Many code edits have already been performed in the past...

```csharp
while (receiver.CSharpKind() == SyntaxKind.ParenthesizedExpression)
    while (receiver.IsKind(SyntaxKind.ParenthesizedExpression))
        foreach (var m in modifiers) {
            if (m.CSharpKind() == modifier)
                return true;
        }
```

```csharp
while (receiver.CSharpKind() == SyntaxKind.ParenthesizedExpression)
    while (receiver.IsKind(SyntaxKind.ParenthesizedExpression))
        foreach (var m in modifiers) {
            if (m.IsKind(modifier))
                return true;
        }
```

Many code edits have already been performed in the past...

By different developers in different programs (>100 students)

```csharp
def product(n, term):
    total, k = 1, 1
    while k <= n:
        total, k = total * k, k + 1
    return total
```

```csharp
def product(n, term):
    total, k = 1, 1
    while k <= n:
        total, k = total * term(k), k + 1
    return total
```

Many code edits have already been performed in the past...

By different developers in different projects (3 projects)

```csharp
if args[i].equals("--launchdiag") {
    if ("--launchdiag".equals(args[i])) {
        if ("-main".equals(args[i])) {
            if ("-main".equals(args[i])) {
                return true;
            }
        }
    }
```

```csharp
if args[i].equals("--launchdiag") {
    if ("--launchdiag".equals(args[i])) {
        if ("-main".equals(args[i])) {
            if ("-main".equals(args[i])) {
                return true;
            }
        }
    }
```

Similar edits share the same structure but with different expressions

```csharp
if numerator == 1:
    return 1
```

```csharp
if numerator == 1:
    return 1
```

Many code edits have already been performed in the past...

By different developers in different programs (>100 students)
IDEs and static analysis tools help detecting and/or automating repetitive edits.

Auto Grader [PLDI’13] fixes common faults in programming assignments.

These tools rely on predefined classes of rules, which are hard to extend.

Refazer
Learning syntactic program transformations from examples

Refazer
Prose framework

Refazer
Prose framework
Refazer
Prose framework

Divide-and-conquer algorithm
Backpropagation functions

Transformation 1
Transformation 2
Transformation 3
Transformation n

DSL

def product(n, term):
    if (n == 1):
        return 1
    return product(n-1, term)*n + return product(n-1, term)*term(n)

def product(n, term):
    total, k = 1, 1
    while k<=n:
        total = total * k
        total = total * term(k)
        k = k + 1
    return total
Divide-and-conquer approach

Backpropagation

Disjunctive Backpropagation

Divide-and-conquer approach
Transformations\((\text{rule}_1, \ldots, \text{rule}_n)\)

\[
\text{Map}(\lambda x \rightarrow \text{operation}, \text{locations})
\]

\[
\text{operation}\ |
\text{locations}
\]

\[
k
\]

\[
\text{Context}(\text{pattern}, \text{path})
\]

\[
\text{pattern}
\]

\[
k
\]

\[
U^\cup
\]

Disjunctive Backpropagation

\[
total \times k
\]

\[
U
\]

Example

\[
\text{Example}\ |
\text{operation}, \text{locations}
\]

\[
k
\]

\[
\text{paths}
\]

\[
\text{pattern}
\]

\[
k
\]

Update\(x, \text{term}(\text{r}_1)\) where \(\text{r}_1 = \text{Reference}(x, 1)\)

Update\(x, \text{term}(k)\)

Ranking functions

• Favor \text{Reference} over \text{ConstNode}

• Favor patterns with non-root paths

Evaluation
Case studies

We investigate whether Refazer can be used to perform repetitive edit in C# projects

We investigate whether Refazer can be used to fix bugs in students' programming assignments

Research questions

RQ1 - Can Refazer learn transformations to automate repetitive edits to large C# projects?

RQ2 - How often can transformations learned from student code edits be used to fix incorrect code of other students who are solving the same programming assignment?

RQ3 - How often can transformations learned from student code edits be used to fix incorrect code of other students who are solving a different programming assignment?

Case study 1

RQ1: Can Refazer learn transformations to automate repetitive edits to large C# projects?

<table>
<thead>
<tr>
<th>Projects</th>
<th>LOCs</th>
<th>Commits</th>
<th>Analyzed</th>
<th>Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roslyn</td>
<td>1,552,827</td>
<td>5,990</td>
<td>212</td>
<td>27</td>
</tr>
<tr>
<td>Entity Framework</td>
<td>766,321</td>
<td>1,932</td>
<td>90</td>
<td>12</td>
</tr>
<tr>
<td>NuGet</td>
<td>149,779</td>
<td>4,746</td>
<td>102</td>
<td>17</td>
</tr>
<tr>
<td>Total</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Case study 1

RQ1: Can Refazer learn transformations to automate repetitive edits to large C# projects?

<table>
<thead>
<tr>
<th>Projects</th>
<th>LOCs</th>
<th>Commits</th>
<th>Analyzed</th>
<th>Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roslyn</td>
<td>1,552,827</td>
<td>5,990</td>
<td>212</td>
<td>27</td>
</tr>
<tr>
<td>Entity Framework</td>
<td>766,321</td>
<td>1,932</td>
<td>90</td>
<td>12</td>
</tr>
<tr>
<td>NuGet</td>
<td>149,779</td>
<td>4,746</td>
<td>102</td>
<td>17</td>
</tr>
<tr>
<td>Total</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental setup

Two random examples → Refazer → One more random example → Stop

Results

In 71% of the scenarios, the synthesized transformations applied the same edits applied by developers.

In 13% of the scenarios, it correctly applied edits missed by developers.

In 16% of the scenarios, it incorrectly applied additional edits or do not synthesize a transformation.

Results

On average, 2.9 examples were required for learning the transformations.

Case study 2

RQ2: How often can transformations learned from student code edits be used to fix incorrect code of other students in the same programming assignment?
We used the last pair of submissions as examples of bug fixes

Benchmark

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Students</th>
<th>Incorrect submissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product product of the first n terms</td>
<td>549</td>
<td>3,218</td>
</tr>
<tr>
<td>Accumulate fold-left of the first n terms</td>
<td>668</td>
<td>6,410</td>
</tr>
<tr>
<td>Repeated function composition, depth n</td>
<td>720</td>
<td>9,924</td>
</tr>
<tr>
<td>G [ G(x) = \sum_{i} (x - f(x - 1)) ]</td>
<td>379</td>
<td>2,229</td>
</tr>
</tbody>
</table>

Results

Learned transformations are useful within the same programming assignments

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Incorrect Submissions</th>
<th>Batch</th>
<th>students</th>
<th>fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>5.3 (8.2)</td>
<td>501 (97%)</td>
<td>3.57 (6.1)</td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>8.9 (10.1)</td>
<td>608 (95%)</td>
<td>5.4 (7.9)</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>12.7 (15.3)</td>
<td>580 (81%)</td>
<td>8 (10.3)</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>5.5 (8.4)</td>
<td>319 (84%)</td>
<td>3.4 (1.7)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8.7 (12)</td>
<td>2,058 (97%)</td>
<td>8.2 (10.3)</td>
<td></td>
</tr>
</tbody>
</table>

Results

Learned transformations are useful within the same programming assignments

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Incorrect Submissions</th>
<th>Batch</th>
<th>students</th>
<th>fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>5.3 (8.2)</td>
<td>501 (97%)</td>
<td>3.57 (6.1)</td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>8.9 (10.1)</td>
<td>608 (95%)</td>
<td>5.4 (7.9)</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>12.7 (15.3)</td>
<td>580 (81%)</td>
<td>8 (10.3)</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>5.5 (8.4)</td>
<td>319 (84%)</td>
<td>3.4 (1.7)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8.7 (12)</td>
<td>2,058 (97%)</td>
<td>8.2 (10.3)</td>
<td></td>
</tr>
</tbody>
</table>

Results

Learned transformations are useful within the same programming assignments

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Incorrect Submissions</th>
<th>Incremental</th>
<th>students</th>
<th>fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>5.3 (8.2)</td>
<td>247 (45%)</td>
<td>4.1 (6.7)</td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>8.9 (10.1)</td>
<td>253 (38%)</td>
<td>7.5 (9.8)</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>12.7 (15.3)</td>
<td>340 (47%)</td>
<td>9.6 (11.5)</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>5.5 (8.4)</td>
<td>174 (40%)</td>
<td>6.1 (7.1)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8.7 (12)</td>
<td>1,054 (44%)</td>
<td>6.8 (8.9)</td>
<td></td>
</tr>
</tbody>
</table>

Results

Learned transformations are useful within the same programming assignments

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Incorrect Submissions</th>
<th>Incremental</th>
<th>students</th>
<th>fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>5.3 (8.2)</td>
<td>247 (45%)</td>
<td>4.1 (6.7)</td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>8.9 (10.1)</td>
<td>253 (38%)</td>
<td>7.5 (9.8)</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>12.7 (15.3)</td>
<td>340 (47%)</td>
<td>9.6 (11.5)</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>5.5 (8.4)</td>
<td>174 (40%)</td>
<td>6.1 (7.1)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8.7 (12)</td>
<td>1,054 (44%)</td>
<td>6.8 (8.9)</td>
<td></td>
</tr>
</tbody>
</table>
Analysis of the first time Refazer can fix a student submission for the 50 students with most attempts

Results

RQ3 - How often can transformations learned from student code edits be used to fix incorrect code of other students who are solving a different programming assignment?

Most learned transformations are not useful among different programming assignments

<table>
<thead>
<tr>
<th>Original Assignment</th>
<th>Target Assignment</th>
<th>Helped students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>Accumulate</td>
<td>34 out of 400 (11%)</td>
</tr>
<tr>
<td>Product</td>
<td>Repeated</td>
<td>31 out of 379 (9%)</td>
</tr>
</tbody>
</table>

Expressivity

Add complete statements

Multiple line edits

Revisar

Repositories

Extracting AST edits

Clustering edits

Learning transformations

Transformation inspection

Revisar

Repositories

Extracting AST edits

Clustering edits

Learning transformations

Transformation inspection
Extracting edits

Clustering edits

Learning transformations

Research questions

RQ1: Can REVISAR be used to discover common quick fixes?

RQII: Are developers interested in applying quick fixes discovered by REVISAR?
RQ1: Can REVISAR be used to discover common quick fixes?

<table>
<thead>
<tr>
<th>Project</th>
<th>Edits</th>
<th>LOCs</th>
<th>Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hive</td>
<td>94,701</td>
<td>1,119,579</td>
<td>11,467</td>
</tr>
<tr>
<td>Ant</td>
<td>49,600</td>
<td>137,203</td>
<td>13,790</td>
</tr>
<tr>
<td>Guava</td>
<td>28,784</td>
<td>325,902</td>
<td>6,633</td>
</tr>
<tr>
<td>Drill</td>
<td>26,173</td>
<td>350,756</td>
<td>2,902</td>
</tr>
<tr>
<td>ExoPlayer</td>
<td>20,726</td>
<td>85,305</td>
<td>3,875</td>
</tr>
<tr>
<td>Giraph</td>
<td>8,836</td>
<td>99,274</td>
<td>1,062</td>
</tr>
<tr>
<td>Guan</td>
<td>4,435</td>
<td>24,753</td>
<td>1,393</td>
</tr>
<tr>
<td>Truth</td>
<td>3,857</td>
<td>27,427</td>
<td>1,137</td>
</tr>
<tr>
<td>Error/Prone</td>
<td>3,200</td>
<td>116,023</td>
<td>2,854</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>240,612</td>
<td>2,386,222</td>
<td>33,113</td>
</tr>
</tbody>
</table>

RQ1: Can REVISAR be used to discover common quick fixes?

<table>
<thead>
<tr>
<th>Project</th>
<th>Edits</th>
<th>LOCs</th>
<th>Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hive</td>
<td>94,701</td>
<td>1,119,579</td>
<td>11,467</td>
</tr>
<tr>
<td>Ant</td>
<td>49,600</td>
<td>137,203</td>
<td>13,790</td>
</tr>
<tr>
<td>Guava</td>
<td>28,784</td>
<td>325,902</td>
<td>6,633</td>
</tr>
<tr>
<td>Drill</td>
<td>26,173</td>
<td>350,756</td>
<td>2,902</td>
</tr>
<tr>
<td>ExoPlayer</td>
<td>20,726</td>
<td>85,305</td>
<td>3,875</td>
</tr>
<tr>
<td>Giraph</td>
<td>8,836</td>
<td>99,274</td>
<td>1,062</td>
</tr>
<tr>
<td>Guan</td>
<td>4,435</td>
<td>24,753</td>
<td>1,393</td>
</tr>
<tr>
<td>Truth</td>
<td>3,857</td>
<td>27,427</td>
<td>1,137</td>
</tr>
<tr>
<td>Error/Prone</td>
<td>3,200</td>
<td>116,023</td>
<td>2,854</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>240,612</td>
<td>2,386,222</td>
<td>33,113</td>
</tr>
</tbody>
</table>

RQ1: Can REVISAR be used to discover common quick fixes?

<table>
<thead>
<tr>
<th>Project</th>
<th>Edits</th>
<th>LOCs</th>
<th>Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hive</td>
<td>94,701</td>
<td>1,119,579</td>
<td>11,467</td>
</tr>
<tr>
<td>Ant</td>
<td>49,600</td>
<td>137,203</td>
<td>13,790</td>
</tr>
<tr>
<td>Guava</td>
<td>28,784</td>
<td>325,902</td>
<td>6,633</td>
</tr>
<tr>
<td>Drill</td>
<td>26,173</td>
<td>350,756</td>
<td>2,902</td>
</tr>
<tr>
<td>ExoPlayer</td>
<td>20,726</td>
<td>85,305</td>
<td>3,875</td>
</tr>
<tr>
<td>Giraph</td>
<td>8,836</td>
<td>99,274</td>
<td>1,062</td>
</tr>
<tr>
<td>Guan</td>
<td>4,435</td>
<td>24,753</td>
<td>1,393</td>
</tr>
<tr>
<td>Truth</td>
<td>3,857</td>
<td>27,427</td>
<td>1,137</td>
</tr>
<tr>
<td>Error/Prone</td>
<td>3,200</td>
<td>116,023</td>
<td>2,854</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>240,612</td>
<td>2,386,222</td>
<td>33,113</td>
</tr>
</tbody>
</table>

RQ1: Can REVISAR be used to discover common quick fixes?

<table>
<thead>
<tr>
<th>Project</th>
<th>Edits</th>
<th>LOCs</th>
<th>Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hive</td>
<td>94,701</td>
<td>1,119,579</td>
<td>11,467</td>
</tr>
<tr>
<td>Ant</td>
<td>49,600</td>
<td>137,203</td>
<td>13,790</td>
</tr>
<tr>
<td>Guava</td>
<td>28,784</td>
<td>325,902</td>
<td>6,633</td>
</tr>
<tr>
<td>Drill</td>
<td>26,173</td>
<td>350,756</td>
<td>2,902</td>
</tr>
<tr>
<td>ExoPlayer</td>
<td>20,726</td>
<td>85,305</td>
<td>3,875</td>
</tr>
<tr>
<td>Giraph</td>
<td>8,836</td>
<td>99,274</td>
<td>1,062</td>
</tr>
<tr>
<td>Guan</td>
<td>4,435</td>
<td>24,753</td>
<td>1,393</td>
</tr>
<tr>
<td>Truth</td>
<td>3,857</td>
<td>27,427</td>
<td>1,137</td>
</tr>
<tr>
<td>Error/Prone</td>
<td>3,200</td>
<td>116,023</td>
<td>2,854</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>240,612</td>
<td>2,386,222</td>
<td>33,113</td>
</tr>
</tbody>
</table>

Quick Fix 01:
```
StringBuffer sb = new StringBuffer();
sb.append("a");
```

Quick Fix 02:
```
void bar(String s) {
    if (s.equals("str"))
        // ...}
```

Quick Fix 03:
```
void bar(String p, byte[] content) throws IOException {
    FileOutputStream os = new FileOutputStream(p);
    // ...
```

Quick Fix 04:
```
Integer a = new Integer(1);
```
RQ1: Can REVISAR be used to discover common quick fixes?

Quick Fix 05:
```java
List foo = getList();
if (foo.size() == 0){
    //...
```

Quick Fix 06:
```java
StringBuffer a = new StringBuffer();
StringBuffer a = new StringBuffer();
```

Quick Fix 07:
```java
List<String> a = new ArrayList<String>();
List<String> a = new ArrayList<String>();
```

Quick Fix 08:
```java
List<String> a = new ArrayList();
List<String> a = new ArrayList();
```

Quick Fix 09:
```java
String a = "a";
final String a = "a";
```

Quick Fix 10:
```java
private String path;
private Path path;
```

RQII: Are developers interested in applying quick fixes discovered by REVISAR?

Results

RQ1: Can REVISAR be used to discover common quick fixes?

Results

For most quick fixes (90%), programmers agree with the suggested edits.
RQII: Are developers interested in applying quick fixes discovered by REVISAR?

Results

Experts and informed programmers tend to prefer better code patterns compared to non-experts and uninformed programmers.

Results

Programmers who use more tool support to detect and apply code patterns tend to prefer better code patterns compared to non-experts and uninformed programmers.

Results

Programmers may favor readability of their original code over other benefits provided by edit patterns.

Results

Static analysis tools can teach programmers to recognize good code patterns.

Results

Some programmers prefer some code patterns for different reasons than those the patterns are designed for.
Programmers found 10/20 (50%) of our pull requests useful and accepted them.

Programmers rejected 4/20 (20%) of our pull requests, but they showed support for the edit patterns we proposed.
Results

For 3/20 (15%) of our pull requests, programmers supported our edits, but the pull requests are still open.

Related work

Refazer x LASE

LADE focuses on abstracting names of variables, methods and types in the same statement kinds

We abstract entire expressions within different statements

Lase learns just one transformation, We learn a set of transformations.

return product(n-1, term) * n

total = total * k
RQ answers

REFAZER learns the intended program transformation in 84% of the cases and using only 2.9 examples on average. REFAZER helped to fix incorrect submissions for 87% of the students. Using transformations learned from other assignments, we fixed submissions for 7-24% of the students.

Benchmark

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of locations</td>
<td>3 to 60</td>
</tr>
<tr>
<td>Median number of locations</td>
<td>5</td>
</tr>
<tr>
<td>Multiple files</td>
<td>25%</td>
</tr>
<tr>
<td>Similar but not identical edits</td>
<td>70%</td>
</tr>
</tbody>
</table>
Repository

Repository → Commits → Repetitive edits