三篇论文

SCIENCE AND TECHNOLOGY

《Towards Automatically Generating Summary Comments for Java Methods》	2010	ASE
《The Strength of Random Search on Automated Program Repair》	2014	ICSE
《How Different Is It Between Machine-Generated and Developer-Provided Patches》	2019	EMSE

GRADUATION DEFENSE

Towards Automatically Generating Summary Comments for Java Methods

2010 ASE

Problem

Given a method signature and body statements for a method M, generate natural language text that summarizes the overall actions of M accurately, adequately, and concisely.

```java
public void hello(String name) {

    }
```

Challenge

1. Method names are inadequate summaries
2. Not all method body statements belong in a summary
3. Using names in the summary loses contextual information from the source code

Towards Automatically Generating Summary Comments for Java Methods

2010 ASE

An s_unit is a Java statement;
s_unit is the control flow expression with one of the if, while, for or switch keywords.

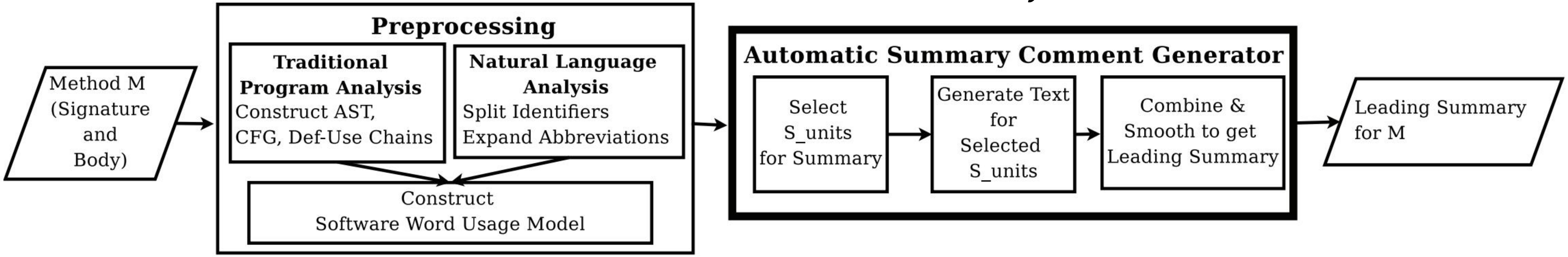


Figure 1: The Summary Comment Generation Process

action, theme, secondary arguments

list.add(Item i)	"add item to list."
void saveImage()	"save image"
Image savedImage()	"get saved image"

Given: f.getContentPane().add(view.getComponent(), CENTER)
We generate:
/* Add component of drawing view to content pane of frame*/

The Strength of Random Search on Automated Program Repair

2014 ICSE

GenProg & Par

Algorithm 1: The GenProg Algorithm

Input : Faulty program P
Input : Test cases T
Input : Mutation operator `Mutate`
Input : Crossover operator `Crossover`
Input : Full fitness predicate `FullFitness`
Input : Sampled fitness `SampleFit`
Input : Parameter $PopSize$
Output : One valid patch pt passing FullFitness

$C_{sub} \leftarrow$ `FaultLocalization`$(P,\ T)$;
$Pop \leftarrow$ `Mutate`$(PopSize, P,\ C_{sub})$;
repeat
    $Fitnesses \leftarrow$ `SampleFit`(Pop);
    $Parents \leftarrow$ `TournSelect`$(Pop,\ PopSize,\ Fitnesses)$;
    $Offsprings \leftarrow$ `Crossover`$(Parents,\ P)$;
    $Pop \leftarrow$ `Mutate`$(Parents, Offsprings)$;
until $\exists\ pt \in Pop.$ `FullFitness`$(pt) = Passed$;
return pt;

Algorithm 2: The RSRepair Algorithm

Input : Faulty program P
Input : Test cases T
Input : Mutation operator `Mutate`
Output : One valid patch pt

index $\leftarrow 0$;   `// Initialize the index value`
$\{n_0, t_1, t_2, \ldots, t_n\} \leftarrow T$;
$T \leftarrow \{(n_0, 1)(t_1, \text{index}), (t_2, \text{index}), \ldots, (t_n, \text{index})\}$;
$C_{sub} \leftarrow$ `FaultLocalization`$(P,\ T)$;
SuccessFlag $\leftarrow false$;
repeat
    $pt \leftarrow$ `Mutate`(P, C_{sub});
    for $i \leftarrow 0$ **to** n **do**
        //Check that whether pt is valid;
        $(t_{index}, \text{index}) \leftarrow$ `GetTestcase`$(T,\ i)$;
        if `PatchValidation`$(P, pt, t_{index}) \neq true$ **then**
            temp $\leftarrow (t_{index}, \text{index} + 1)$;
            $T \leftarrow$ `Prioritize`(T, temp);
            break;
        else if $i = n$ **then**
            SuccessFlag $\leftarrow true$;
        else
            continue;
        end
    end
until SuccessFlag $= true$;
return pt;

The Strength of Random Search on Automated Program Repair

2014 ICSE

RQ1: Whether can GenProg search a valid patch with fewer patch trials, compared to RSRepair?

RQ2: Does GenProg find a valid patch much faster than RSRepair in terms of requiring fewer Number of Test Case Executions (NTCE) within a successful repair process?

How Different Is It Between Machine-Generated and Developer-Provided Patches

2019 EMSE

Patch Overfitting

Traditionally, a patch is considered as correct if it passes all the test cases. However, the test suites in real world systems are usually weak such that most of the patches that pass all tests are incorrect.

Machine-generated Patches VS Developer-provided Patches

- Edit point--modified location
- Code modification--atomic operations : insertion, deletion,and replacement

SLSM

DLSM

SLDM

DLDM

How Different Is It Between Machine-Generated and Developer-Provided Patches

2019 EMSE

- RQ1: *How do machine-generated correct patches differ from developer-provided ones?*

APR-generated correct patches can be classified into four types based on their edit points and code modifications, while most of them (around 75%) are identical to their ground truth (i.e., SLSM patches).

- RQ2: How do different types of patches distribute?

- RQ3: Do APR tools tend to generate correct patches but different from the developer-provided ones for bugs with certain characteristics?

How Different Is It Between Machine-Generated and Developer-Provided Patches

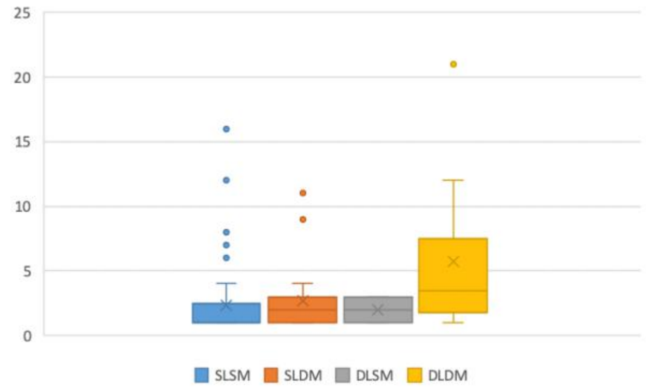
2019 EMSE

TABLE VIII. PATCH DISTRIBUTION FROM APR TECHNIQUES PERSPECTIVE

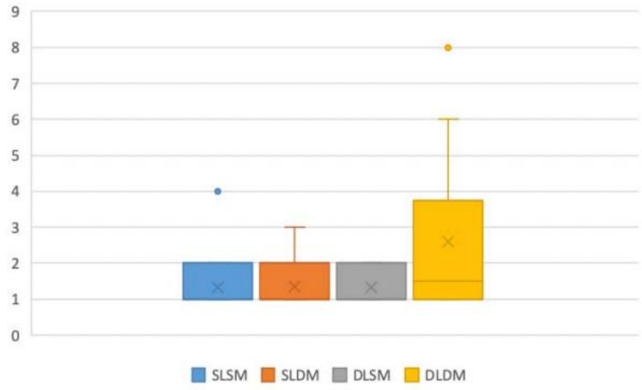
Technique	#SLSM	#SLDM	#DLSM	#DLDM	Total
CapGen	22	2	2	0	26
SimFix	23	6	0	4	33
AVATAR	18	8	0	0	26
Nopol	0	1	0	3	4
jGenProg	4	0	0	0	4
jKali	1	0	0	0	1
JAID	14	9	0	2	25
Elixir	22	4	0	0	26
ACS	16	0	0	1	17
ssFix	12	3	0	0	15
Total	132	33	2	10	177

How Different Is It Between Machine-Generated and Developer-Provided Patches

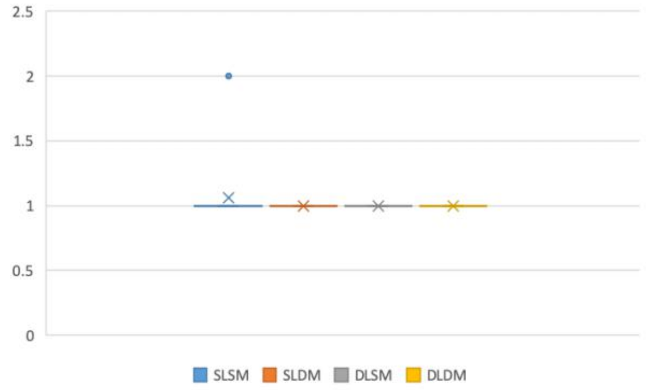
2019 EMSE



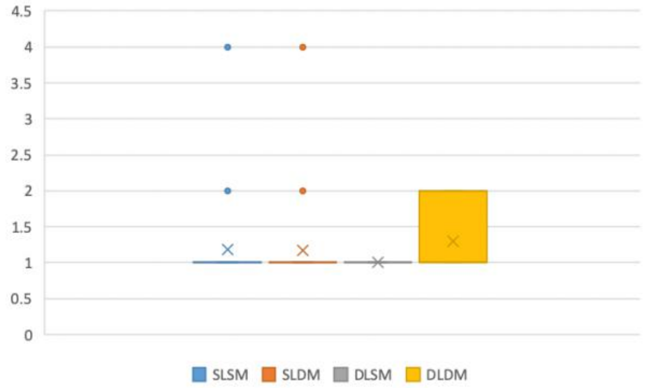
a) Patch size



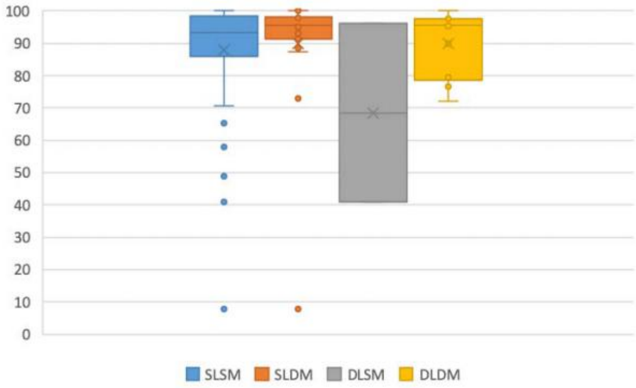
b) Number of chunks



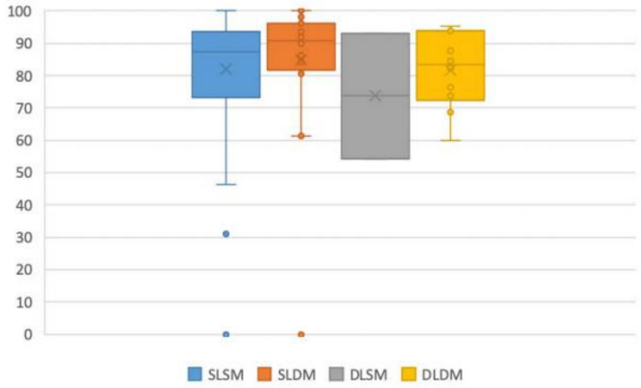
c) Number of modified files



d) Number of modified methods



e) Line coverage



f) Branch coverage

Fig. 4. Distributions of Bug Characteristics

南京航空航天大学
Nanjing University of Aeronautics and Astronautics

2022

THANK YOU

THNAK YOU

GRADUATION DEFENSE