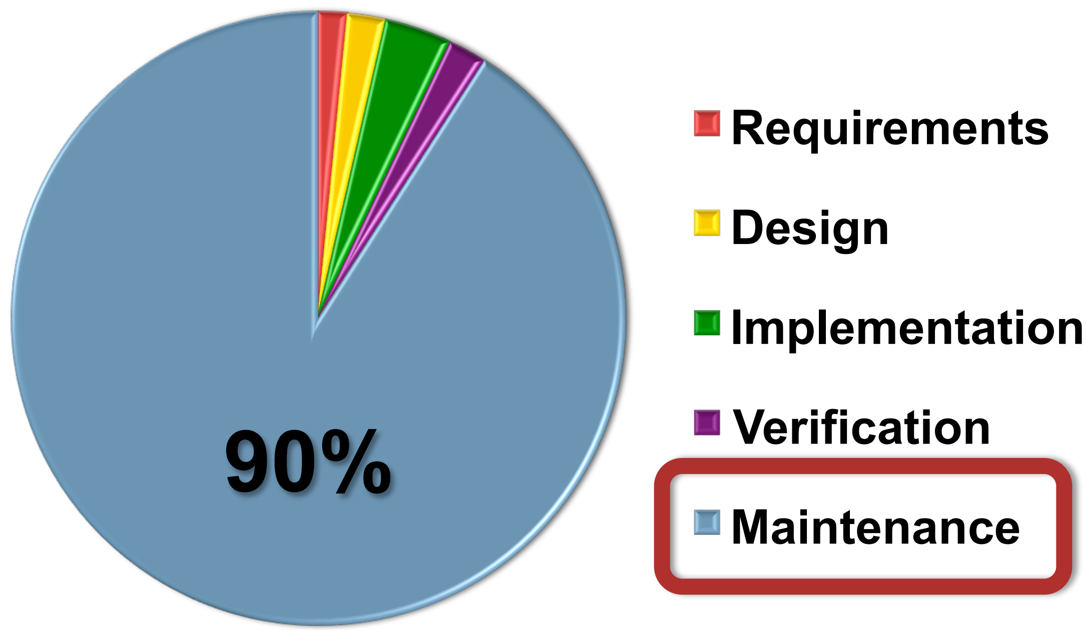
LEVERAGING LIGHTWEIGHT ANALYSES TO AID SOFTWARE MAINTENANCE

ZACHARY P. FRY

PHD PROPOSAL

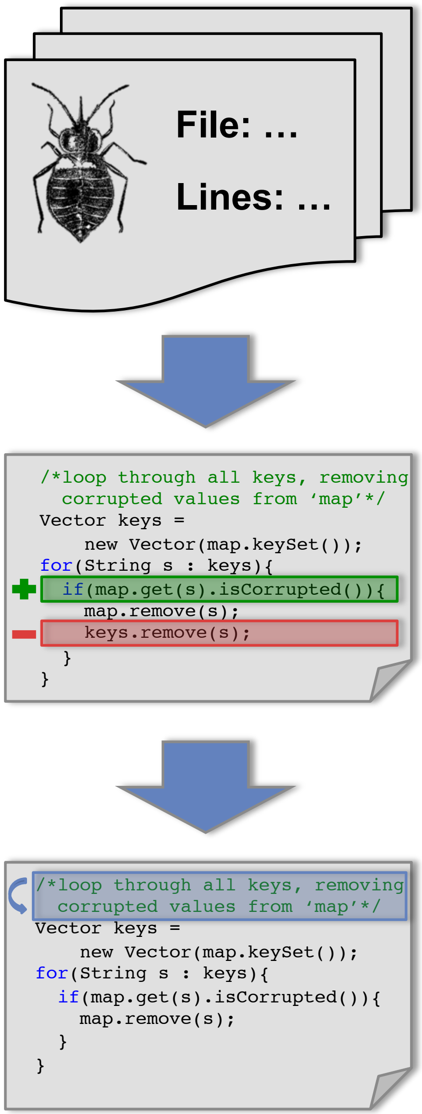
MAINTENANCE COSTS

For persistent systems, software maintenance can account for up to 90% of the software lifecycle costs.



R.C. Seacord, D. Plakosh, and G. A. Lewis. Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 2003.

KEY PARTS OF THE MAINTENANCE PROCESS



Bug Reporting

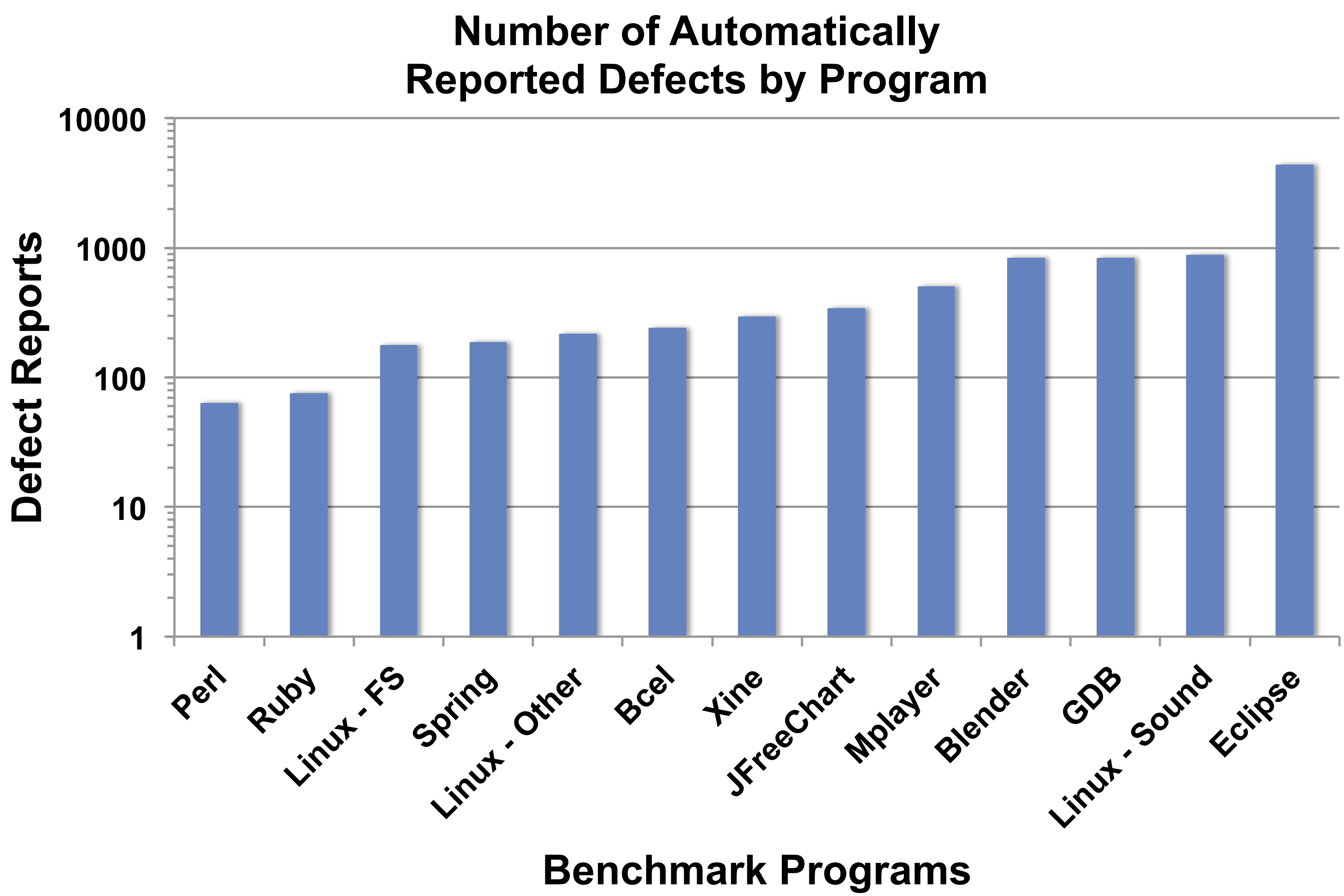
Bug Fixing

Update Documentation

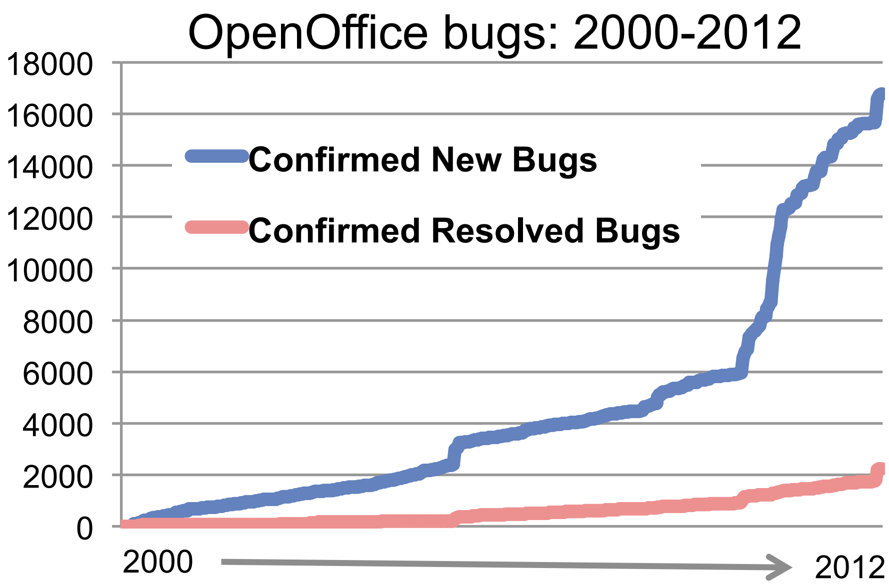
MAINTENANCE PROCESSES IN PRACTICE

- **Manual bug reporting is costly**
  - Reputation
  - Human effort
- **Automatic bug finders yield thousands of bugs, requiring verification and triage.**

MAINTENANCE PROCESSES IN PRACTICE



MAINTENANCE PROCESSES IN PRACTICE



Bug reports come in at an alarming rate, humans simply cannot triage and fix them all.

<u>Automatic program repair</u>

MAINTENANCE PROCESSES IN PRACTICE

Fixing bugs means lots of code changes.

Comments are often overlooked

- Out-of-date documentation

/* A reporter reporting the number of page faults since startup should have units UNITS_COUNT. */

→

/* The number of tabs currently open would have UNITS_COUNT. */

MAINTENANCE PROCESSES IN PRACTICE

Automated techniques have helped to facilitate the maintenance process.

However, the process remains costly.

Research question: Can we reduce the effort necessary for specific parts of the maintenance process, thereby reducing the overall cost?

PROPOSAL THESIS

By using <u>lightweight analyses</u> to extract and use <u>latent information</u> encoded by humans in software development artifacts we can <u>reduce the costs</u> of software maintenance by relieving bottlenecks in various stages throughout the process.

RESEARCH CONSIDERATIONS

Overall Goal:

- **Reduce maintenance costs**

Design Constraint:

- **Minimize additional human effort**
- **Ease of incremental adoption**

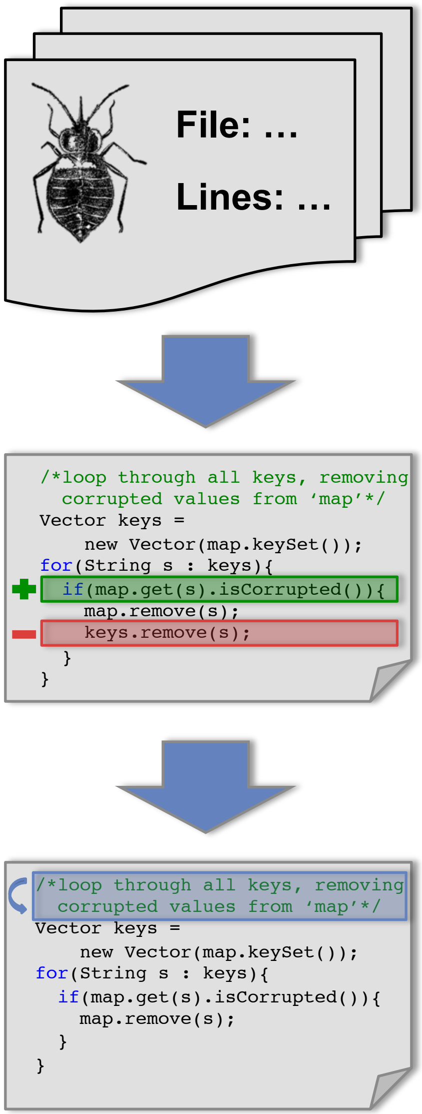
Overall Intuition:

- **Leverage information often overlooked by existing techniques**

THE REST OF THIS PRESENTATION

- **An overview of the proposed thrusts**
  - Clustering Duplicate Automatically-Generated Defect Reports
  - Improved Fitness Functions for Automatic Program Repair
  - Ensuring Documentation Consistency
- **Proposed research timeline**
- **Conclusion and Questions**

PROJECT OUTLINE



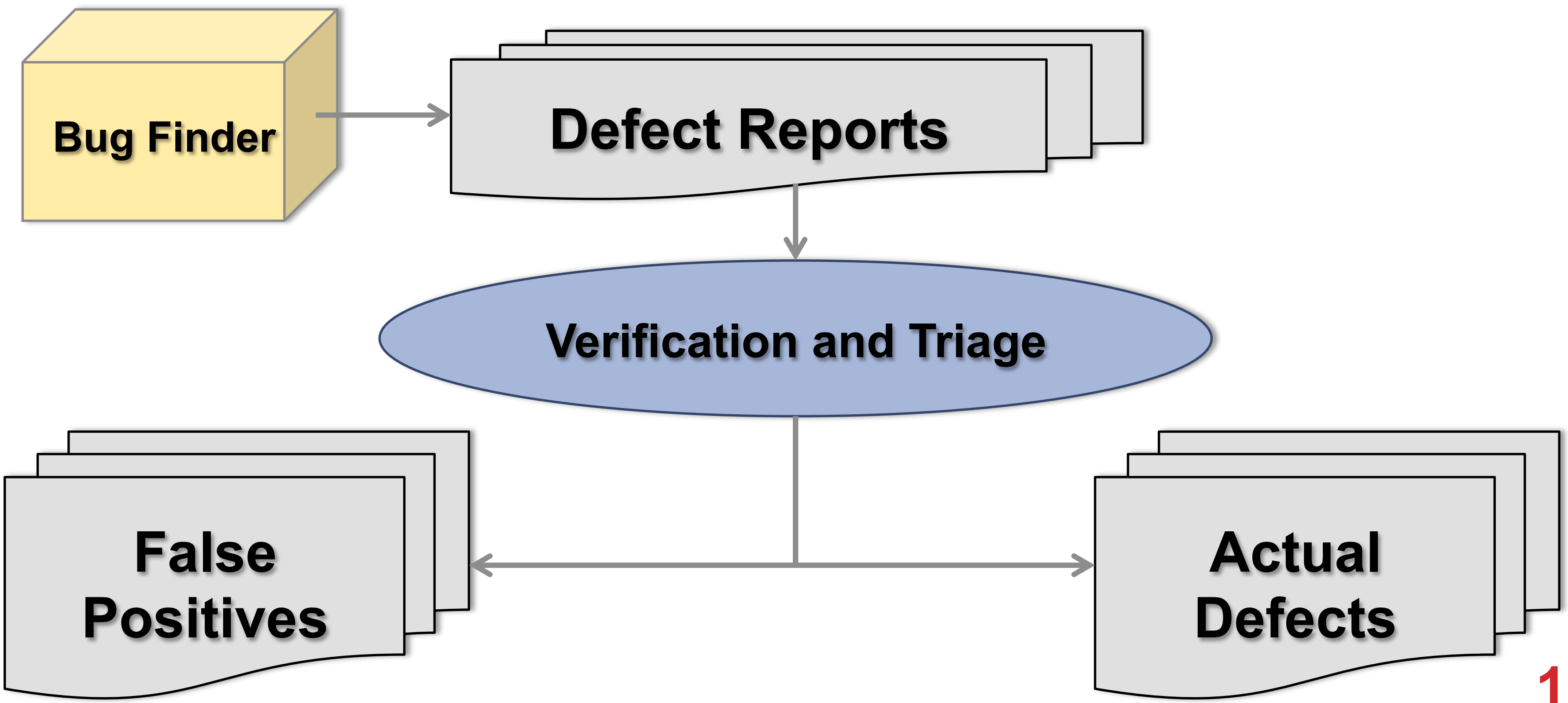
Clustering Duplicate Automatically-Generated Defect Reports

Improved Fitness Functions for Automatic Program Repair

Ensuring Documentation Quality

CLUSTERING DUPLICATE DEFECT REPORTS

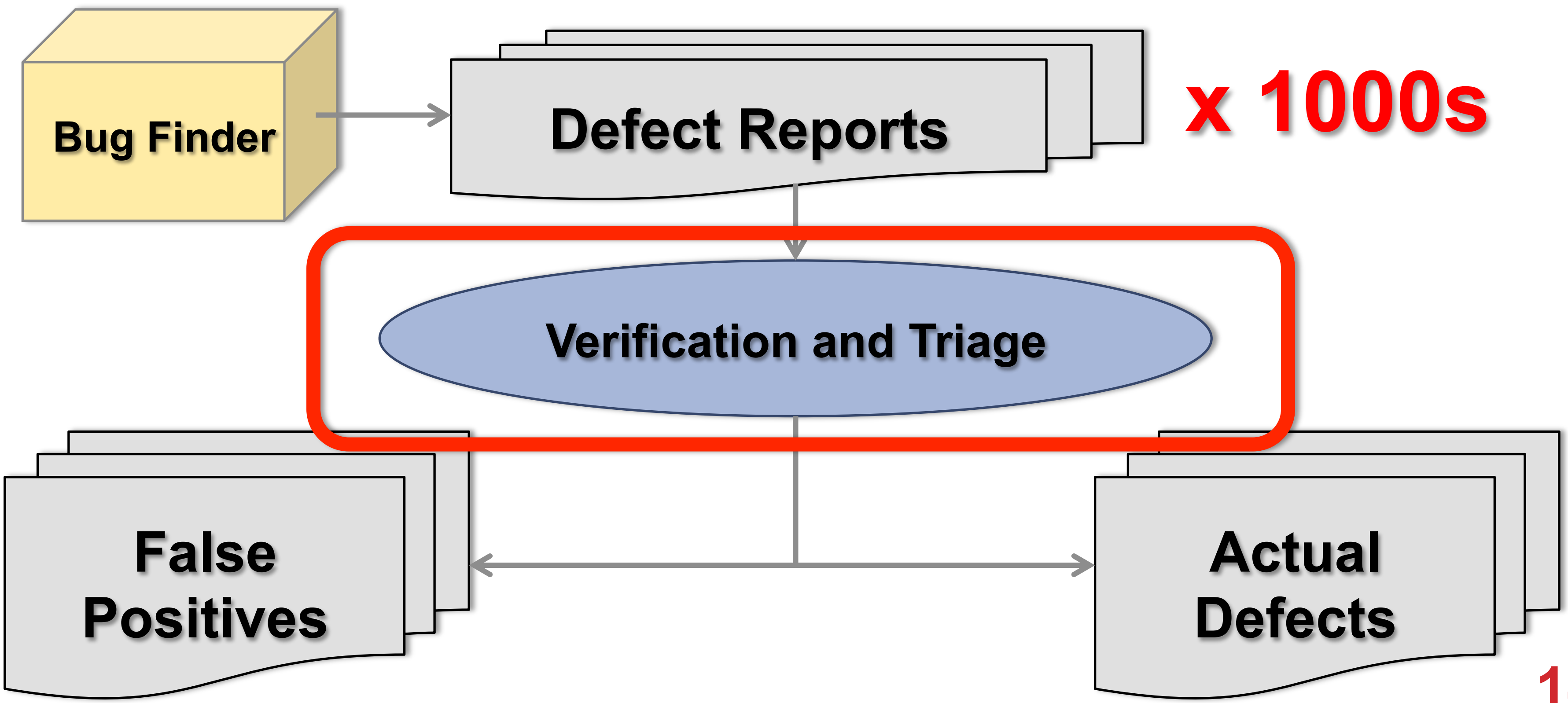
Automatic bug finders successfully report *many* bugs with little developer effort



CLUSTERING DUPLICATE DEFECT REPORTS

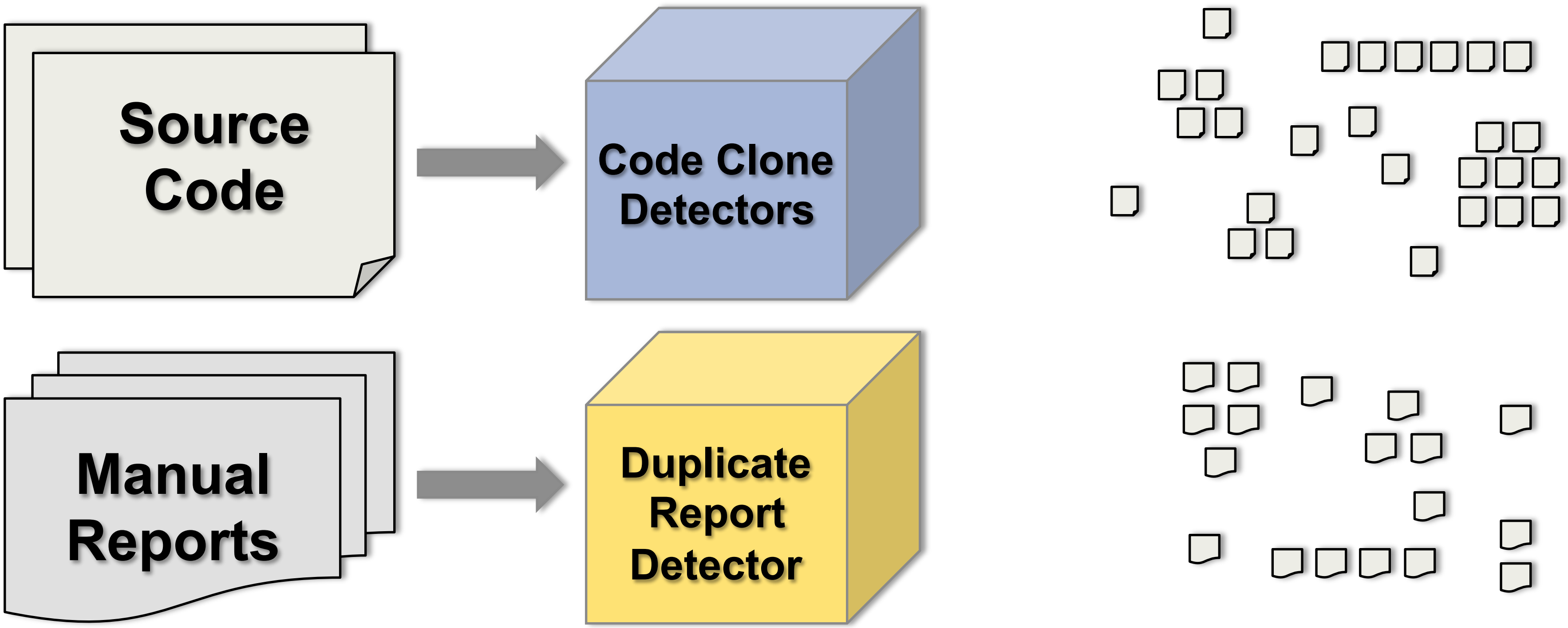
Automatic bug finders successfully report *many* bugs with little developer effort

However…



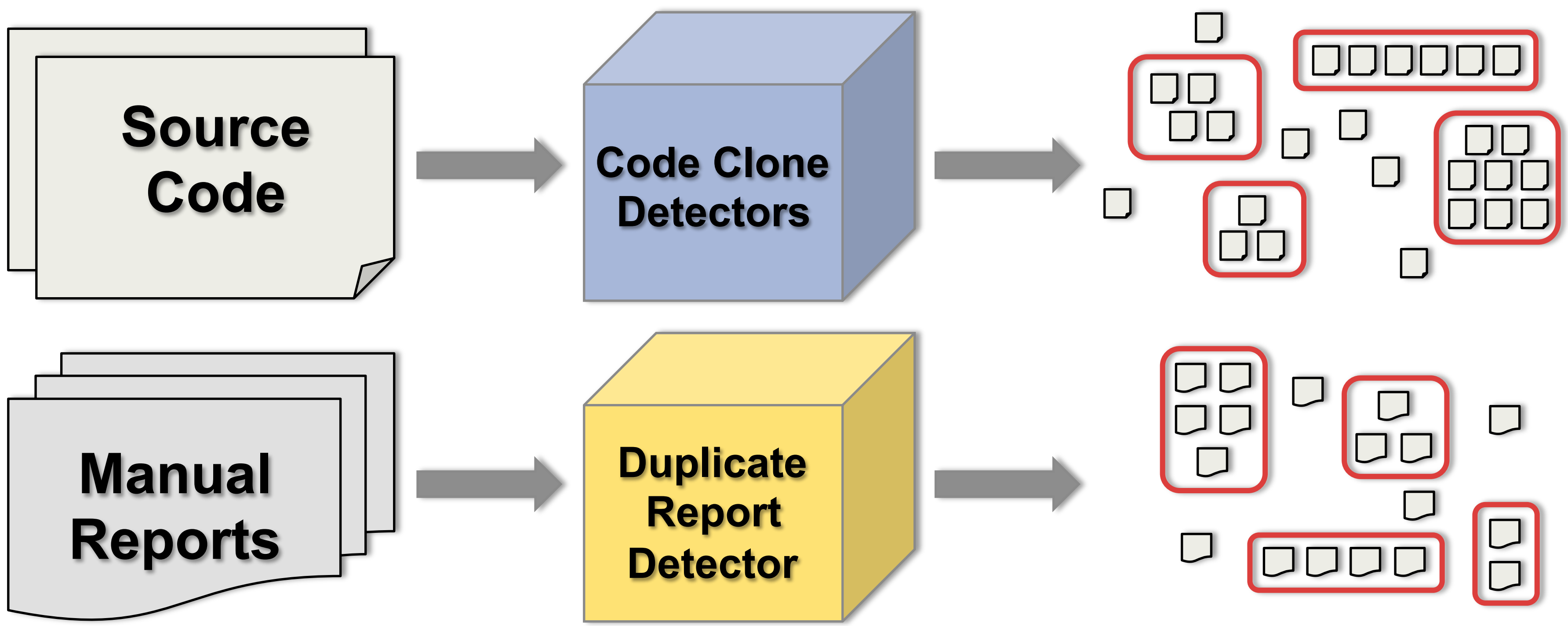
CLUSTERING DUPLICATE DEFECT REPORTS

Intuitions: Duplicates are detrimental in related fields.



CLUSTERING DUPLICATE DEFECT REPORTS

Intuitions: Duplicates are detrimental in related fields.



CLUSTERING DUPLICATE DEFECT REPORTS

Hypothesis: By exploiting the special structure of automatic defect detection tools' output we can accurately cluster defect reports to save effort by handling similar defect reports aggregately.

Success depends on:

- Internal accuracy of the produced clusters
- Amount of effort saved from clustering defect reports

CLUSTERING DUPLICATE DEFECT REPORTS

Defect Report 1

File:

NSReader.java

Suspected Line:

```
plot = lst.get(i);
```

Defect Report 2

File:

NSReader.java

Suspected Line:

```
p = lst.get(i);
```

Defect Report 3

File:

NSReader.java

Suspected Line:

```
plot = lst.get(n);
```

Defect Report 4

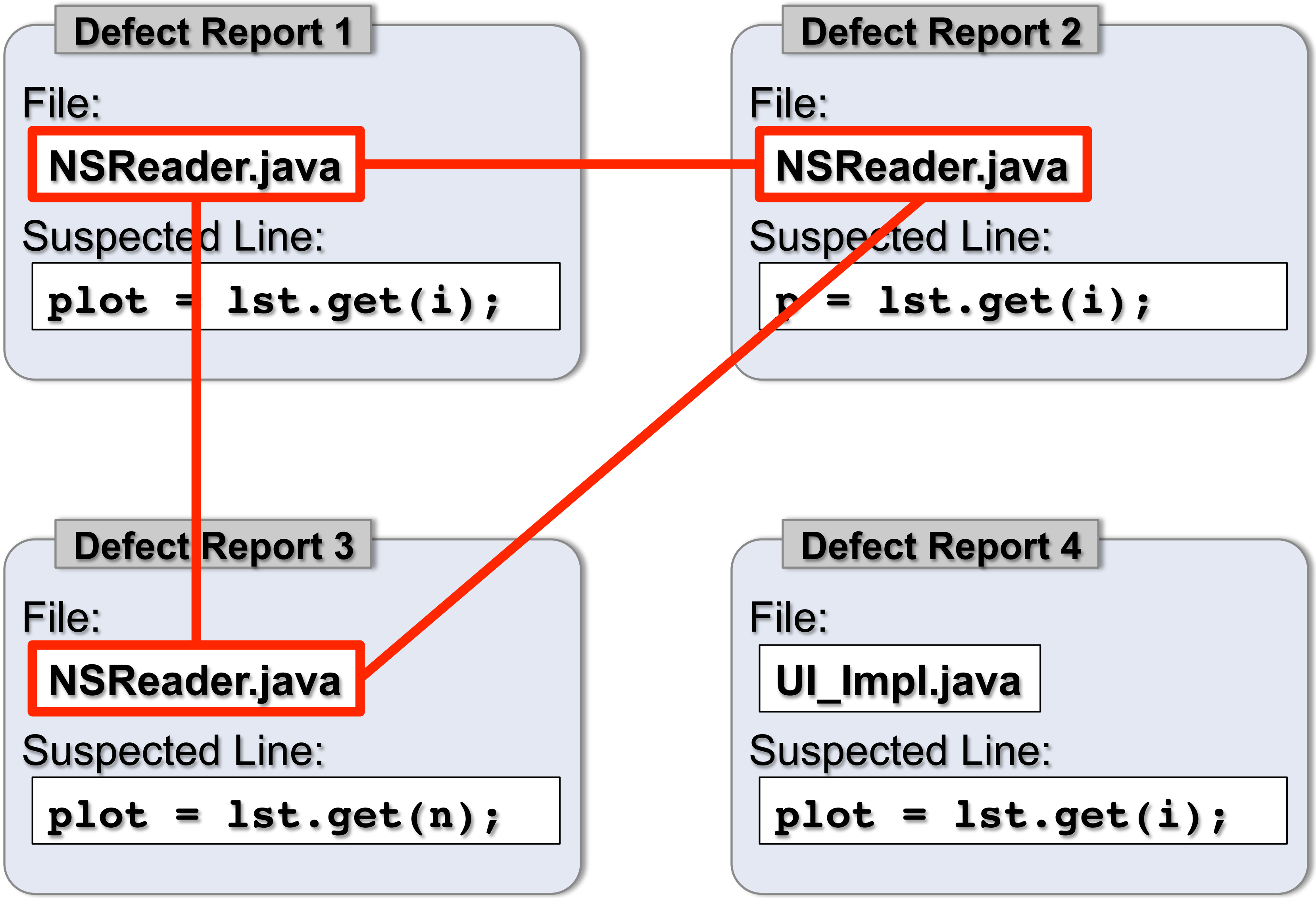
File:

UI_Impl.java

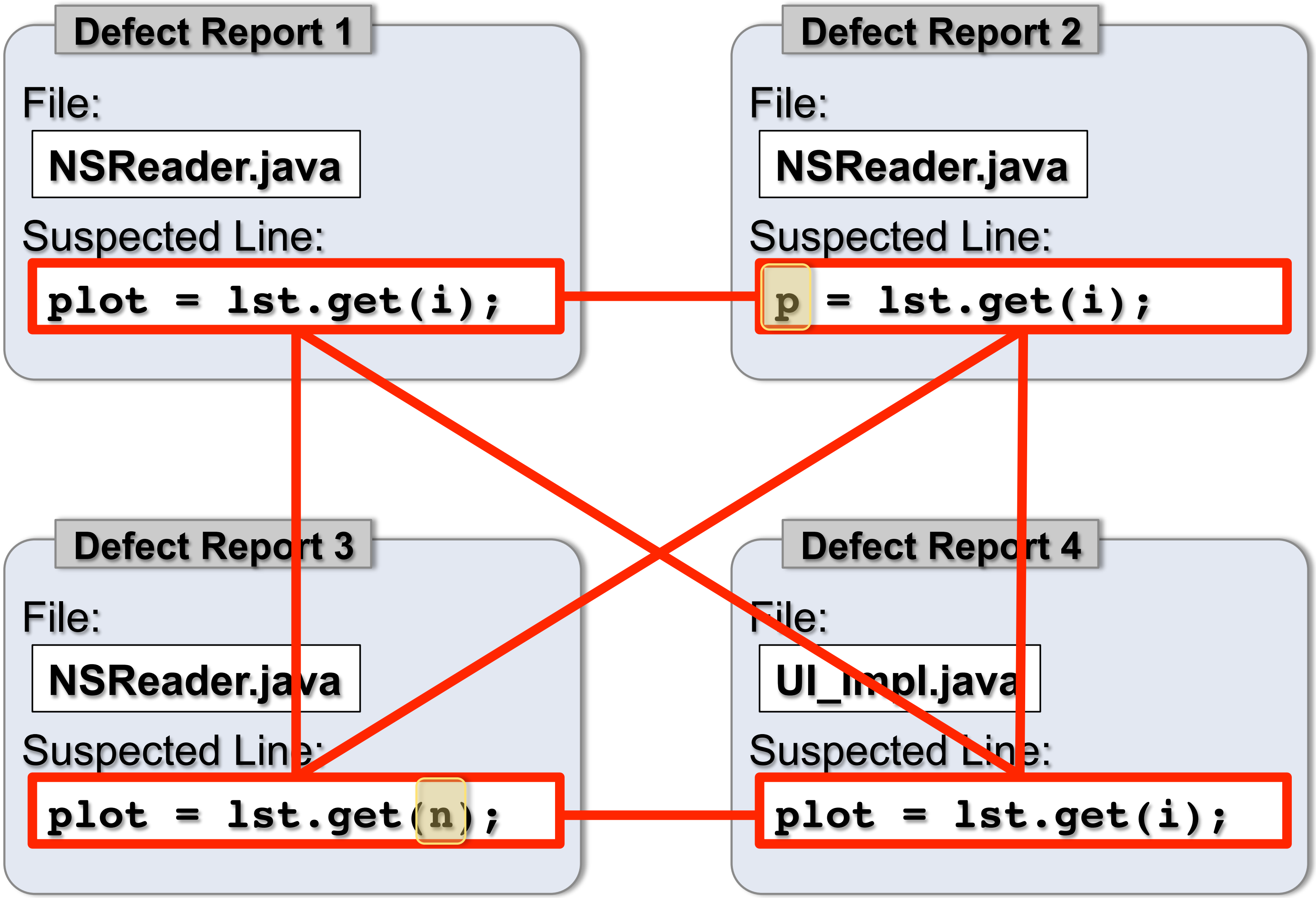
Suspected Line:

```
plot = lst.get(i);
```

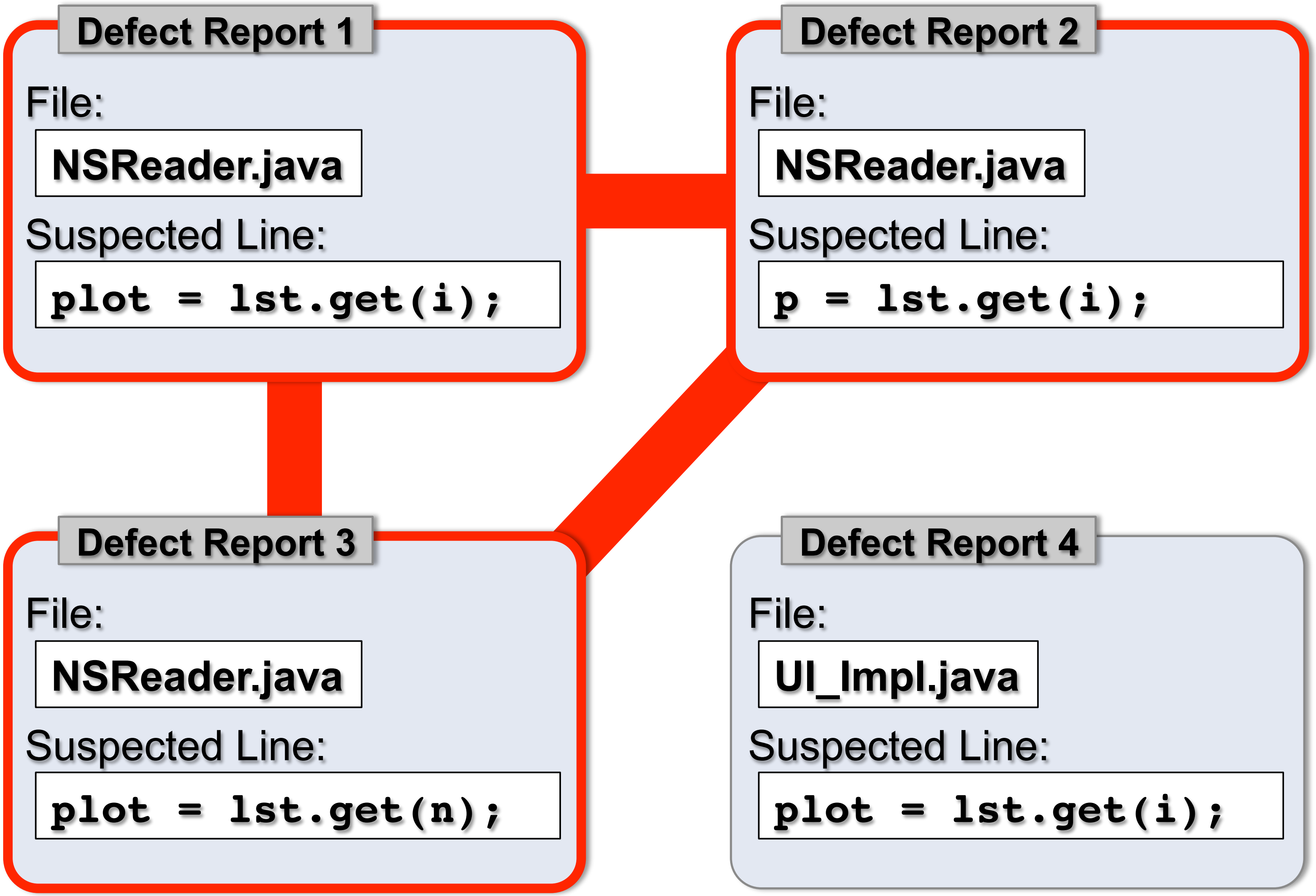
CLUSTERING DUPLICATE DEFECT REPORTS



CLUSTERING DUPLICATE DEFECT REPORTS

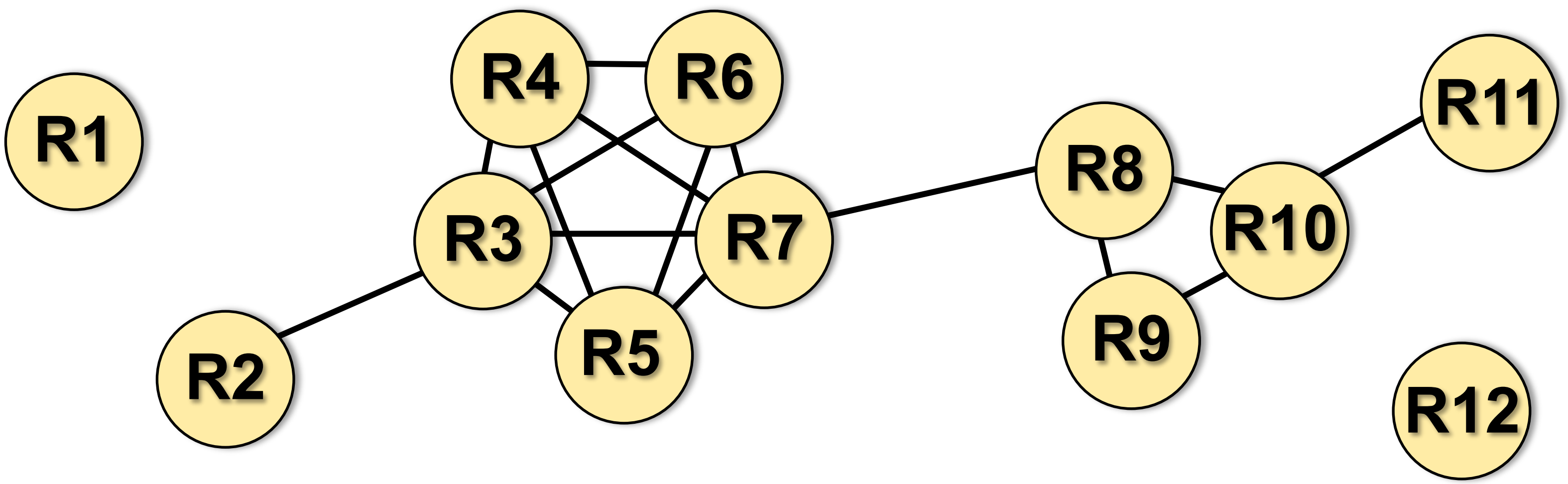


CLUSTERING DUPLICATE DEFECT REPORTS



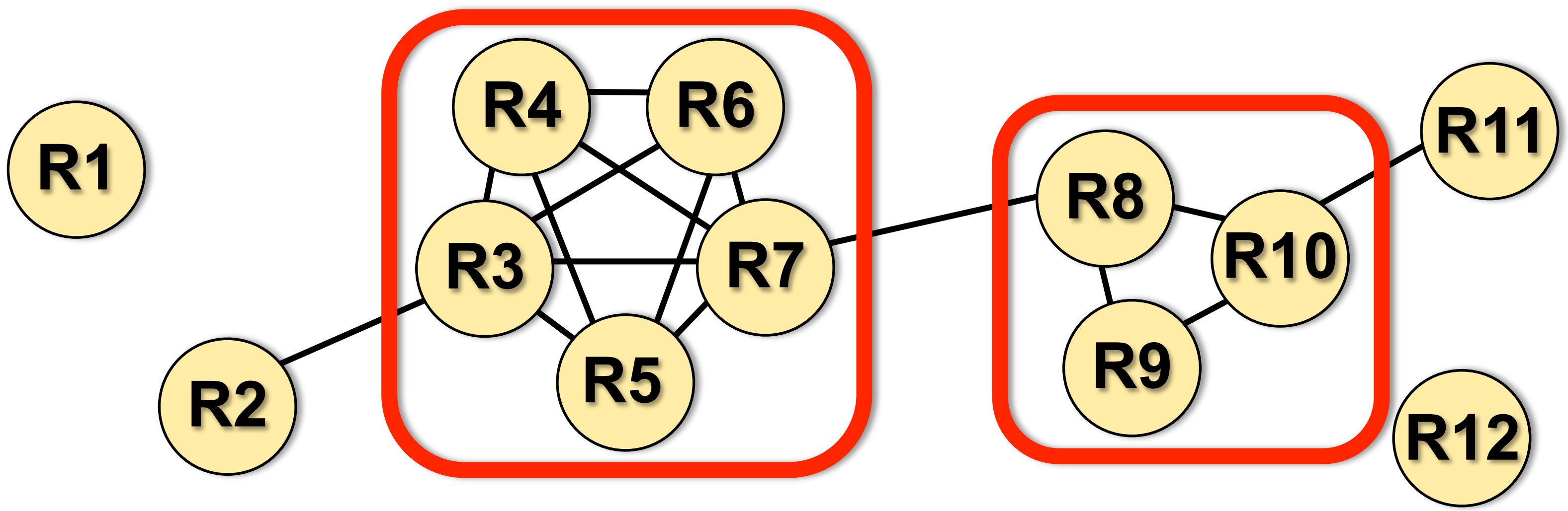
CLUSTERING DUPLICATE DEFECT REPORTS

Clustering technique:



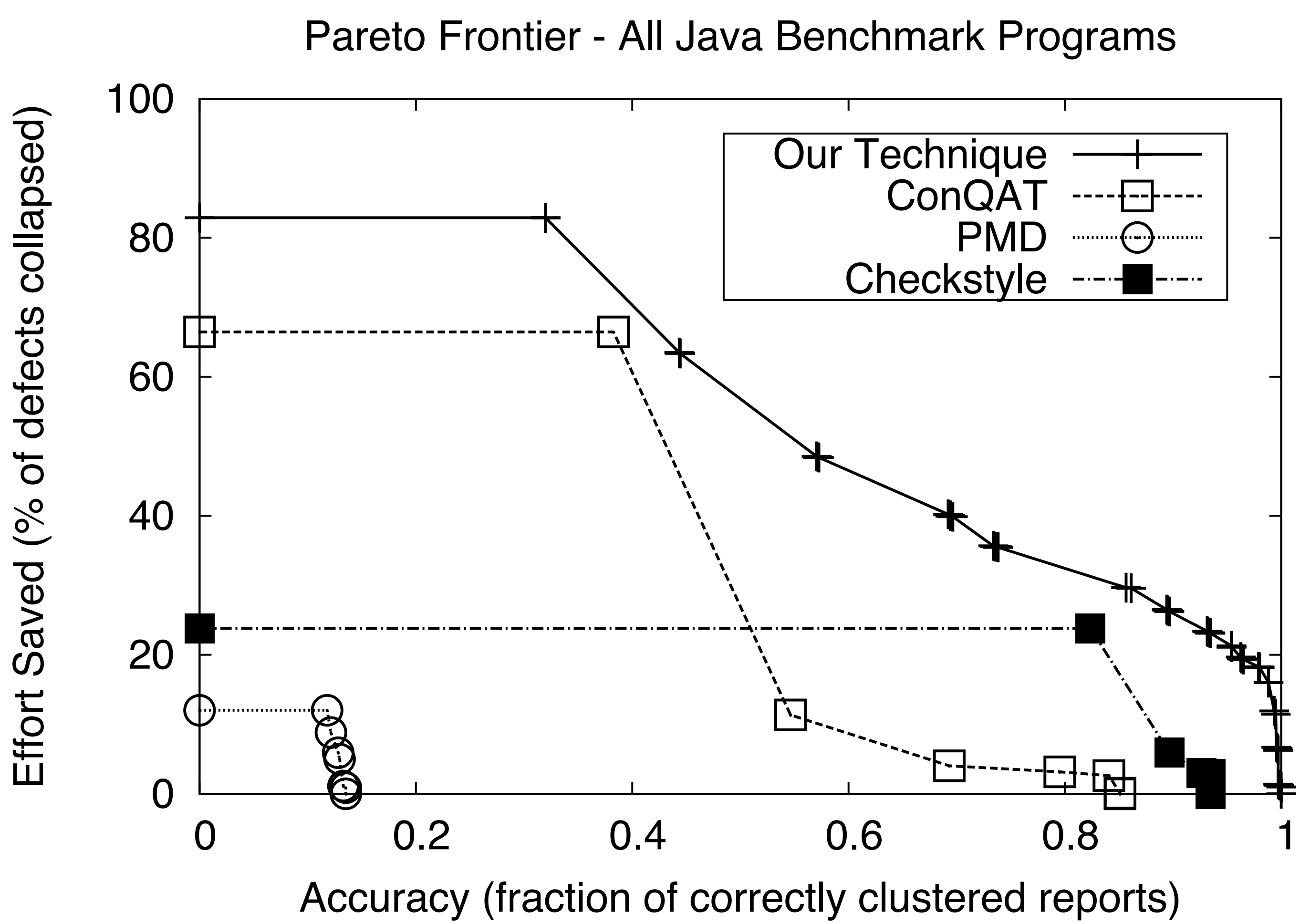
CLUSTERING DUPLICATE DEFECT REPORTS

Clustering technique:



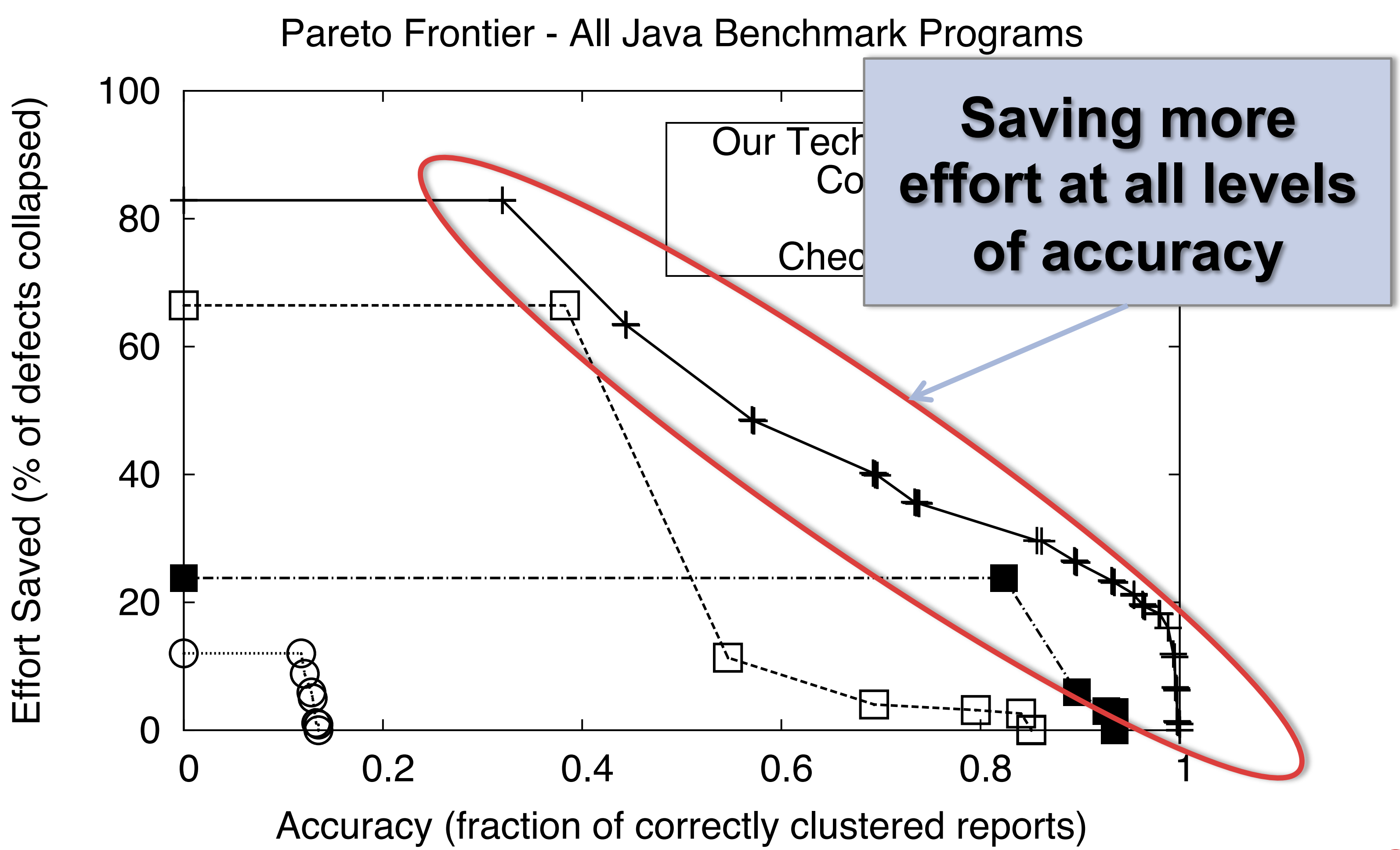
CLUSTERING DUPLICATE DEFECT REPORTS

Preliminary Cluster Accuracy vs. Effort Savings



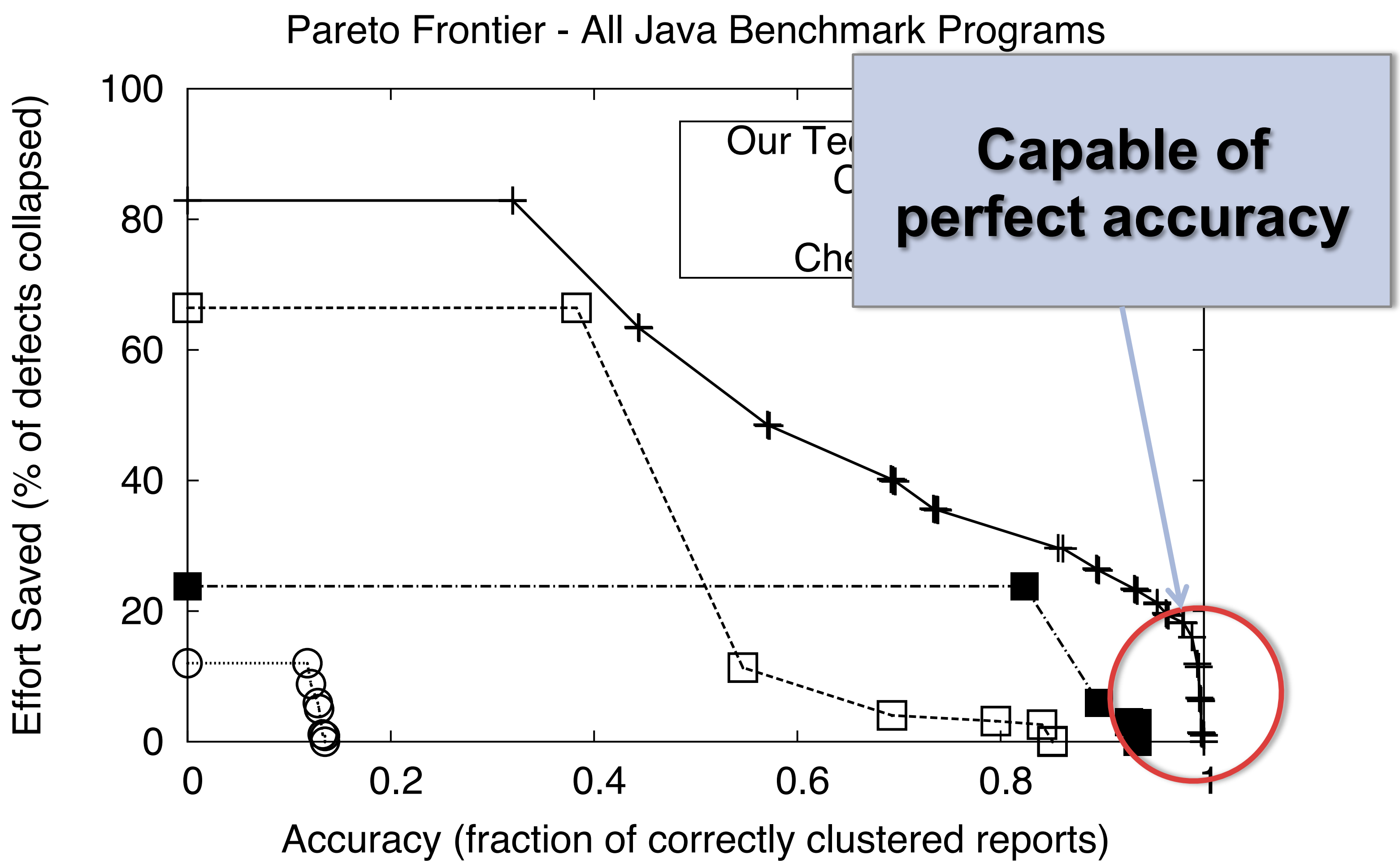
CLUSTERING DUPLICATE DEFECT REPORTS

Preliminary Cluster Accuracy vs. Effort Savings

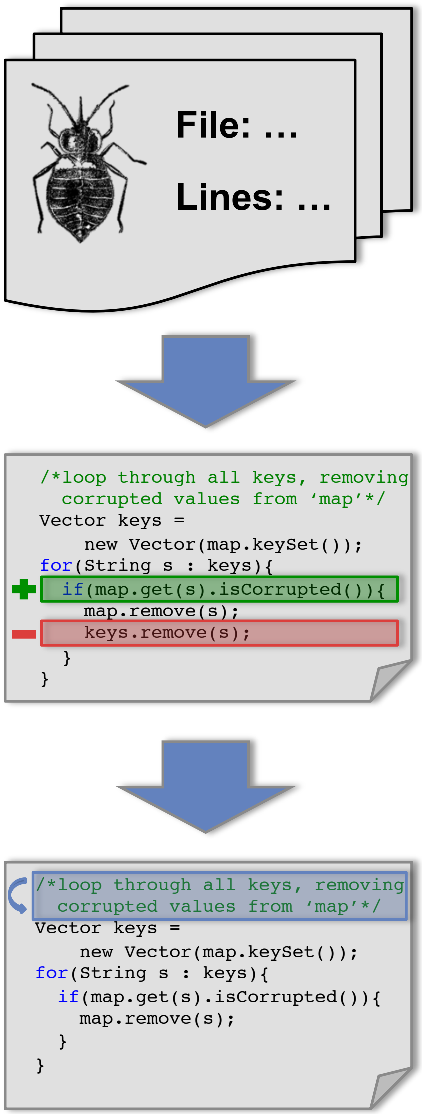


CLUSTERING DUPLICATE DEFECT REPORTS

Preliminary Cluster Accuracy vs. Effort Savings



PROJECT OUTLINE

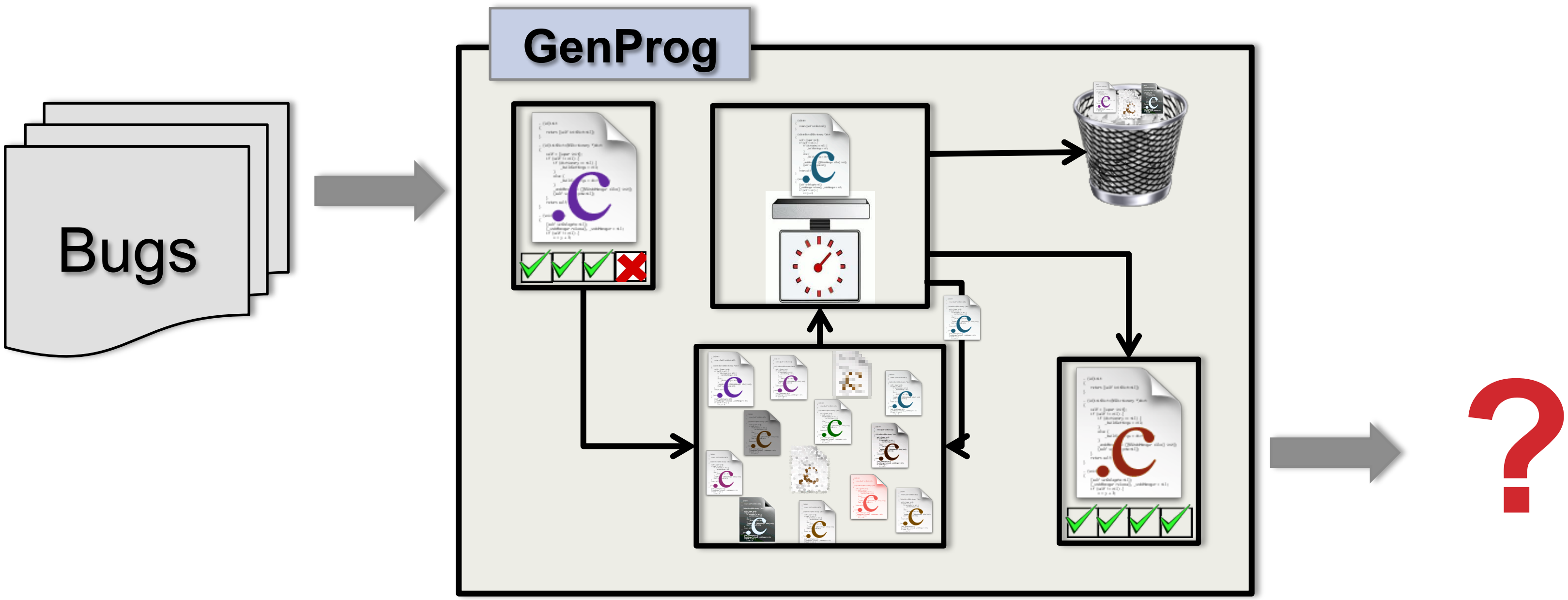


Clustering Duplicate Automatically-Generated Defect Reports

Improved Fitness Functions for Automatic Program Repair

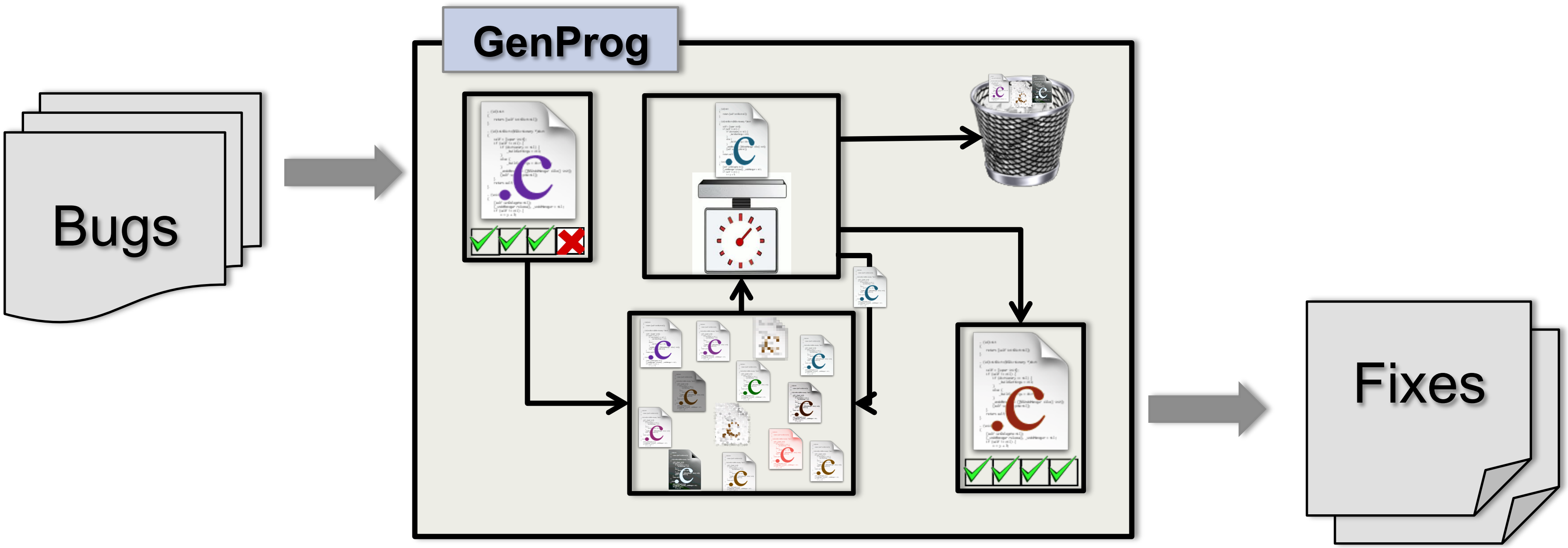
Ensuring Documentation Quality

Automatic program repair

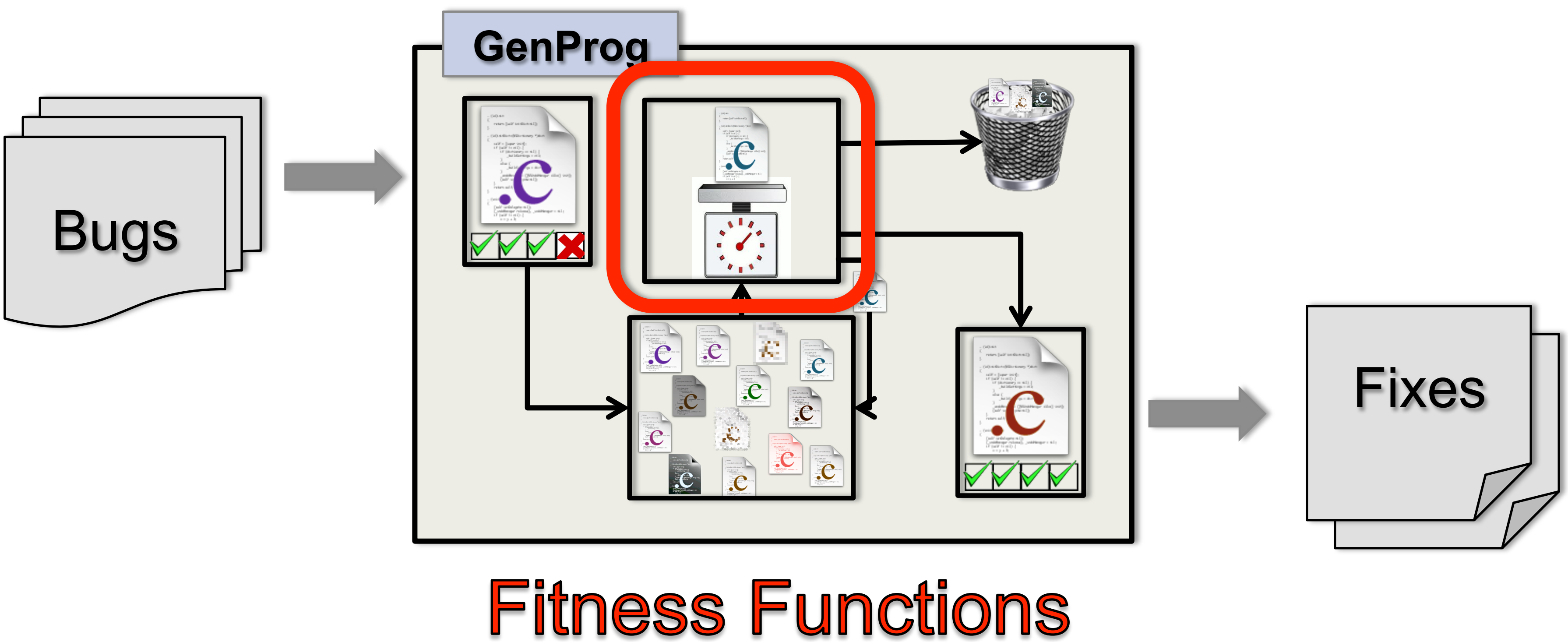


IMPROVED FITNESS FUNCTIONS

Automatic program repair can fix bugs.



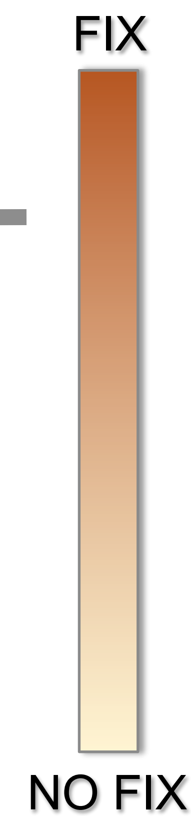
Automatic program repair can fix bugs.



IMPROVED FITNESS FUNCTIONS

- **Measuring proximity to a fix**
  - Insert, delete, and swapping lines in the program

Fix   d(135)   i(251,205)   i(774,111)   s(598,324)



IMPROVED FITNESS FUNCTIONS

- **Measuring proximity to a fix**
  - Insert, delete, and swapping lines in the program

Fix d(135) i(251,205) i(774,111) s(598,324)

M1 i(251,205)✓ i(774,111)✓ s(598,324)✓ d(63)✗ **75%**

FIX

NO FIX

IMPROVED FITNESS FUNCTIONS

- **Measuring proximity to a fix**
  - Insert, delete, and swapping lines in the program

Fix	d(135)	i(251,205)	i(774,111)	s(598,324)	
M1	i(251,205) ✓	i(774,111) ✓	s(598,324) ✓	d(63) ✗	**75%**
M2	d(84) ✗	s(844,265) ✗	i(774,111) ✓	i(735,431) ✗	**25%**

FIX

NO FIX

IMPROVED FITNESS FUNCTIONS

- **The current model of fitness does not correlate well with proximity to a fix.**

Intuitions:

- **Not all *test cases* are created equal.**
- **Not all *bugs* are created equal.**
- **Not all *fixes* are created equal.**

We propose to address the naivety of the current fitness representation.

IMPROVED FITNESS FUNCTIONS

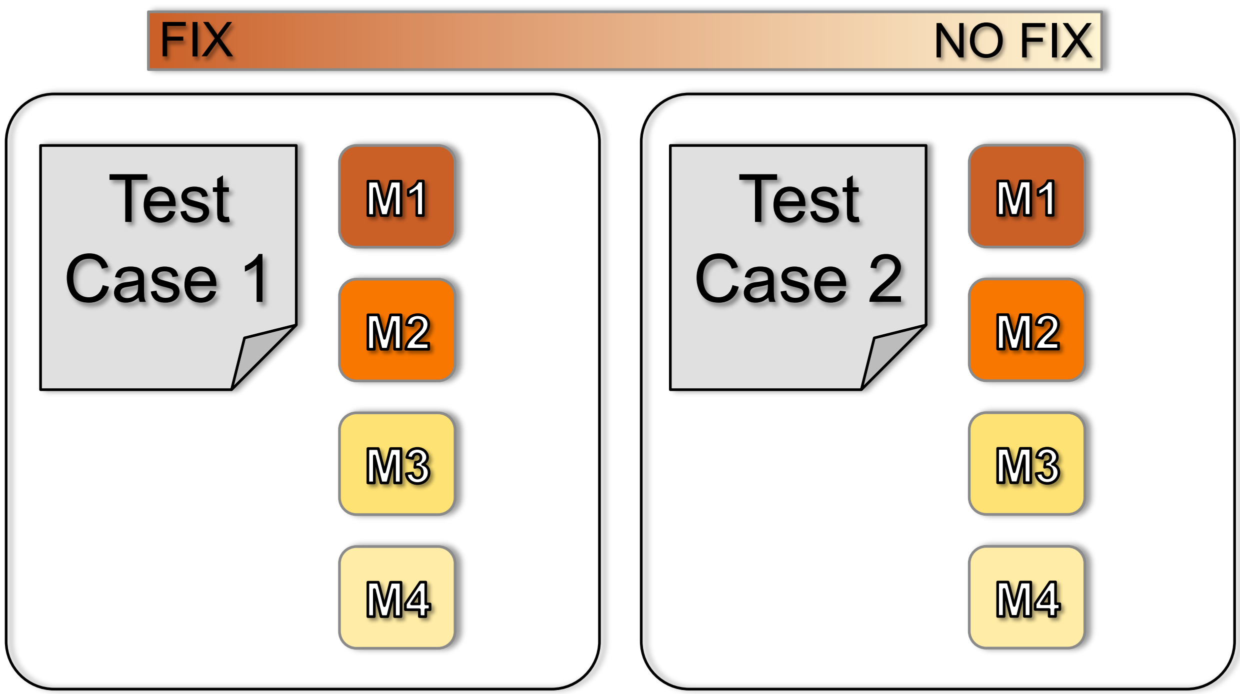
Hypothesis: By taking into account previously unused information about test cases, bugs, and fixes we can better inform the evolutionary bug fixing process to fix bugs faster and more often.

Success depends on:

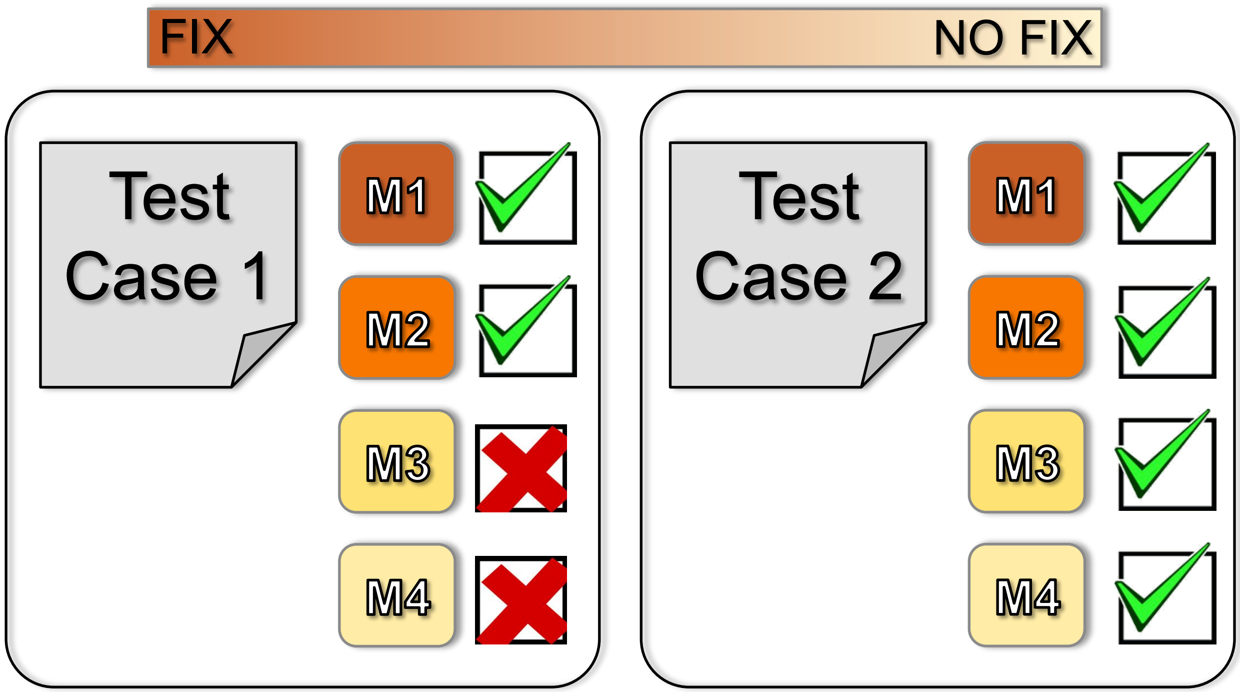
- Increase the number of bugs fixed
- For bugs that can currently be fixed, shorten the time it takes to fix them

IMPROVED FITNESS FUNCTIONS

Approach: weight test cases based on known fixes

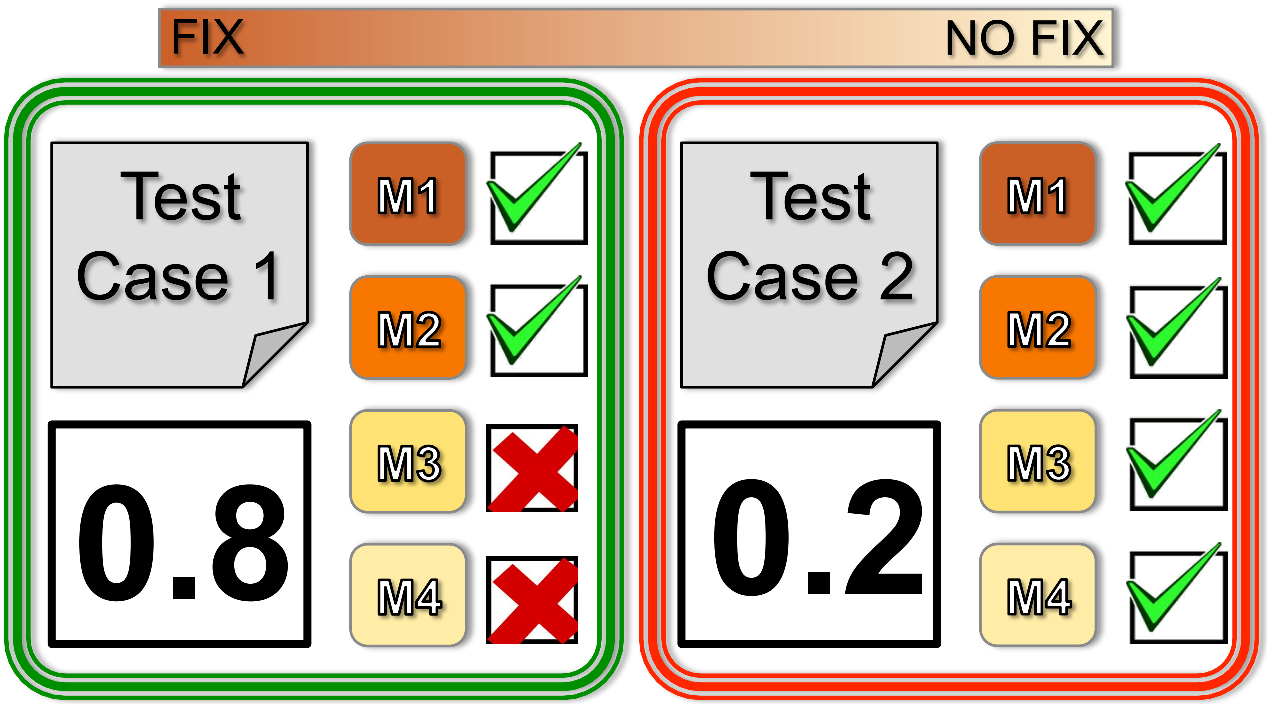


Approach: weight test cases based on known fixes



IMPROVED FITNESS FUNCTIONS

Approach: weight test cases based on known fixes



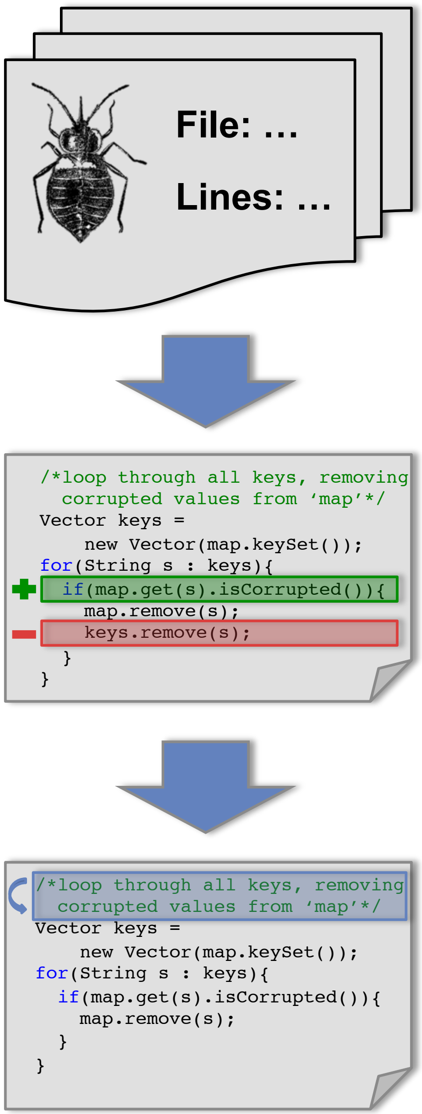
IMPROVED FITNESS FUNCTIONS

Evaluation:

- **How many <u>more bugs</u> can we fix?**
  - 55 out of 105 bugs fixed in the most recently published work[1]
- **How much can we <u>speed up</u> fixes?**
  - Computational time and monetary cost

1. Claire Le Goues, Westley Weimer, Stephanie Forrest: Representations and Operators for Improving Evolutionary Software Repair. Genetic and Evolutionary Computing Conference (GECCO) 2012

PROJECT OUTLINE



Clustering Duplicate Automatically-Generated Defect Reports

Improved Fitness Functions for Automatic Program Repair

Ensuring Documentation Quality

ENSURING DOCUMENTATION QUALITY

- **"The documentation becomes increasingly inaccurate thereby making future changes even more difficult." (Parnas)**
- **Real developers:**
  - 76% agree documentation is crucial to understanding
  - But poorly executed in practice (27% complete, 33% consistent)

ENSURING DOCUMENTATION QUALITY

```
/*loop through all keys, removing
  corrupted values from 'map'*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

```
/*loop through all keys, removing
  corrupted values from 'map'*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    map.remove(s);
  }
}
```

```
/*loop through all keys, removing
  corrupted values from 'map'*/
HashMap validMap = new HashMap();
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isValid()){
    validMap.put(s,map.get(s));
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

```
/*loop through all keys, removing
  corrupted values from 'map'*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    map.remove(s);
  }
}
```

INCONSISTENT!

Comment incorrectly describes the functionality

```
/*loop through all keys, removing
  corrupted values from 'map'*/
HashMap validMap = new HashMap();
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isValid()){
    validMap.put(s,map.get(s));
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    map.remove(s);
  }
```

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals(“Primary”))
      printf(“debug: %s\n”,
        map.get(s).toString());
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

INCOMPLETE!

Comment fails to describe all relevant functionality

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    map.remove(s);
  }
```

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals(“Primary”))
      printf(“debug: %s\n”,
        map.get(s).toString());
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

- Reduce understandability over time

Intuitions:

- Existing tools can accurately extract concepts from code and generate comments about those concepts.
- There should be natural language overlap in a high quality comments and the associated code.

ENSURING DOCUMENTATION QUALITY

Hypothesis: By comparing concepts extracted from the code with the existing comments, we can accurately identify inconsistent and incomplete documentation.

Success depends on:

- The accuracy of our incomplete and inconsistent comment identification technique
- The ease with which humans update and understand comments when using our tool

ENSURING DOCUMENTATION QUALITY

Approach:

```
/*loop through all keys, removing
  corrupted values from 'map'*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals("Primary"))
      printf("debug: %s\n",
        map.get(s).toString());
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

Approach:

```
/*loop through all keys, removing
  corrupted values from map'*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals("Primary"))
      printf("debug: %s\n",
        map.get(s).toString());
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

Approach:

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals(“Primary”))
      printf(“debug: %s\n”,
        map.get(s).toString());
    map.remove(s);
  }
}
```

Approach:

Generated Documentation (DeltaDoc):

Now call `printf` if `s` is `“Primary”`

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals(“Primary”))
      printf(“debug: %s\n”,
        map.get(s).toString());
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

Approach:

Generated Documentation (DeltaDoc):

Now call `printf` if `s` is `“Primary”`

Existing comment lacks this info, thus is **incomplete**.

```
/*loop through all keys, removing
  corrupted values from ‘map’*/
Vector keys =
    new Vector(map.keySet());
for(String s : keys){
  if(map.get(s).isCorrupted()){
    if(s.equals(“Primary”))
      printf(“debug: %s\n”,
        map.get(s).toString());
    map.remove(s);
  }
}
```

ENSURING DOCUMENTATION QUALITY

Evaluation: Human studies

Study 1 – (FIX) – Humans identify and fix low-quality comments with and without our tool

Study 2 – (RATE) – Using the resulting data set, have different humans identify and rate the modified comments

Evaluation:

- **Compare our tool's accuracy when identifying low-quality comments with humans' abilities to do the same task**
  - Use identification data from both FIX and RATE
  - Inter-annotator agreement vs. tool-human agreement

Evaluation:

- **Measure our tools' effectiveness in helping humans identify and fix low quality comments**
  - Effort (time) in FIX
  - Use ratings from RATE to compare data from groups in FIX

SUMMARY

We propose work that will specifically target three parts of the maintenance process to reduce the overall cost:

1. **Cluster automatically-generated defect reports to facilitate triage and bug fixing**
2. **Improve fitness functions to aid in automatic program repair to fix more bugs, faster**
3. **Identify incomplete and inconsistent comments to promote continued documentation quality and foster program understanding**

COMPREHENSIVE GOALS - REVISITED

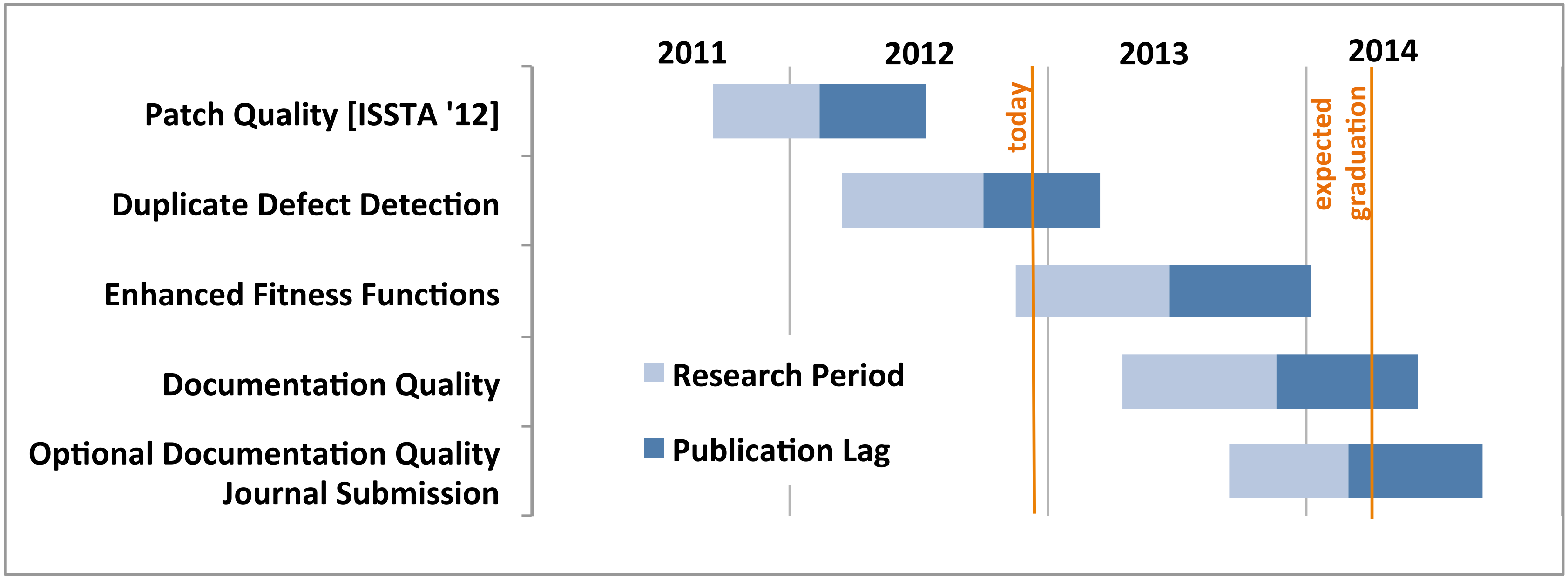
We desire techniques that add minimal human effort

- **Techniques work "*off the shelf*"**
- **Encourages incremental adoption**

Use latent, often-overlooked information

- **Syntactic, semantic defect report fields**
- **Test case quality, types of bugs/fixes**
- **Natural language in code and comments**

RESEARCH TIMELINE



Publications to date:

- E. Schulte, Z. Fry, E. Fast, W. Weimer, S. Forrest. *Software Mutational Robustness*. Genetic Programming and Evolvable Machines 2013. (*under submission*)
- Z. Fry, W. Weimer. *Clustering Static Analysis Defect Reports to Reduce Maintenance Costs.* International Conference on Tools and Algorithms for the Construction and Analysis of Systems 2013 (TACAS). (under submission)
- Z. Fry, B. Landau, W. Weimer. *A Human Study of Patch Maintainability.* International Symposium on Software Testing and Analysis 2012 (ISSTA). (Acc Rate: 29%)
- Z. Fry, W. Weimer. *A Human Study of Fault Localization Accuracy.* International Conference on Software Maintenance 2010 (ICSM). (Acc. Rate: 26%)

QUESTIONS?