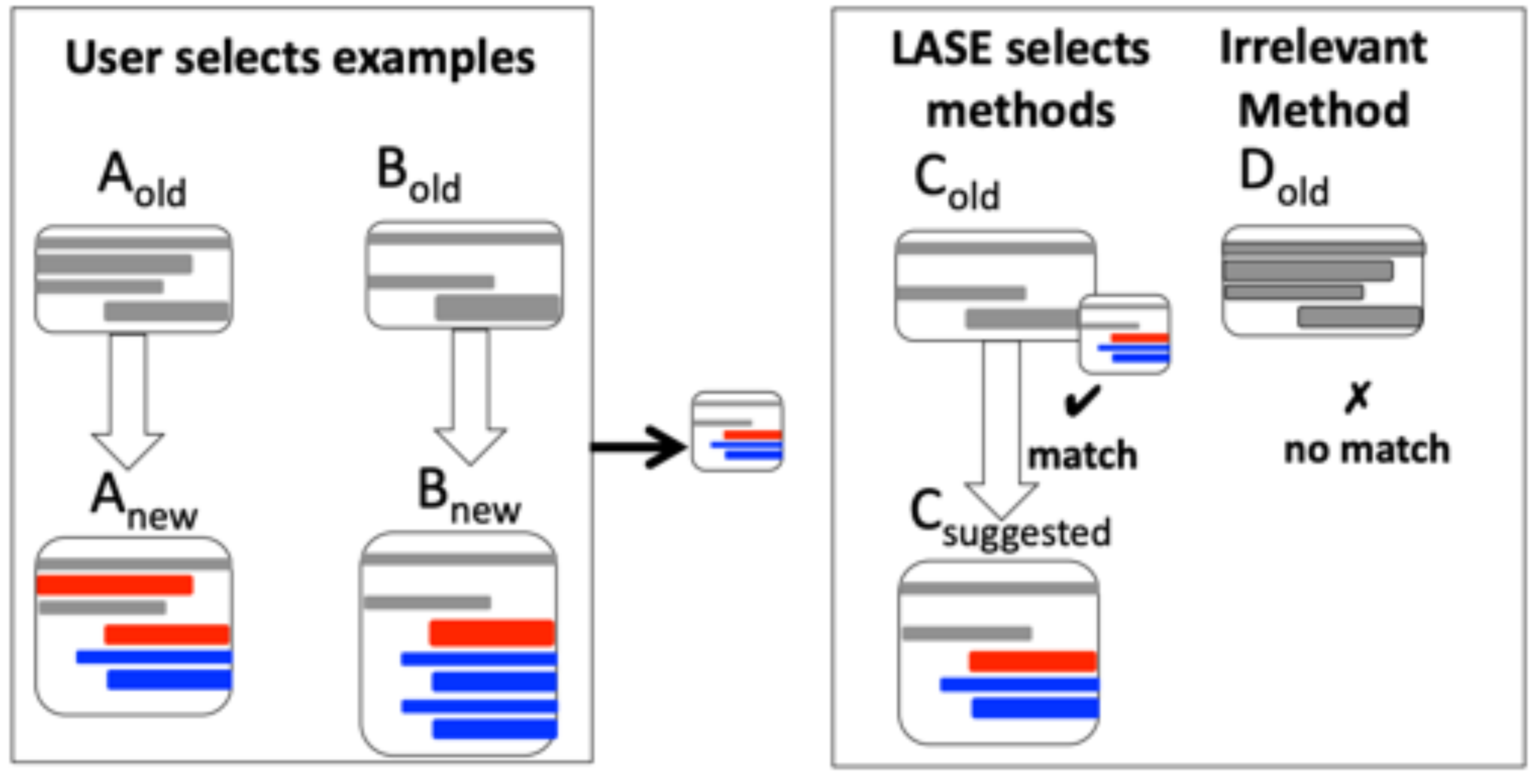
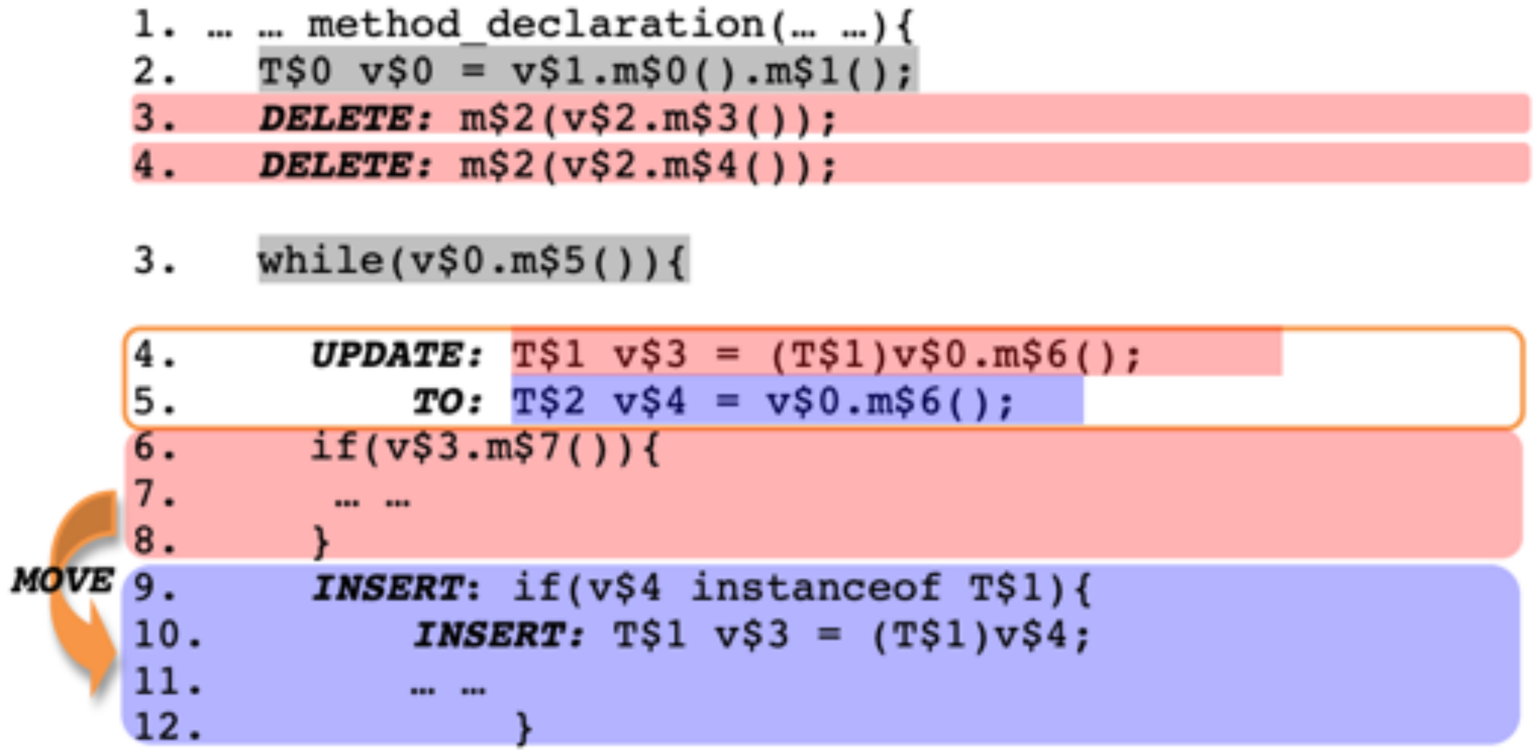
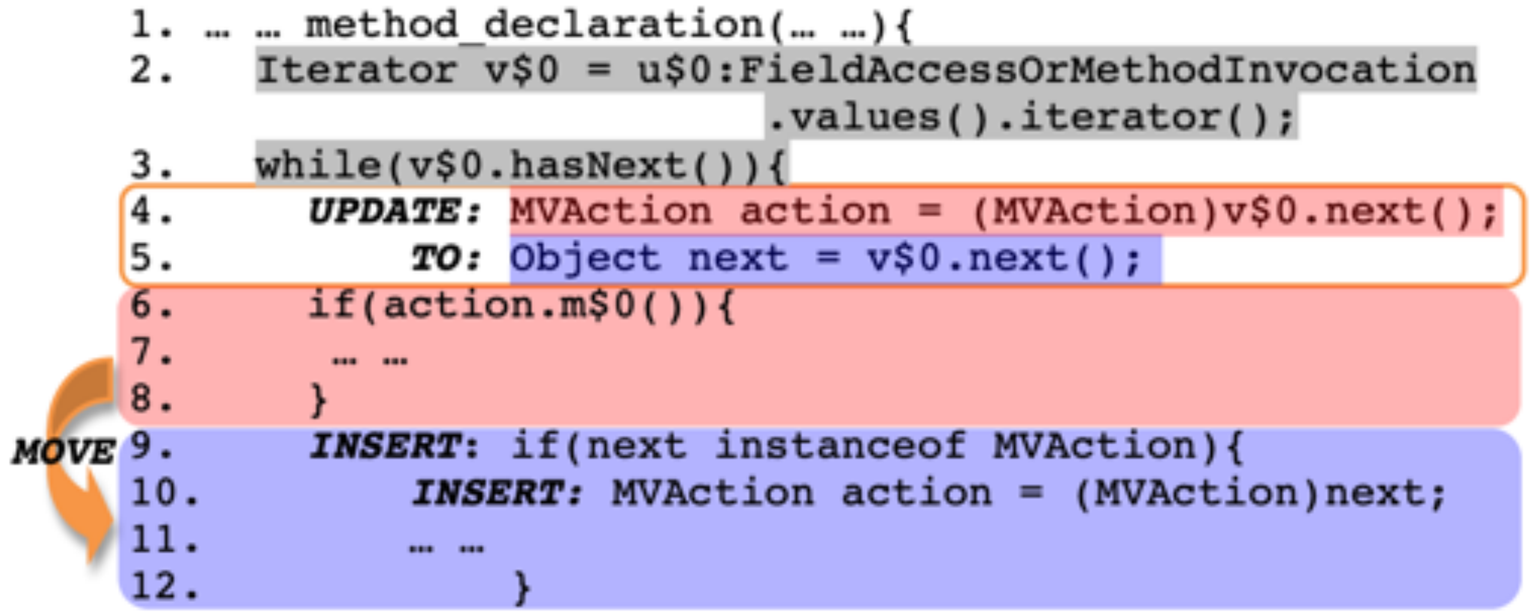


Fig. 3. LASE learns an edit from two or more examples. LASE locates other methods to change.



```
… … method_declaration(… …){
    T$0 v$0 = v$1.m$0().m$1();
    DELETE: m$2(v$2.m$3());
    DELETE: m$2(v$2.m$4());

    while(v$0.m$5()){

      UPDATE: T$1 v$3 = (T$1)v$0.m$6();
          TO: T$2 v$4 = v$0.m$6();
      if(v$3.m$7()){
       … …
      }
      INSERT: if(v$4 instanceof T$1){
          INSERT: T$1 v$3 = (T$1)v$4;
          … …
            }
```

Fig. 4. Edit script from SYDIT abstracts all concrete names. Gray marks edit context, red marks deletions, and blue marks additions.



```
… … method_declaration(… …){
    Iterator v$0 = u$0:FieldAccessOrMethodInvocation
                          .values().iterator();
    while(v$0.hasNext()){
      UPDATE: MVAction action = (MVAction)v$0.next();
          TO: Object next = v$0.next();
      if(action.m$0()){
       … …
      }
      INSERT: if(next instanceof MVAction){
          INSERT: MVAction action = (MVAction)next;
          … …
            }
```

Fig. 5. Edit script from LASE abstracts code names that differ in the examples and uses concrete names for common ones. Gray marks edit context, red marks deletions, and blue marks additions.

Na Meng, Miryung Kim, and Kathryn S. McKinley. 2013. LASE: locating and applying systematic edits by learning from examples. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, P

API & Language Migration

API & Language Migration

- You wrote code in Java, now you want to use it in a Python project.

- React just introduced 3 new features. You want to update your code to use these.

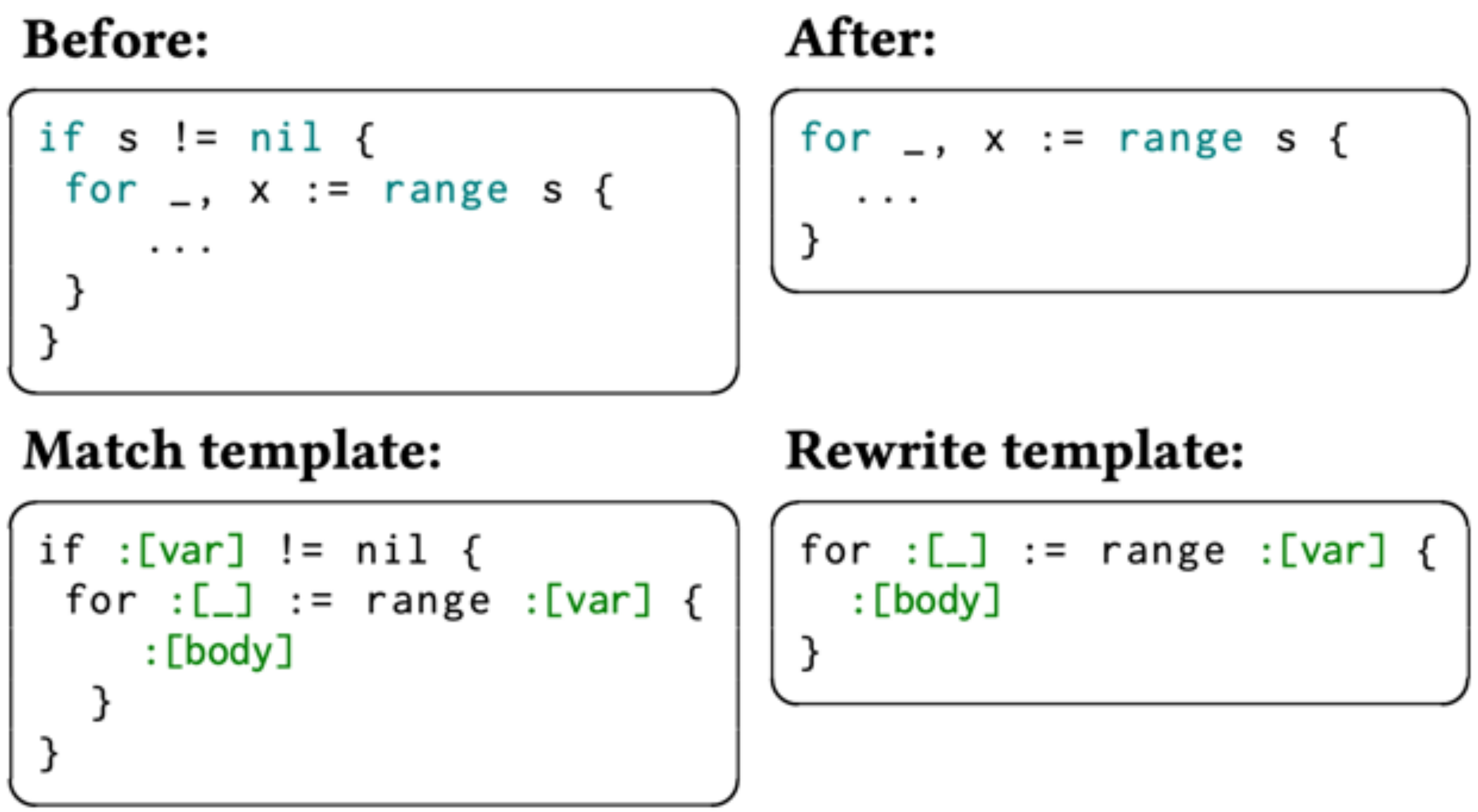
Change

- What happens when an upstream system introduces a change?
- Backwards compatible change: upstream system provides everything they did before and more
  - Nothing needs to change on downstream system
  - Just have new functionality to be used
- Breaking change: upstream system no longer fulfills contract it did before
  - Method might be deprecated, renamed, or changed in its behavior
- Burden of change
  - Downstream system will not work until is updated to work with new version

Program transformation for API migration

- Describe a code pattern to find all the code you want to update
- Describe a code pattern that describes what the code should be updated to

Specifying program transformations



Before:

```
if s != nil {
 for _, x := range s {
    ...
 }
}
```

After:

```
for _, x := range s {
  ...
}
```

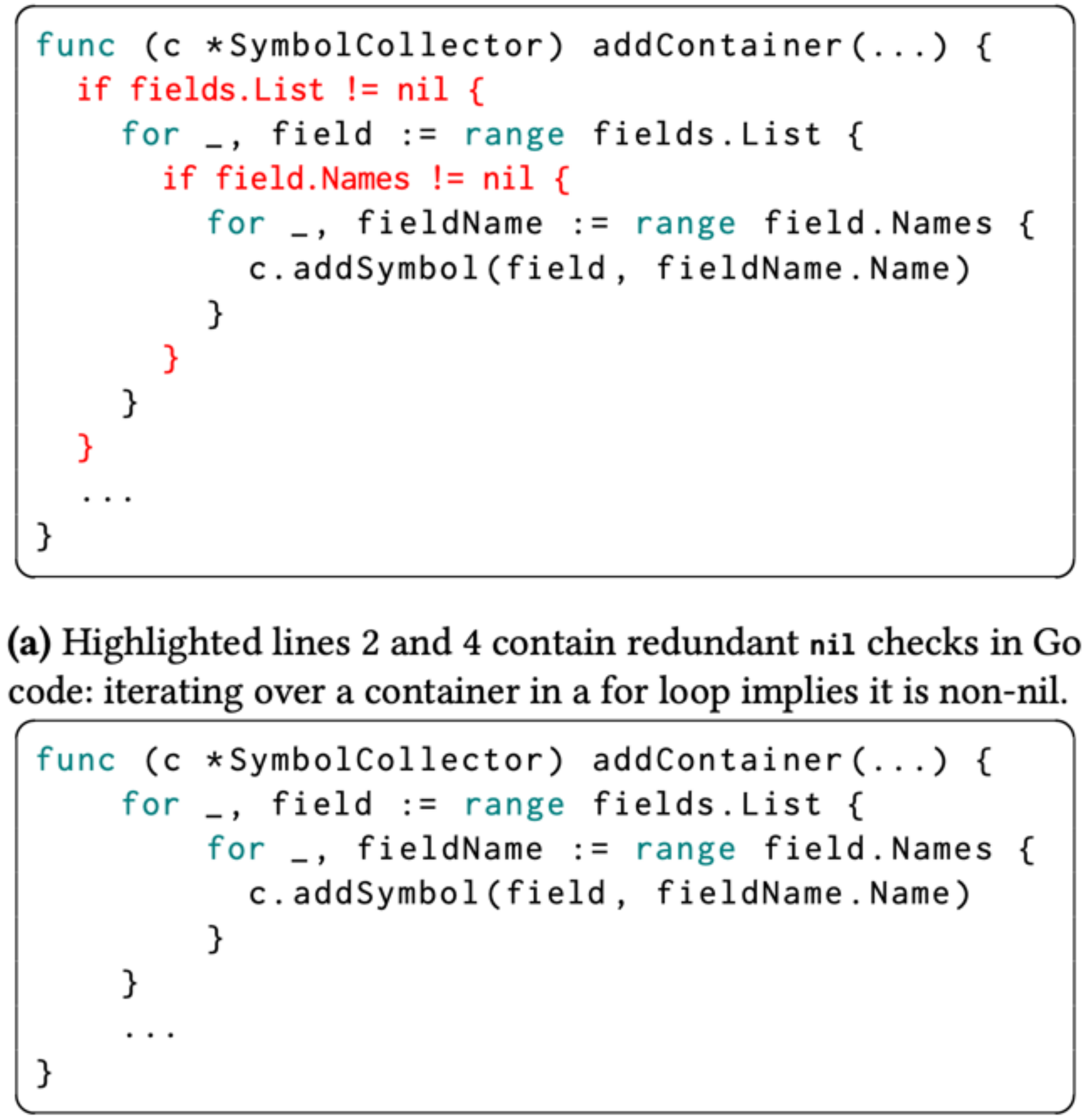
Match template:

```
if :[var] != nil {
 for :[_] := range :[var] {
    :[body]
  }
}
```

Rewrite template:

```
for :[_] := range :[var] {
  :[body]
}
```

Figure 1. *Top:* A textual description for simplifying a nil check Go code, taken from the Go staticcheck tool. *Bottom:* Our match template and rewrite templates for the nil-check pattern above.



```
func (c *SymbolCollector) addContainer(...) {
 if fields.List != nil {
  for _, field := range fields.List {
   if field.Names != nil {
    for _, fieldName := range field.Names {
     c.addSymbol(field, fieldName.Name)
    }
   }
  }
 }
 ...
}
```

(a) Highlighted lines 2 and 4 contain redundant nil checks in Go code: iterating over a container in a for loop implies it is non-nil.

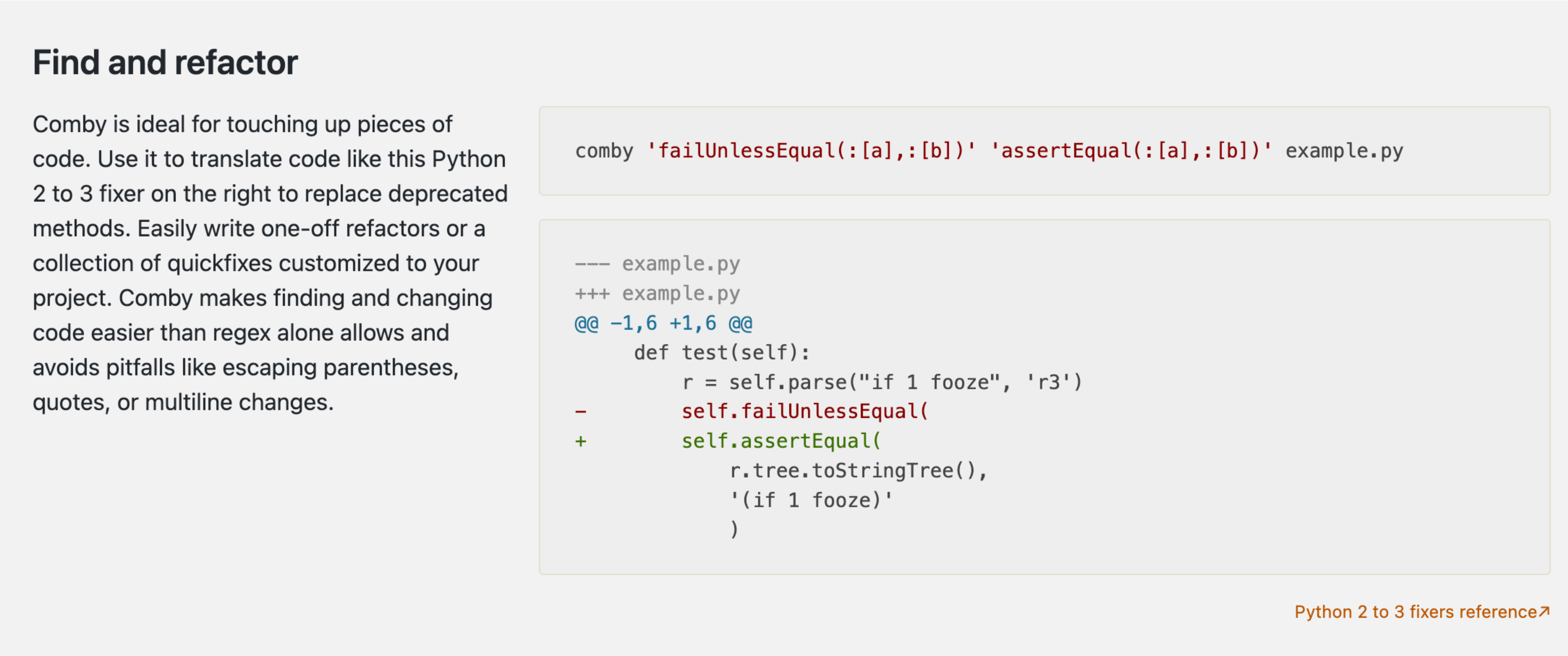
```
func (c *SymbolCollector) addContainer(...) {
  for _, field := range fields.List {
    for _, fieldName := range field.Names {
     c.addSymbol(field, fieldName.Name)
    }
  }
  ...
}
```

(b) Rewrite output simplifying the Go code above.

Figure 2. Redundant code pattern and simplification.

Comby.dev

- Tool for writing code transformations



Find and refactor

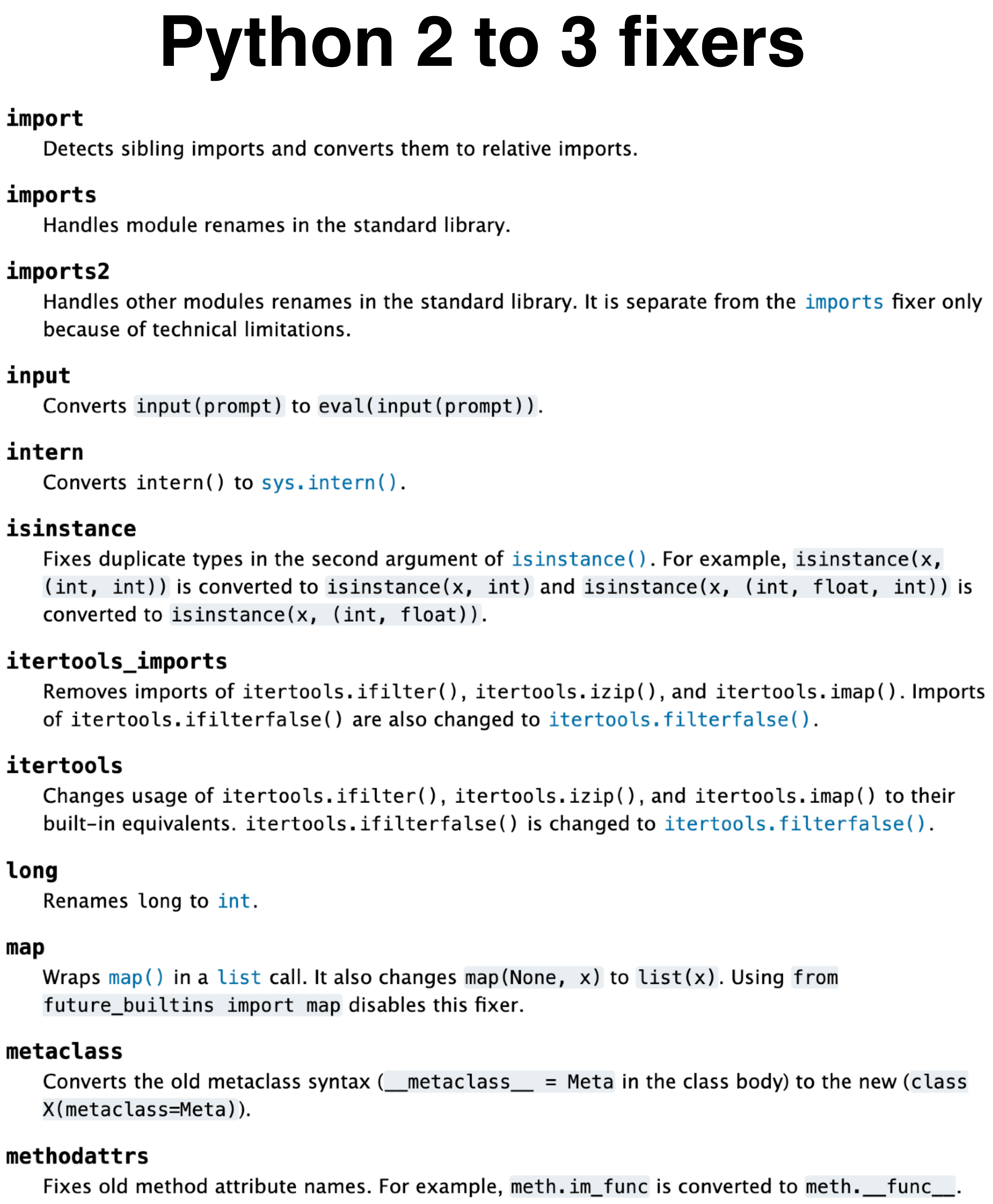
Comby is ideal for touching up pieces of code. Use it to translate code like this Python 2 to 3 fixer on the right to replace deprecated methods. Easily write one-off refactors or a collection of quickfixes customized to your project. Comby makes finding and changing code easier than regex alone allows and avoids pitfalls like escaping parentheses, quotes, or multiline changes.

```
comby 'failUnlessEqual(:[a],:[b])' 'assertEqual(:[a],:[b])' example.py
```

```
--- example.py
+++ example.py
@@ -1,6 +1,6 @@
     def test(self):
         r = self.parse("if 1 fooze", 'r3')
-        self.failUnlessEqual(
+        self.assertEqual(
             r.tree.toStringTree(),
             '(if 1 fooze)'
             )
```

Python 2 to 3 fixers reference↗

Fixers: reusable transformations



Python 2 to 3 fixers

`import`

Detects sibling imports and converts them to relative imports.

`imports`

Handles module renames in the standard library.

`imports2`

Handles other modules renames in the standard library. It is separate from the `imports` fixer only because of technical limitations.

`input`

Converts `input(prompt)` to `eval(input(prompt))`.

`intern`

Converts `intern()` to `sys.intern()`.

`isinstance`

Fixes duplicate types in the second argument of `isinstance()`. For example, `isinstance(x, (int, int))` is converted to `isinstance(x, int)` and `isinstance(x, (int, float, int))` is converted to `isinstance(x, (int, float))`.

`itertools_imports`

Removes imports of `itertools.ifilter()`, `itertools.izip()`, and `itertools.imap()`. Imports of `itertools.ifilterfalse()` are also changed to `itertools.filterfalse()`.

`itertools`

Changes usage of `itertools.ifilter()`, `itertools.izip()`, and `itertools.imap()` to their built-in equivalents. `itertools.ifilterfalse()` is changed to `itertools.filterfalse()`.

`long`

Renames `long` to `int`.

`map`

Wraps `map()` in a `list` call. It also changes `map(None, x)` to `list(x)`. Using `from future_builtins import map` disables this fixer.

`metaclass`

Converts the old metaclass syntax (`__metaclass__ = Meta` in the class body) to the new (`class X(metaclass=Meta)`).

`methodattrs`

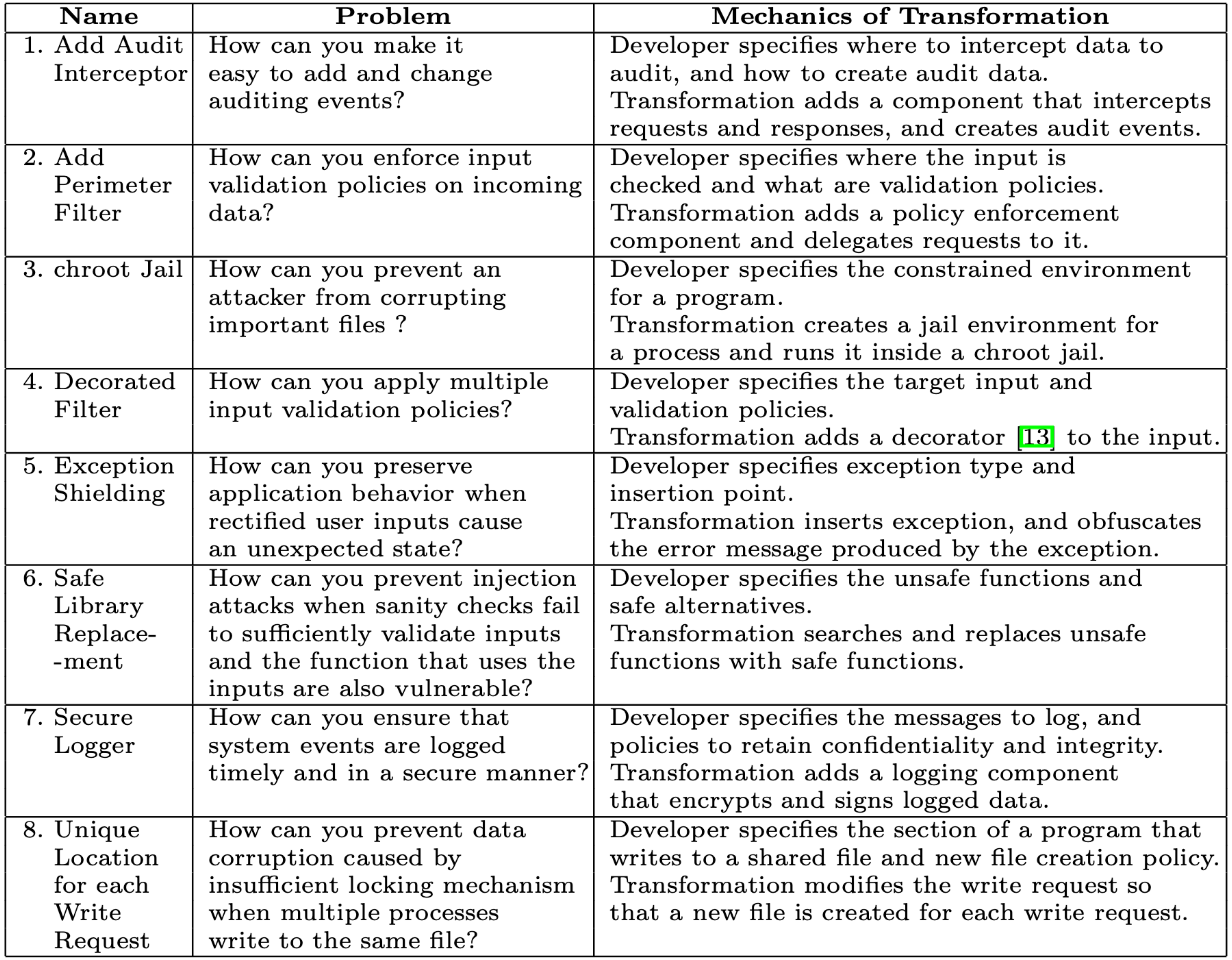
Fixes old method attribute names. For example, `meth.im_func` is converted to `meth.__func__`.

Security fixes

Security fixes

- Code that is insecure can be characterized by code patterns
- Find all of the code that matches an insecure pattern
- Describe how it should be changed to fix it

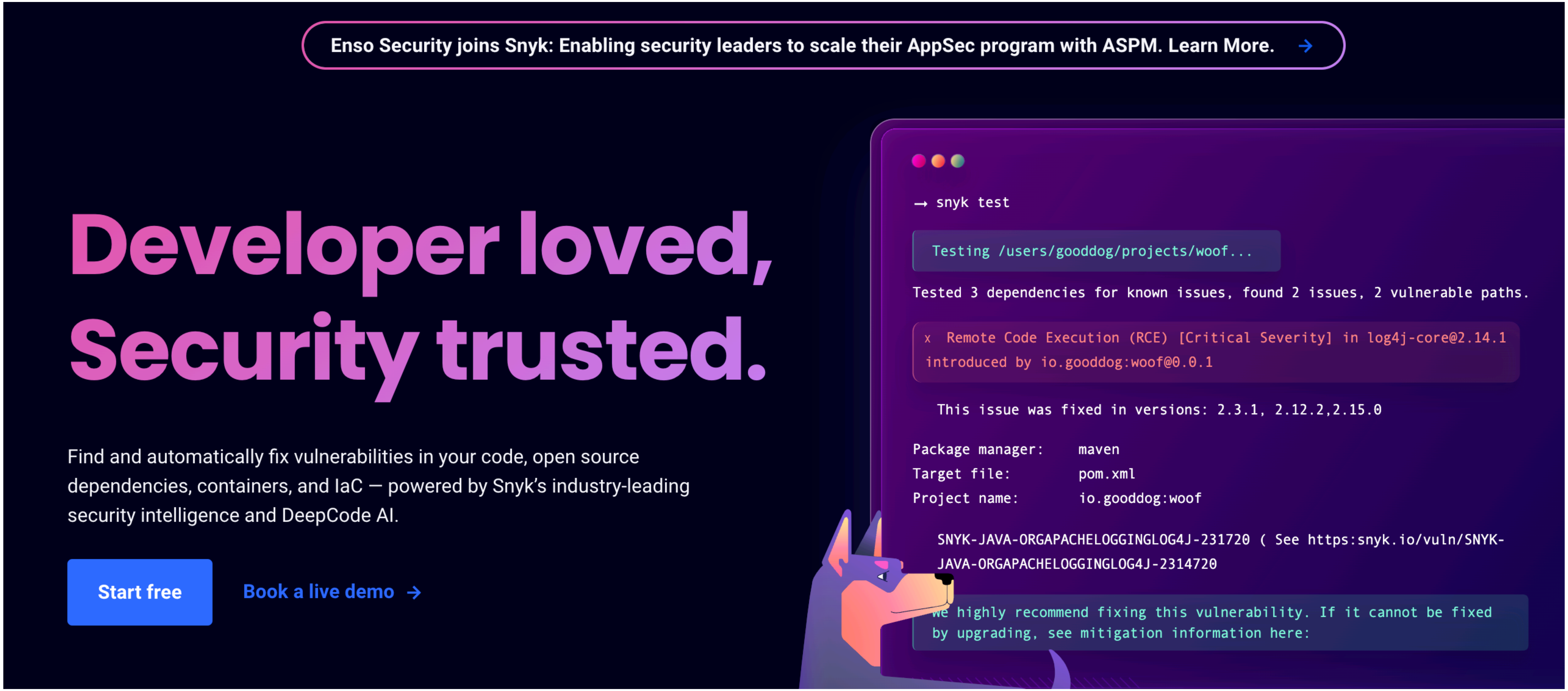
Table 1. Security-oriented program transformations to prevent injection attacks



Name	Problem	Mechanics of Transformation
1. Add Audit Interceptor	How can you make it easy to add and change auditing events?	Developer specifies where to intercept data to audit, and how to create audit data. Transformation adds a component that intercepts requests and responses, and creates audit events.
2. Add Perimeter Filter	How can you enforce input validation policies on incoming data?	Developer specifies where the input is checked and what are validation policies. Transformation adds a policy enforcement component and delegates requests to it.
3. chroot Jail	How can you prevent an attacker from corrupting important files ?	Developer specifies the constrained environment for a program. Transformation creates a jail environment for a process and runs it inside a chroot jail.
4. Decorated Filter	How can you apply multiple input validation policies?	Developer specifies the target input and validation policies. Transformation adds a decorator [13] to the input.
5. Exception Shielding	How can you preserve application behavior when rectified user inputs cause an unexpected state?	Developer specifies exception type and insertion point. Transformation inserts exception, and obfuscates the error message produced by the exception.
6. Safe Library Replace--ment	How can you prevent injection attacks when sanity checks fail to sufficiently validate inputs and the function that uses the inputs are also vulnerable?	Developer specifies the unsafe functions and safe alternatives. Transformation searches and replaces unsafe functions with safe functions.
7. Secure Logger	How can you ensure that system events are logged timely and in a secure manner?	Developer specifies the messages to log, and policies to retain confidentiality and integrity. Transformation adds a logging component that encrypts and signs logged data.
8. Unique Location for each Write Request	How can you prevent data corruption caused by insufficient locking mechanism when multiple processes write to the same file?	Developer specifies the section of a program that writes to a shared file and new file creation policy. Transformation modifies the write request so that a new file is created for each write request.

M. Hafiz, P. Adamczyk and R. Johnson. Systematically Eradicating Data Injection Attacks using Security-oriented Program Transformations. In ESSoS09: Symposium on Engineering Secure Software and Systems. Leuven, Belgium. Feb, 2009

Snyk Demo



Enso Security joins Snyk: Enabling security leaders to scale their AppSec program with ASPM. Learn More. →

Developer loved, Security trusted.

Find and automatically fix vulnerabilities in your code, open source dependencies, containers, and IaC — powered by Snyk's industry-leading security intelligence and DeepCode AI.

```
→ snyk test

Testing /users/gooddog/projects/woof...

Tested 3 dependencies for known issues, found 2 issues, 2 vulnerable paths.

x  Remote Code Execution (RCE) [Critical Severity] in log4j-core@2.14.1
introduced by io.gooddog:woof@0.0.1

   This issue was fixed in versions: 2.3.1, 2.12.2,2.15.0

Package manager:    maven
Target file:        pom.xml
Project name:       io.gooddog:woof

   SNYK-JAVA-ORGAPACHELOGGINGLOG4J-231720 ( See https:snyk.io/vuln/SNYK-
   JAVA-ORGAPACHELOGGINGLOG4J-2314720

  We highly recommend fixing this vulnerability. If it cannot be fixed
  by upgrading, see mitigation information here:
```

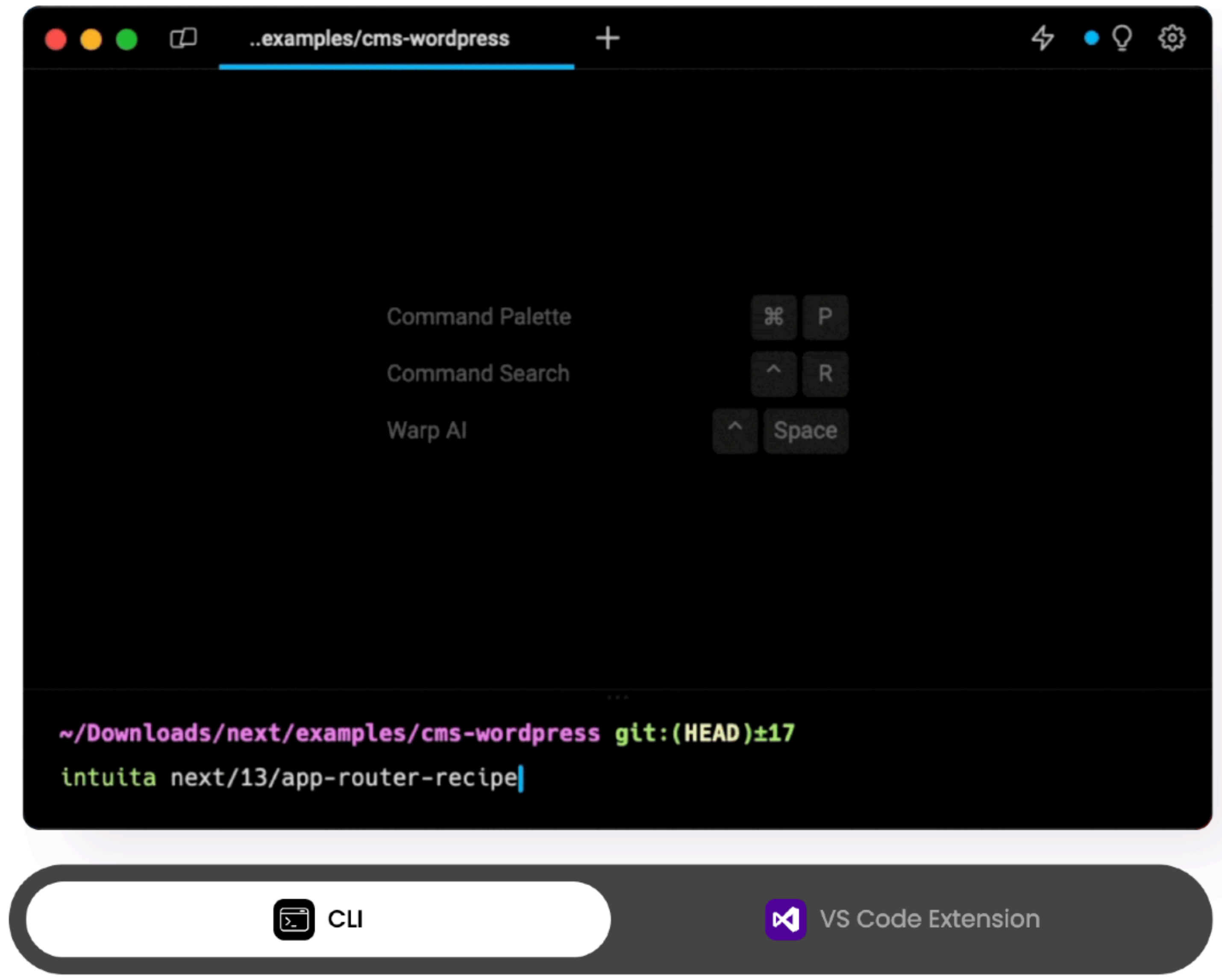
10 min break

Intuita CodeMods

Code migrations at lightning speed.

Save days of manual work by running open-source automation recipes to automate framework upgrades, such as Next.js, right from your IDE or CLI.

Get started with CLI →



```
~/Downloads/next/examples/cms-wordpress git:(HEAD)±17
intuita next/13/app-router-recipe
```

Build Automations with AI

Simply run `intuita learn` and let AI instantly build your custom code automation recipe. Automate the boring and get back to shipping more new features to end-users.

https://www.intuita.io/

In-Class Activity

- In groups of 2, try out CodeMod Studio.
  - https://codemod.studio/
  - Identify a JS migration problem (e.g., migrate old React code to use newer features such as WebHooks)
  - Read docs to understand how to build a CodeMod for your problem.
  - Build a CodeMod and try it out with sample code snippets.