# Design of Repair Operators for Automated Program Repair

Shin Hwei Tan

National University of Singapore

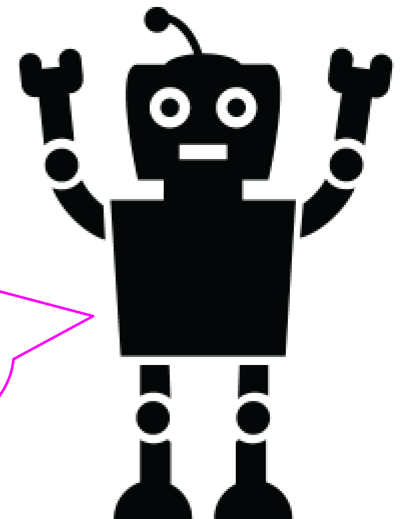
# What is automated program repair?

Given a failing Test *T* , buggy program *P*

1.Fault localization – Where to fix?

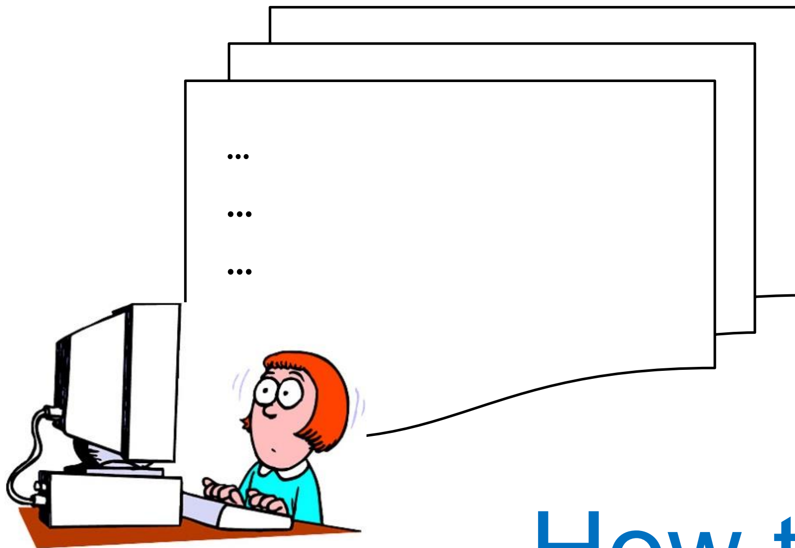
2. Patch Generation using repair operators – How to fix?

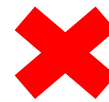
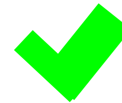
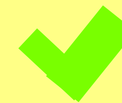
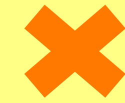
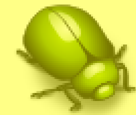
3. Patch Validation – Are all tests passing?



# How to extract useful repair operators?

| | *GenProg* [ICSE '12] | *relifix* [ICSE '15] |
|---|---|---|
| **Search** | • Genetic Programming | • Random Local Search |
| **Operators** | Mutations & crossovers | Contextual Operators |
| **Extracted from** | Genetic Operators | Human Repair of Software Regression & investigation of types of regressions |

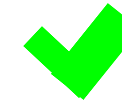



| | | | |
|---|---|---|---|
| Test 1 |  | | |
| Test 2 |  | | |
| Test 3 |  |  |  |

Regression!

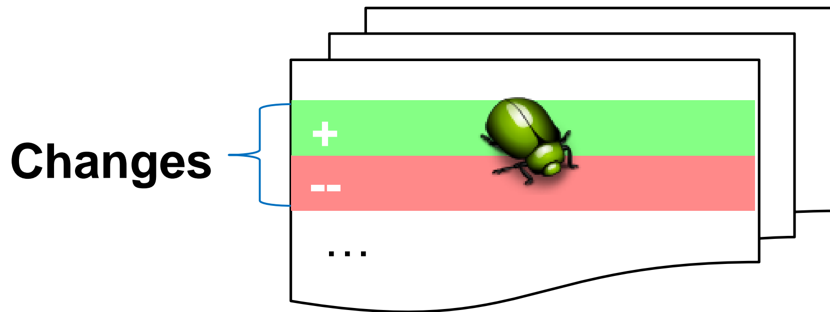
How to repair?

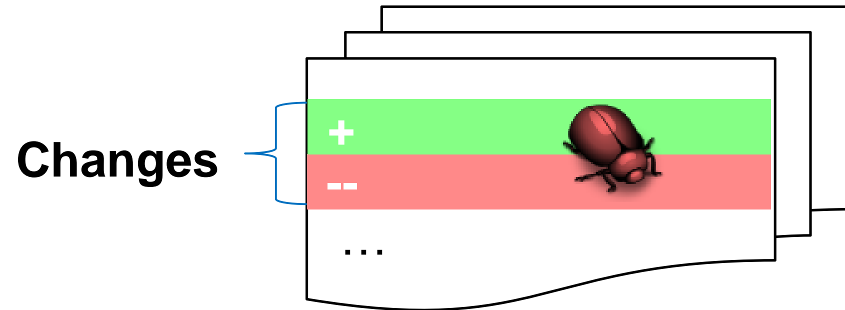
```
+...
while (out > line)
   out;
-...
```

| | |
|---|---|
| Test 1 | |
| Test 2 | |
| Test 3 |  |

Regression Fixed!

# Types of Software regressions




- Changes break existing functionality
Repair: Roll back to previous version

- Changes unmasks existing bug
Repair: Re-mask problematic change

- formulate the software regression repair problem as problem of reconciling problematic changes

- Changes introduce bug in other unchanged parts
Repair: Re-mask problematic change

# Most frequently used Operators in Human Repair

| Operator | Operator Type | Count |
| --- | --- | --- |
| Add condition | Non-contextual | 27 |
| Add statement | Non-contextual | 21 |
| Use changed expression as input for other operator | Contextual | 13 |
| Revert to previous statement | Contextual | 11 |
| Replace with new expression | Non-contextual | 13 |
| Remove incorrectly added statement | Contextual | 9 |
| Change type | Non-contextual | 5 |
| Add method | Non-contextual | 5 |
| Add parameter | Non-contextual | 4 |
| Add local variable | Non-contextual | 3 |
| Swap changed statement with neighbouring statement | Contextual | 2 |
| Negate added condition | Contextual | 1 |
| Convert statement to condition variable statement | Contextual | 1 |
| Add field | Non-contextual | 1 |
| **Total** | 6 Contextuals | 116 |

# Contextual Operators

- Use changed expression as input for other operator

```
- if (((f = lookup_file (p)) != 0 && f->is_target)
+ if (((f = lookup_file (p)) != 0 && (f->is_target || intermed_ok))
```

- Revert to previous statement

```
- /* Removing this loop will fix Savannah bug #16670:
- do we want to? */
- while ( out > line && isblank (( unsigned char ) out[-1]))
- --out ;
```

# Experimental Results

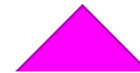
- Evaluated on 7 open source projects
  - *relifix* repairs 23 bugs, *GenProg* only fixes five bugs
  - *relifix* is less likely to introduce new regressions than *GenProg*
- Related questions:
  - How about regression in automatically generated patches?
  - How to avoid Regression Introducing Patches?

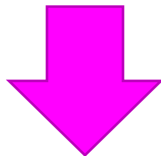
# Search-Based Program Repair

Final Patch

Search-Based Repair Tools

Tests Fail

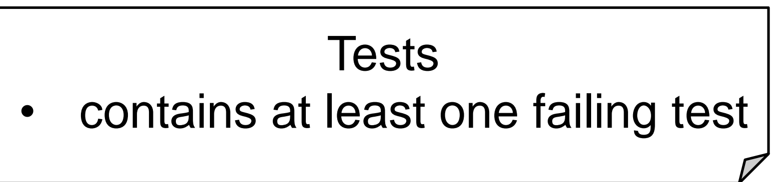




Patch Generation

How do the tests look like?

Candidate Patches




How do the patches look like?

ation

# Search-Based Program Repair

Test Script

```
$command $argument1 $argument2
RETVAL=$?
[ $RETVAL -eq 0 ] && echo Success
[ $RETVAL -ne 0 ] && echo Failure
```

Check exit status of command
Non-zero exit status denotes test failure

```
- exit(-2);
```

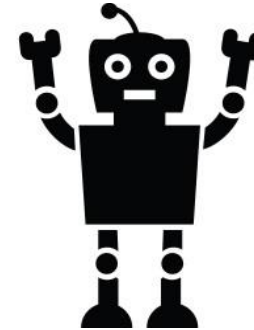
Tests

Patch Evaluation

# Repair patterns from human patches

**Human patches**

**Automatic Program Repair**



***Anti-patterns***

Set of generic forbidden transformations
that can be enforced on top of any search-based repair tool.

# Problem: Weak Oracle

Failing Test Script

```
$command $argument1 $argument2
RETVAL=$?
[ $RETVAL -eq 0 ] && echo Success
[ $RETVAL -ne 0 ] && echo Failure
```

Test outcome determined by exit status

- Statements like exit call/assertions serve as test proxies
- Test proxies should *not be* randomly manipulated

***A1: Anti-delete CFG exit node***

✖ Remove return statements, exit calls, functions with the word “error”, assertions.

```
static void BadPPM(char* file) {
    fprintf(stderr, "%s: Not a PPM file.\n", file);
-   exit(-2);
}
```

# Problem: Inadequate Test Coverage

- Repair tools allow removal of code as long as all test passes
- Statements are mistakenly considered as redundant code
- *Anti-patterns:*
  - *A2: Anti-delete Control Statement*
  - *A3: Anti-delete Single-statement CFG*
  - *A4: Anti-delete Set-Before-If*

***A2: Anti-delete Control Statement***

✖Remove control statements (e.g., if-statements, switch-statements, loops).

```
  call_result = call_user_function_ex(...);
- if (call_result == SUCCESS && ...) {
-   if (SUCCESS == statbuf_from_array(...))
-      ret = 0;
- } else if (call_result == FAILURE) {…
```

# Problem: Non-termination

- Automatically generated patches may incorrectly removes loop update
- ➢ Cause infinite loop

***A5:Anti-delete Loop-Counter Update***

✖Remove assignment statement *A* inside loop *L* if:

$$\{Var\ in\ Termination\ Condition\ of\ L\} \cap \{Var\ in\ LHS\ of\ assignment\ A\} = \emptyset$$

```
while( x> 5)
- x++;
```

# Problem: Trivial Patch

- Trivial patch – patch that insert return-statements based on expected output

Ex:
```
+if(test1)
+ return out1
```

***A6: Anti-append Early Exit***

✖Insert return/goto statement at any location except for after the last statement in a CFG node.

```
+ if ((type != 0))
+   return;
zend_error((1<<3L),"Uninitialized string offset:",...);
```

# Problem: Functionality Removal

- Removes functionality by inserting T/F

***A7: Anti-append Trivial Conditions***

✖ Insert trivial condition.

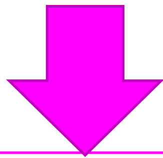
❖A condition is trivial if and only if it is:

1) True/False Constant
2) Tautology/Contradiction in expression (e.g., if(x || y || !y))
3) Static analysis (e.g., if(x || y != 0), y is initialized)

```
- if ((fmap[j].key != format->ptr[i + 1]))
+ if ((fmap[j].key != format->ptr[i + 1]) && !(1))
      continue;
```

# Integrating *Anti-patterns*

Search-Based Repair Tools

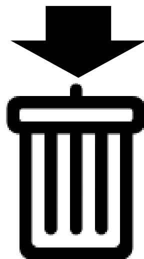


Patch Generation

Candidate Patches

Is Anti-pattern?

YES



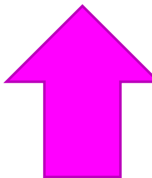
NO

Patch Evaluation

Tests
- contains at least one failing test

Tests Fail

All Tests Pass



Final Patch

# How could anti-pattern helps?

- Evaluated on 12 open source projects
  - Enforcing anti-patterns leads to patches with better fix localization and delete less functionality.
  - Tools integrated with anti-patterns generate patches faster due to repair space reduction.
- Related questions:
  - Are existing program repair techniques effective in generating patches?
    - Anti-patterns reveal many problems in automatically generated patches
  - How about anti-patterns for repair operators? Could we get rid of repair operators that are ineffective?

# *Design of Repair Operators: Codeflaws*

Programming Competition Benchmark for Objective Evaluation of Program Repair

# Codeflaws Benchmark

- Obtained from Codeforces online database
- Diverse types of defects
  - 40 defects types
- Large number of defects
  - 4085 real defects
- Large number of programs
  - 7945 programs
- Large Held-out test suite for patch validation
  - 5-350 tests, Average: 40
- Non-trivial programs (algorithmically complex)
- Support large-scale controlled Experiments
- https://codeflaws.github.io/

# Frequency and Effectiveness of Repair Operators

| Repair Operator | GenProg | | SPR | | Prophet | | Angelix | |
|---|---|---|---|---|---|---|---|---|
| | Freq(%) | Eff(%) | Freq(%) | Eff(%) | Freq(%) | Eff(%) | Freq(%) | Eff(%) |
| Delete Statement | 17.53 | 41.22 | | | | | | |
| Insert Assignment | 17.39 | 38.46 | 5.77 | 43.10 | 4.80 | 39.51 | | |
| Insert If | 16.92 | 38.74 | 7.96 | 50.00 | | | 5.96 | 32.56 |
| Loosen /Tighten Condition | | | 54.53 | 22.35 | 46.06 | 19.95 | 3.12 | 4.44 |
| Variable Replacement | | | 8.51 | 56.73 | 6.46 | 29.36 | 19.42 | 0.36 |
| Relational Operator Replacement | | | | | | | 31.07 | 42.41 |

High frequency, Low Effectiveness

# Future Research

- Applications of Program Repair
  - Test-Driven Merging
    - Instead of using Longest Common Subsequence, use tests to drive merging of multiple programs
      - Provide additional guarantee that merged program pass all tests
- *Anti-patterns* beyond Program Repair
  - *Anti-patterns* as specification for guiding repair
  - *Anti-patterns* as selected “code smells”
  - Adapt *anti-patterns* to other search-based software engineering activities (e.g., specific code anti-patterns identifying energy hot-spots)