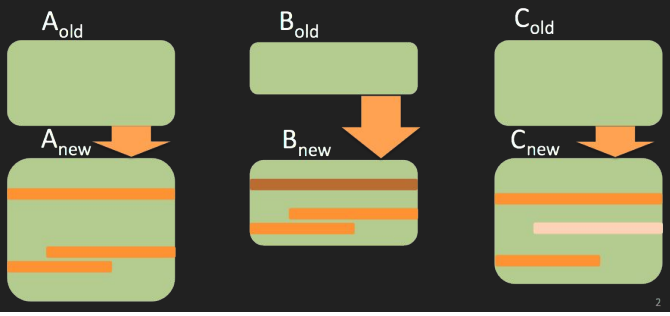
Does Automated Refactoring Obviate Systematic Editing?

Na Meng, Lisa Hua, Miryung Kim, Kathryn McKinley
ICSE'15

Presenter: Lisa Hua
12/1/2015

Motivating Scenario

- Developers may need to apply similar but not necessarily identical code changes to multiple locations
- E.g., Pat needs to update database transaction code to prevent SQL injection attacks.



Concrete Example for Systematic Edit: --- similar edits to multiple locations

```
public class CompareEditorInput {
  private ICompareContainer fContainer;
  private boolean fContainerProvided;
  private Splitter fComposite;
  public IActionBars getActionBars() {
  - if (fContainer == null) {
  + IActionBars actionBars =
fContainer.getActionBars();
  + if (actionBars == null && !
fContainerProvided) {
    return Utilities.findActionBars
(fComposite);
  }
  - return fContainer.getActionBars();
  + return actionBars;
  }
}
```

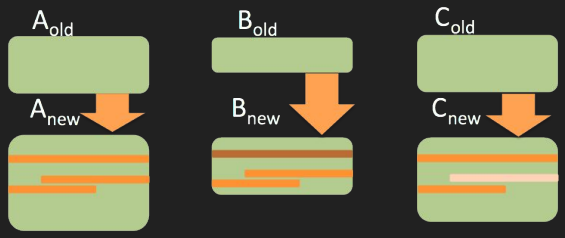
```
public ISLocator getServiceLocator{
- if (fContainer == null) {
+ ISLocator serviceLocator = fContainer.
getServiceLocator();
+ if (serviceLocator == null && !
fContainerProvided) {
   return Utilities.findSite
(fComposite);
 }
- return fContainer.getServiceLocator();
+ return serviceLocator;
 }
}}
```


Systematic Tools

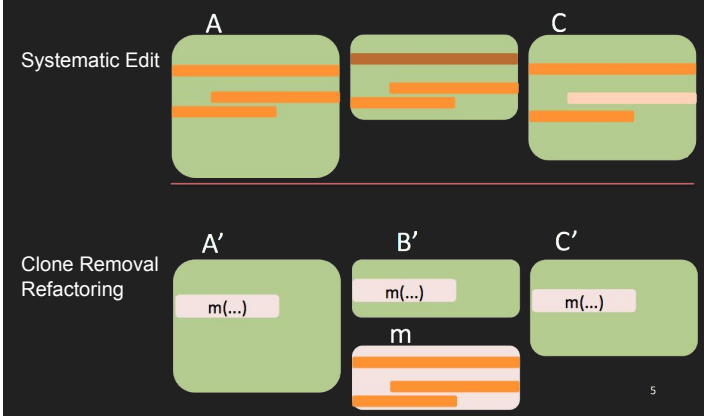
- Simultaneous text editing [2002], Linked Editing [2004], Clever [2009]
- Example-based program transformation [Meng et al. 2011, 2013]

Pro: Perform code change propagation

Con: Encourage code duplication



Two Code Maintenance Alternatives



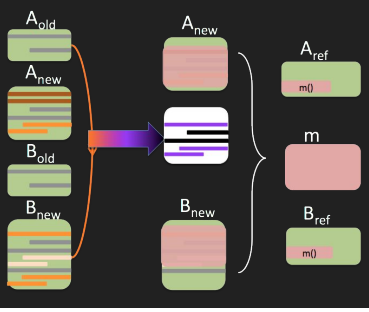
Problem Statement

- Does systematic editing encourages code duplication?
- Does clone removal refactoring eliminate the need for systematic editing?

- RASE: Exploit systematic edits for clone removal refactoring
- A case study to measure if it is always feasible and desirable to remove code clones via refactoring.

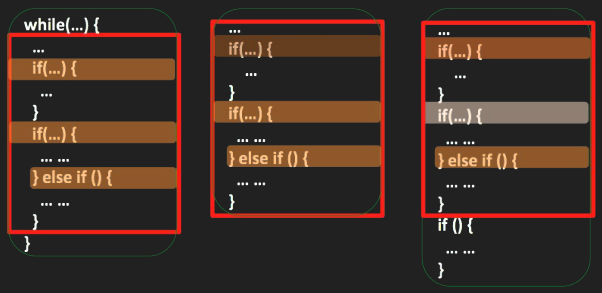
RASE: Exploit systematic edits for clone removal refactoring

- Input: Systematic edits
- Step 1: Scope refactoring region and analyze variations
- Step 2: Create and apply an executable refactoring plan



Step 1: Scope code to refactor

- Refactor the maximum syntactically valid contiguous code clones enclosing edits



Step 2: Create and apply a refactoring plan

- Challenges to extract common code

- Type variations
- Method variations
- Variable/Expression variations
- Multiple variables to return
- Non-local jump statements

Step 2: Create and apply a refactoring plan

- Challenges to extract common code — Refactoring

- Type variations	→	- Parameterize type
- Method variations	→	- Form template method
- Variable/Expression variations	→	- Add parameter
- Multiple variables to return	→	- Introduce return object
- Non-local jump statements	→	- Introduce exit label

Type Variations - Parameterize Type

```
public void mA(IC c) {
  …
  Insert e = getEdit(c);
  …
}
```
Code to extract

```
public void mB(RC c) {
  …
  Remove e = getEdit(c);
  …
}
```
Code to extract

```
class C<T0,T1>{
  public void extractMethod(T1 c){
    …
    T0 e = getEdit(c);
    …
  }
}
public void mA(IC c){
  new C<Insert,IC>().extractMethod(c);
}
public void mB(RC c){
  new C<Remove, RC>.extractMethod(c);
}
```


Method Variations - Form Template Methods

```
public void add() {
  …
  input.addCompareInput()
  …
}
```
Code to extract

```
public void remove() {
  …
  input.removeCompareInput()
  …
}
```
Code to extract

```
abstract class Template{
  public void extractMethod(…){
    …
    m(input);
    …
  }
  public abstract void m(Input input);
}
class Add extends Template {
  public void m(Input input){
    input.addCompareInput();
  }}
class Remove extends Template {
  public void m(Input input) {
    input.removeCompareInput();
  }
}
public void add(){
  new Add().extractMethod(…);
}
public void remove() {
  new Remove().extractMethod(…);
}
```

Multiple Variables to Return - Introduce Return Objects

```
public void foo() {
…
String str1= … ;
String str2= … ;
…
System.out.println
(str1+str2);
```

Code to extract

```
public RetObj {
 public String str1;
 public String str2;...
}
public RetObj extractMethod(){
…
return new RetObj(str1,str2);
}

public void foo() {
…
RetObj retObj=extractMethod(…);
String str1 = retObj.str1;
String str2 = retObj.str2;
…
System.out.println (str1+str2);
}
```


Non-Local Jump Statement - Introduce Exit Labels

```
public void bar() {
 while (!stack.isEmpty()) {
…
 elem = stack.pop();
 if (elem==null)
   continue;
 if (elem.equals(known))
   break;
 push(elem.next());
}
}
```

Code to extract

```
enum Label{CONTINUE,BREAK.FALL}
public Label extractMethod(…){
…
 elem = stack.pop();
 if (elem==null)
  return Label.CONTINUE;
 if (elem.equals(known))
  return Label.BREAK;
return Label.FALL;
}
public void bar() {
 while (!stack.isEmpty()) {
  Label l = extractMethod(…);
  if (l.equals(Label.CONTINUE))
   continue;
  if (l.equalis(Label.BREAK))
   break;
 }}
```


Test Suite

- 56 similarly changed method pairs from
  - org.eclipse.compare
  - org.eclipse.core.runtime
  - org.debug.core
  - jEdit.core
  - jEdit
- 30 similarly changed method groups from
  - Elasticsearch
  - jfreechart

Q1: Is clone removal refactoring feasible?

	ID	edits	types	Δcode
Pair	2	15	E, A	-1
	9	77	E, R	-7
	22	285	E, F	-47
	29	56	E, L, R	4
Group	1	137	E, A, F, T	-7
	5	36	E, T	-6
	8	44	E, A, F	-4
	29	211	E	-149

RASE refactors:

- 30 of 56 method pairs
- 20 of 30 method groups

A: add parameter

T: parameterize type

R: introduce return object

L: introduce exit label

F: form template method

E: extract method

Q2: Why does refactoring fail?

Reason	#method pairs	#method groups
Limited language support for generic types, e.g., v instanceof $T	7	2
Unmovable methods,e.g.,super()	5	0
No edited statements found	8	2
No common code extracted	6	6

42% of systematic edits cannot be removed via clone removal refactoring.

Q3: Is clone removal refactoring desirable?

- Average duration of version history: 1.3 years

		Feasible	Infeasible
Refactored		5	0
Un-refactored	Co-evolved	4	7
	Divergent	7	10
	Unchanged	34	19

6% of systematic edits are refactored by developers.

RASE refactors all of them.

"We don't typically refactor unless we have to change the code for some bug fix or new feature. "

Conclusion

- Automatic clone removal refactoring cannot obviate systematic editing
- Both clone removal refactoring and automated systematic editing are needed and they are complementary
- Determining refactoring desirability remains as further work