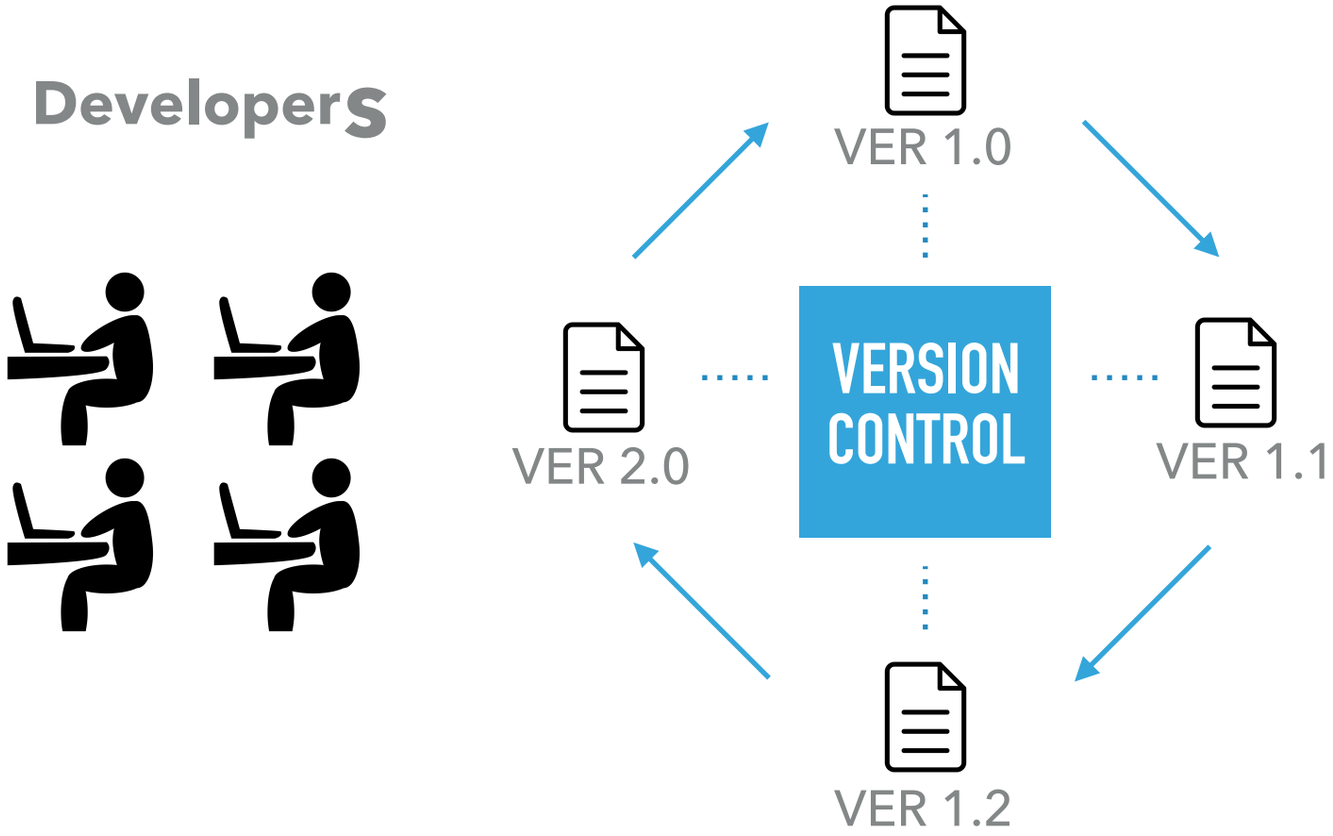
VISUALIZING A TANGLED CHANGE FOR SUPPORTING ITS DECOMPOSITION AND COMMIT CONSTRUCTION

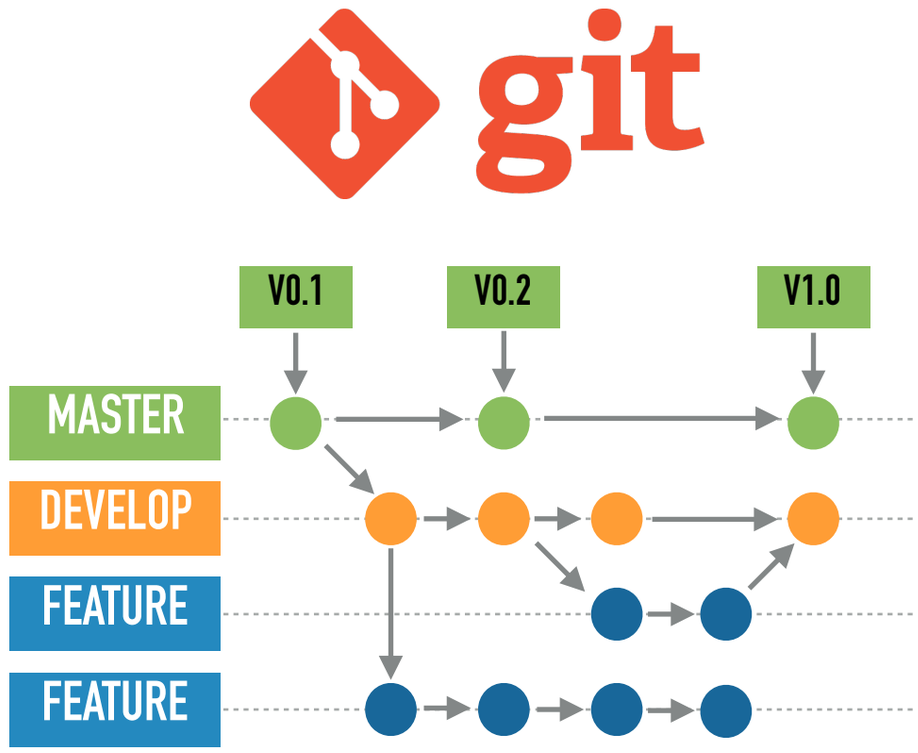
Sarocha Sothornprapakorn, Shinpei Hayashi, Motoshi Saeki
Tokyo Institute of Technology

COMPSAC, 24TH JULY 2018

INTRODUCTION

VERSION CONTROL IS IMPORTANT FOR SOFTWARE DEVELOPMENT





An easy to understand commit should contain only one task (new feature / bug fixing / refactoring)

- Yida Tao, FSE 2012

Yoda Tao, FSE 2012 “How do software engineers understand code changes?: An exploratory study in industry”

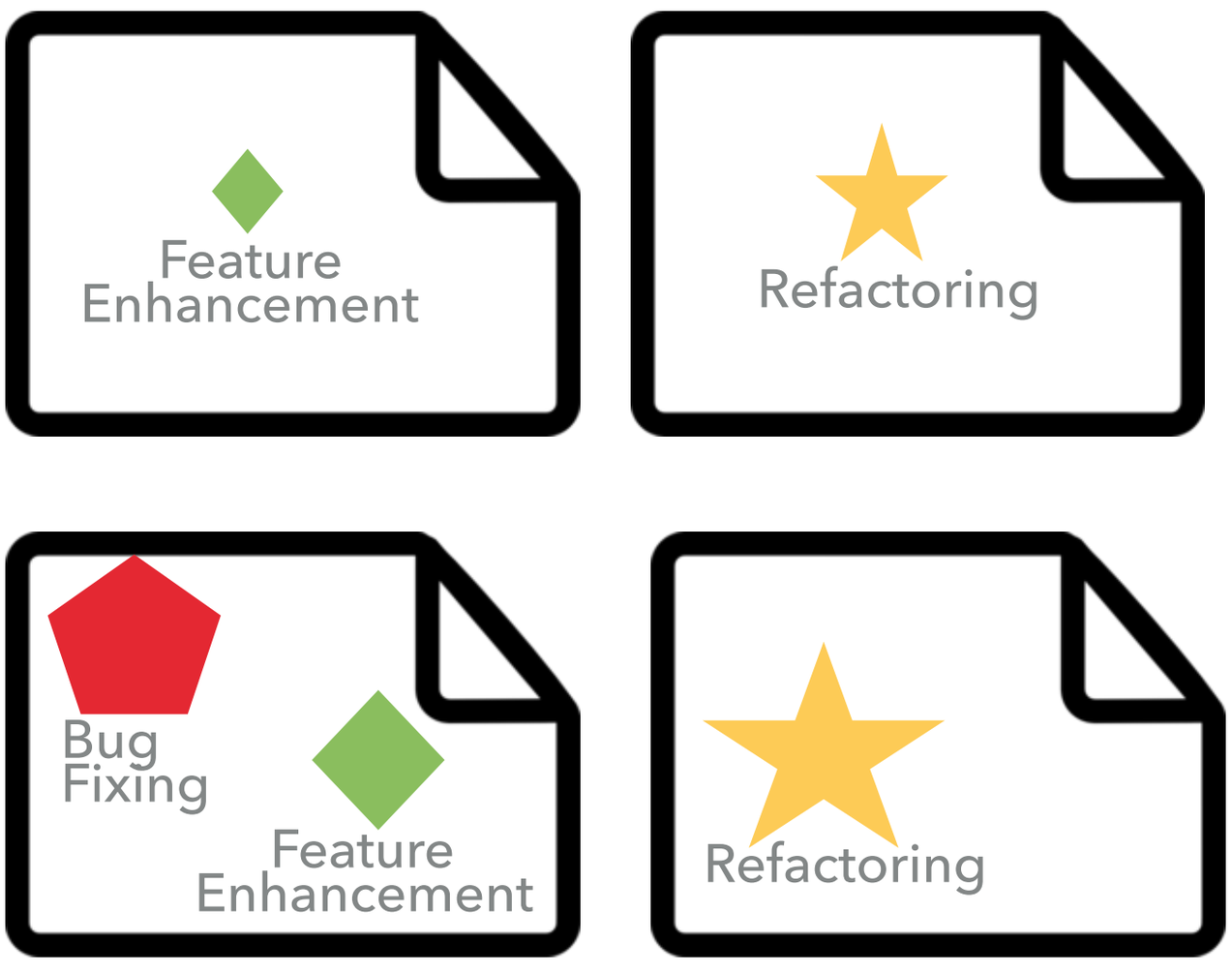
> “ WHEN COMMITTING CHANGES, DEVELOPERS FREQUENTLY GROUP SEPARATE CHANGES INTO A SINGLE COMMIT, RESULTING IN A TANGLED CHANGE.”

Kim Herzig and Andreas Zeller

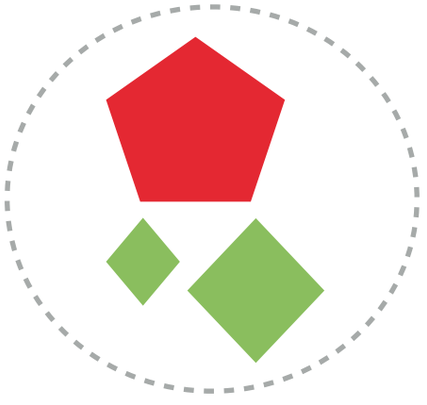
“The Impact of Tangled Code Changes”, MSR 2013

TANGLED CHANGE WITH REFACTORING

Big changes across many files



Refactoring
+ Behavior preserved



Non-refactoring
+ Behavior changed

INSPECT TANGLED CHANGES USING FILE LIST-BASED IS DIFFICULT

src/seedu/addressbook/commands/FindCommand.java
src/seedu/addressbook/commands/ViewAllCommand.java
src/seedu/addressbook/data/person/Address.java
src/seedu/addressbook/data/person/Contact.java
src/seedu/addressbook/data/person/Email.java
src/seedu/addressbook/data/person/Phone.java
test/java/seedu/addressbook/commands/FindCommandTest.java

Add New Feature :
- find address (sensitive case)
- fixed wrong view all info
Refactor :
- extract superclass contact from address, phone, email
- rename function getPersonsWithNameContainingAnyKeyword -> getPersonsContainingAnyKeyword

Commit: ebcd1ea19a6c3f69b69d37ab6ea993932395(

...addressbook/commands/FindCommand.java

Hunk 1 : Lines 16-22 Reverse hu

```
16 16
17 17      public static final String COMMAND_WORD =
18 18
19    -    public static final String MESSAGE_USAGE =
   19 +    public static final String MESSAGE_USAGE =
20 20
21 21      private final Set<String> keywords;
22 22
```

Hunk 2 : Lines 33-54 Reverse hu

```
33 33
34 34      @Override
35 35      public CommandResult execute() {
36    -        final List<ReadOnlyPerson> personsFou
   36 +        final List<ReadOnlyPerson> personsFou
37 37          return new CommandResult(getMessageFo
38 38      }
39 39
40 40      /**
41    -     * Retrieves all persons in the address b
   41 +     * Retrieves all persons in the address b
```

Example **file list-based** visualization
from SourceTree git client application

PROBLEM = FILE LIST BASED VISUALIZATION IS INAPPROPRIATE

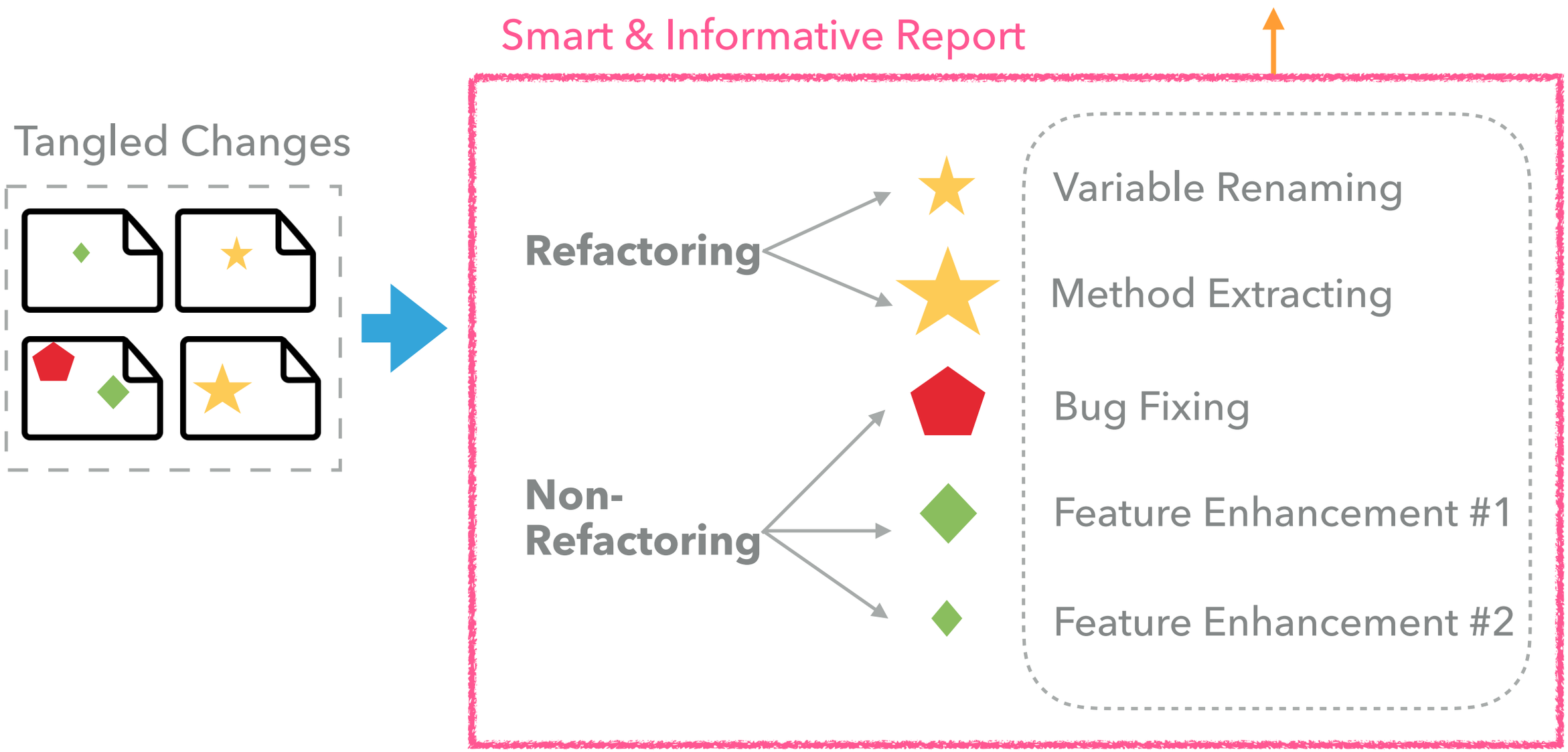
1. Hard to define behavior preserving and non-behavior preserving

2. Hard to separate changes from different tasks

3. Multiple grouping criteria -> Task level commits

TANGLED CHANGE WITH REFACTORING

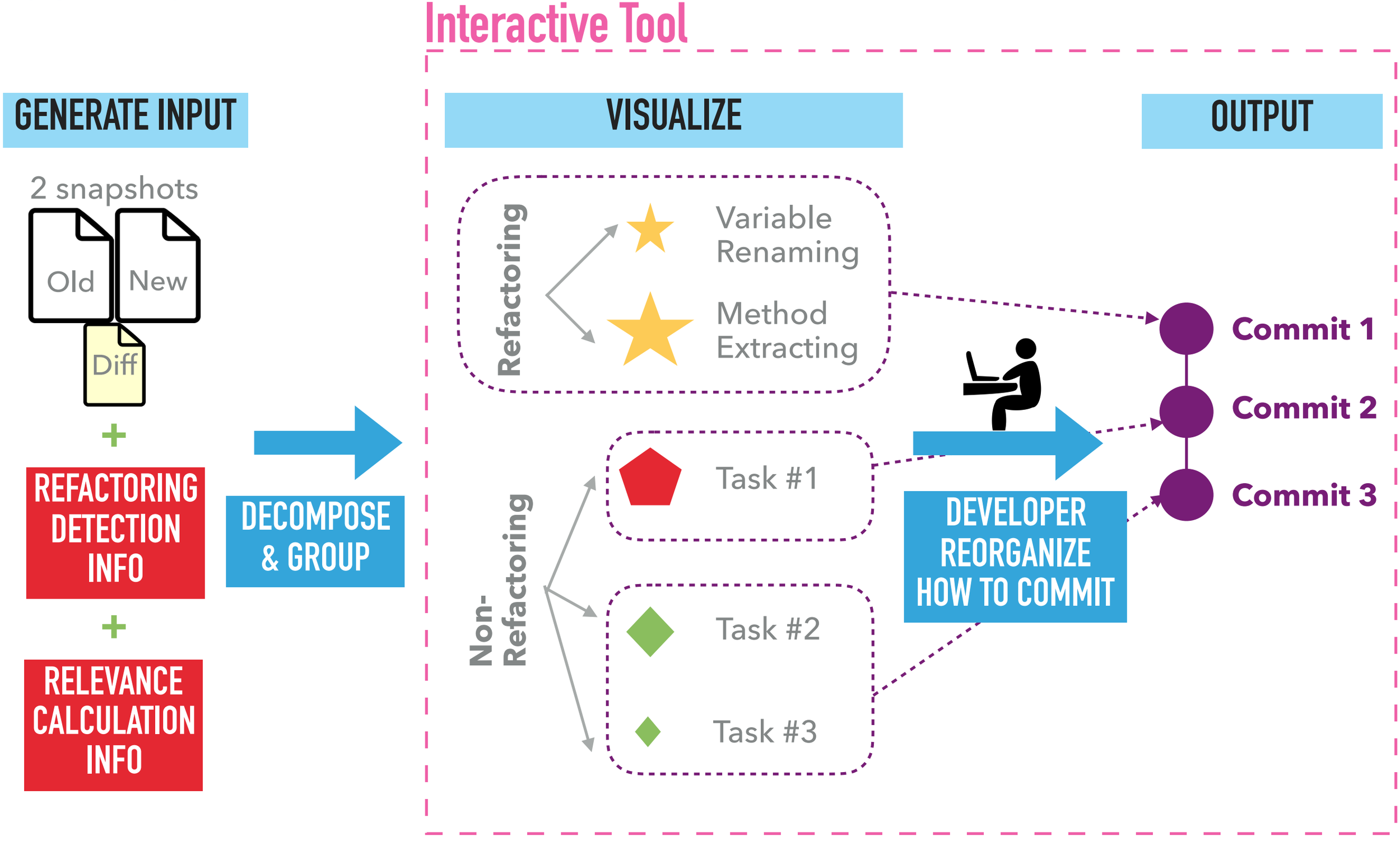
These grouping can be interactively **ReorganizeD** by a **DEVELOPER**



Purpose : Construct new commits from a tangled change which contains refactoring & non-refactoring

PROPOSED TECHNIQUE

USE FLOW


Interactive Tool
GENERATE INPUT
VISUALIZE
OUTPUT
2 snapshots
Old
New
Diff
+
REFACTORING DETECTION INFO
+
RELEVANCE CALCULATION INFO
DECOMPOSE & GROUP
Refactoring
Variable Renaming
Method Extracting
Non-Refactoring
Task #1
Task #2
Task #3
DEVELOPER REORGANIZE HOW TO COMMIT
Commit 1
Commit 2
Commit 3

MAIN CLAIM

" Informative tree visualization supports the tangled change decomposition and understanding "

CONTRIBUTIONS

1. New tangled change visualization technique
   + Enables user to configures change grouping

2. Semi-automated interactive tool + Refactoring detection technique + Relevance calculation technique => Tangled change decomposition

3. Controlled experiments

MOTIVATION EXAMPLE

SimpleCode.java

```
1  1
2  2   public class SimpleCode {
3  3       public static void main(String args[]) {
4      -        printA();
5      -        printB();
6      -        printC();
   4   +        printAA();
   5   +        printBB();
   6   +        printCC();
   7   +        Hello h = new Hello("Tua");
   8   +        h.printHello();
7  9       }
8      -    public static void printA() {
   10  +    public static void printAA() {
9  11          System.out.println("Hello World A!");
10 12      }
11     -    public static void printB() {
12     -        System.out.println("Hello World B!");
   13  +    public static void printBB() {
   14  +        System.out.println("Hello World BB!");
13 15      }
14     -    public static void printC() {
   16  +    public static void printCC() {
15 17          System.out.println("Hello World C!");
16 18      }
17 19  }
```

Hello.java

```
1  1
2  2   public class Hello {
3  3       private String name;
4      -
   4   +    public Hello(String name) {
   5   +        this.name = name;
   6   +    }
   7   +    public void printHello() {
   8   +        System.out.println("Hello " + name + "
   9   +    }
5  10  }
```

- **Refactoring**
  - printA() -> printAA()
  - printB() -> printBB()
  - printC() -> printCC()
- **Non-refactoring**
  - PrintHello + name
  - Change print “Hello World BB!”

INPUT

SimpleCode.java

```
-  printA();
+ printAA();
```

```
-  printB();
+ printBB();
```

```
-  printC();
+ printCC();
```

```
+ Hello h = new Hello("Tua");
```

```
+ h.printHello();
```

```
-  public static void printA() {
+ public static void printAA() {
```

```
-  public static void printB() {
+ public static void printBB() {
```

```
-  System.out.println("Hello World B!");
+ System.out.println("Hello World BB!");
```

```
-  public static void printC() {
+ public static void printCC() {
```

Hello.java

```
+ public Hello(String name) {
+      this.name = name;
+ }
```

```
+ public void printHello() {
+      System.out.println("Hello " + name +
        ", How are you today?");
+ }
```

REFACTORING DETECTION

Main Purpose : Separate refactoring operations

Refactoring Miner Tool

- Multiple refactorings captured
- Some of refactorings are related –> **Mapping Rules**

Refactoring detected results

3 refactorings found

- Rename Method printA() renamed to printAA() in class SimpleCode
- Rename Method printB() renamed to printBB() in class SimpleCode
- Rename Method printC() renamed to printCC() in class SimpleCode

Tsantalis et al.,CASCON '13, "A multidimensional empirical study on refactoring activity"

REFACTORING MAPPING

SimpleCode.java

```
-  printA();
+ printAA();
```

```
-  printB();
+ printBB();
```

```
-  printC();
+ printCC();
```

```
+ Hello h = new Hello("Tua");
```

```
+ h.printHello();
```

```
-  public static void printA() {
+ public static void printAA() {
```

```
-  public static void printB() {
+ public static void printBB() {
```

```
-  System.out.println("Hello World B!");
+ System.out.println("Hello World BB!");
```

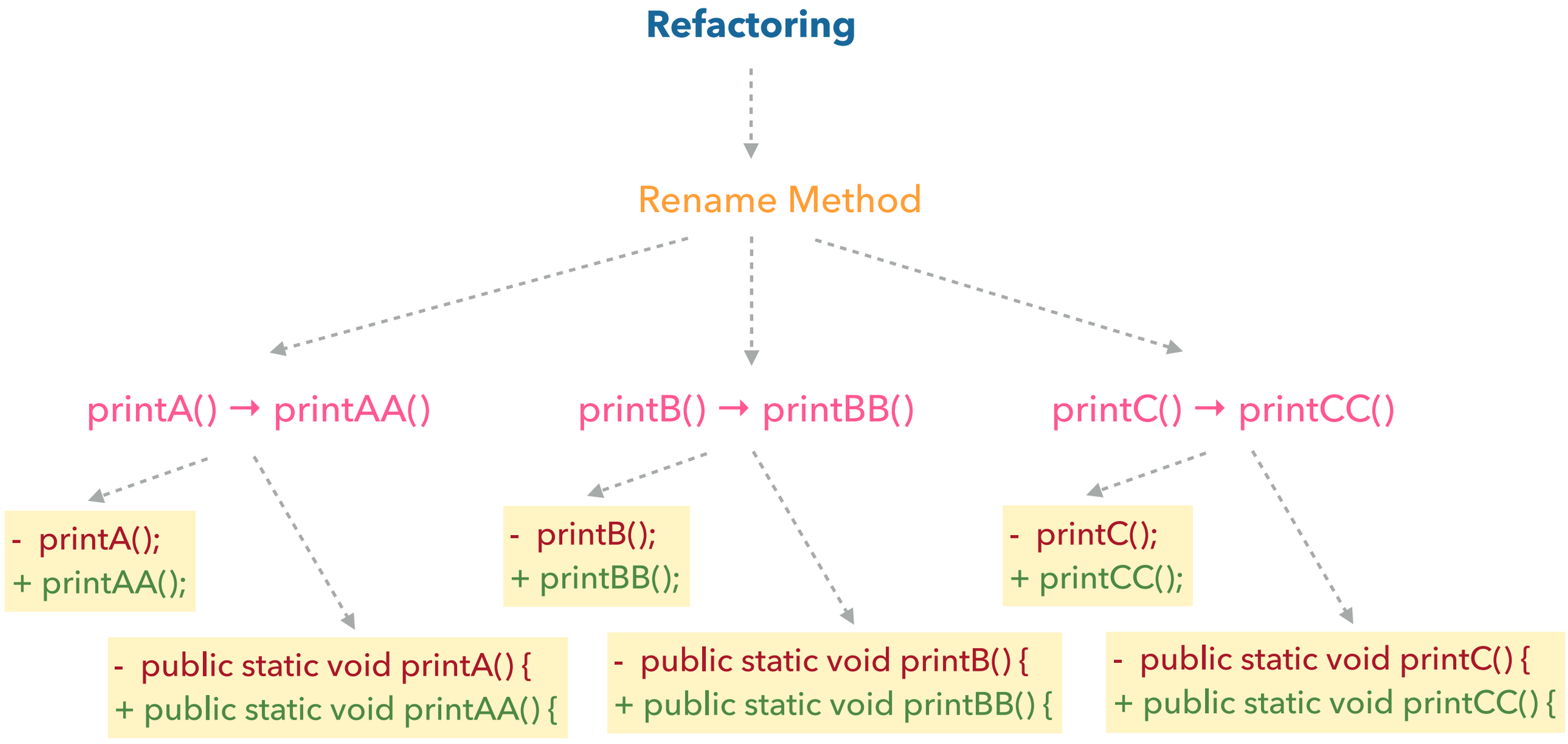
```
-  public static void printC() {
+ public static void printCC() {
```

Hello.java

```
+ public Hello(String name) {
+      this.name = name;
+ }
```

```
+ public void printHello() {
+      System.out.println("Hello " + name +
        ", How are you today?");
+ }
```

REFACTORING TREE RESULT



RELEVANCE CALCULATION

Main Purpose : Separate changes based on relevance

System Dependency Graphs

- Calculate **distance** between changes
- Setup threshold

SimpleCode.java

```
1  1
2  2   public class SimpleCode {
3  3       public static void main(String args[]) {
4      -        printA();
5      -        printB();
6      -        printC();
   4  +        printAA();
   5  +        printBB();
   6  +        printCC();
   7  +    Hello h = new Hello("Tua");
   8  +    h.printHello();
7  9       }
```

Hello.java

```
1  1
2  2   public class Hello {
3  3       private String name;
4      -
   4  +    public Hello(String name) {
   5  +        this.name = name;
   6  +    }
   7  +    public void printHello() {
   8  +        System.out.println("Hello " + name + "
   9  +    }
5  10  }
```

INPUT

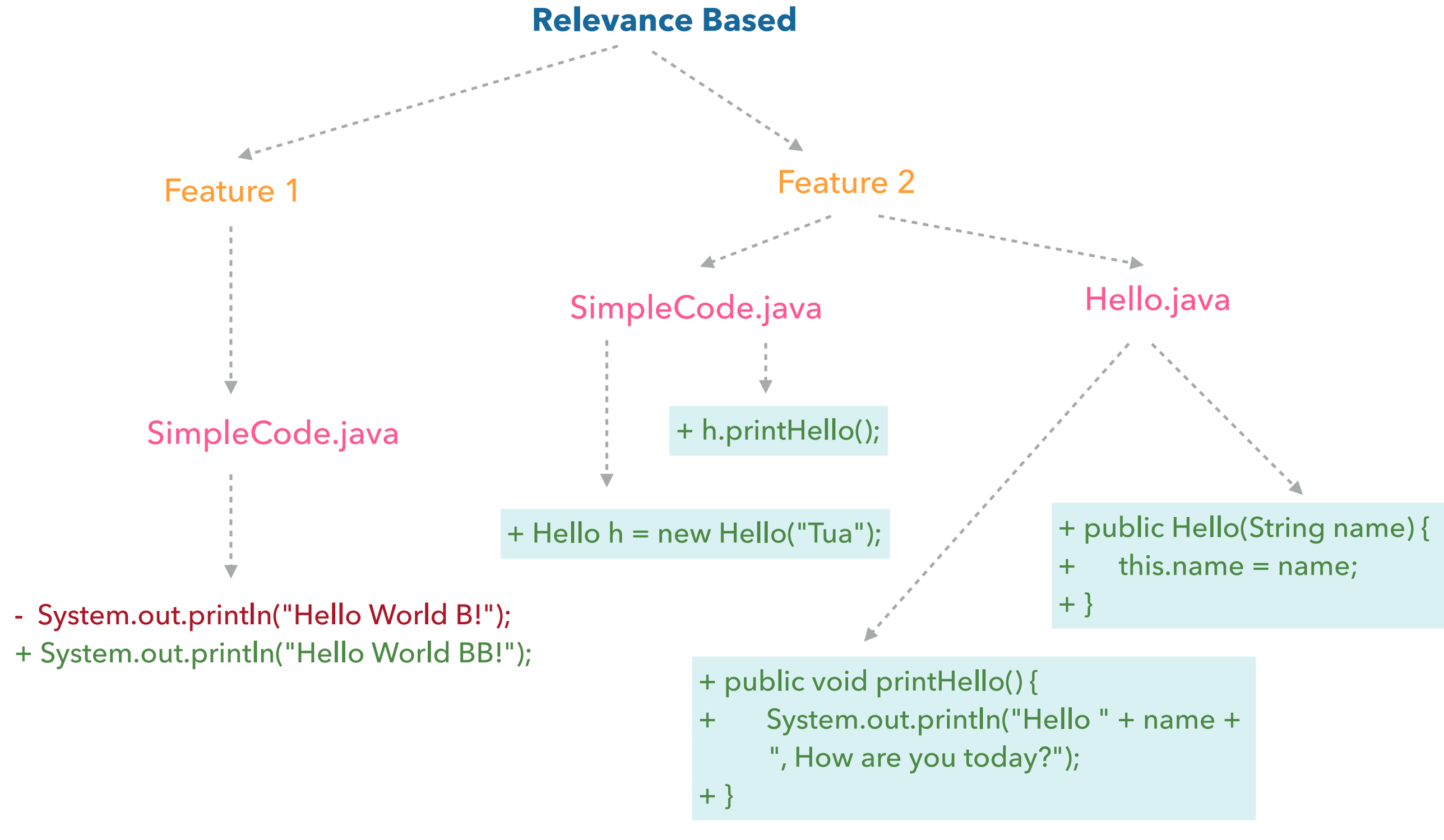
SimpleCode.java

```
-  printA();
+ printAA();
-  printB();
+ printBB();
-  printC();
+ printCC();
+ Hello h = new Hello("Tua");
+ h.printHello();
-  public static void printA() {
+ public static void printAA() {
-  public static void printB() {
+ public static void printBB() {
-  System.out.println("Hello World B!");
+ System.out.println("Hello World BB!");
-  public static void printC() {
+ public static void printCC() {
```

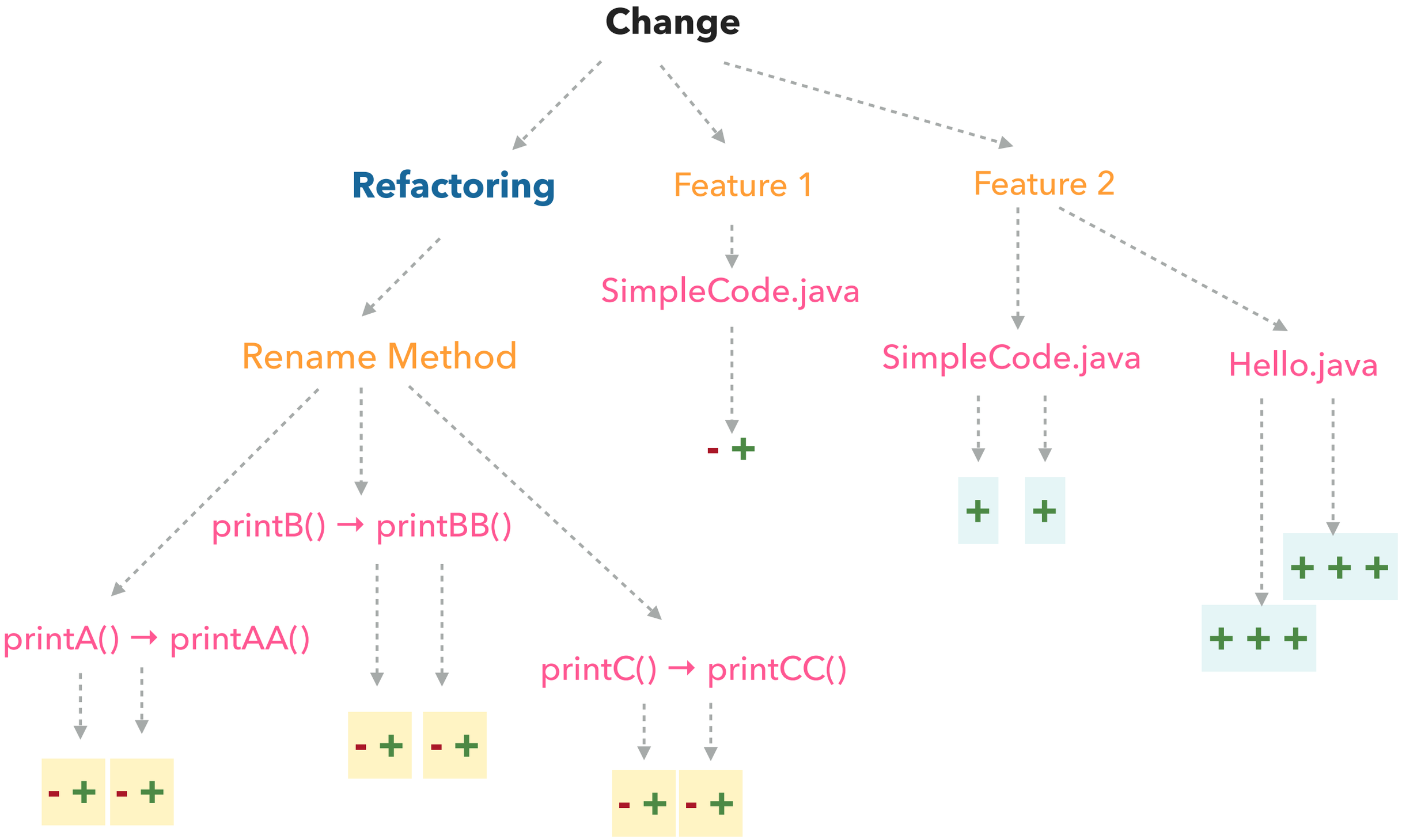
Hello.java

```
+ public Hello(String name) {
+       this.name = name;
+ }
+ public void printHello() {
+       System.out.println("Hello " + name +
          ", How are you today?");
+ }
```

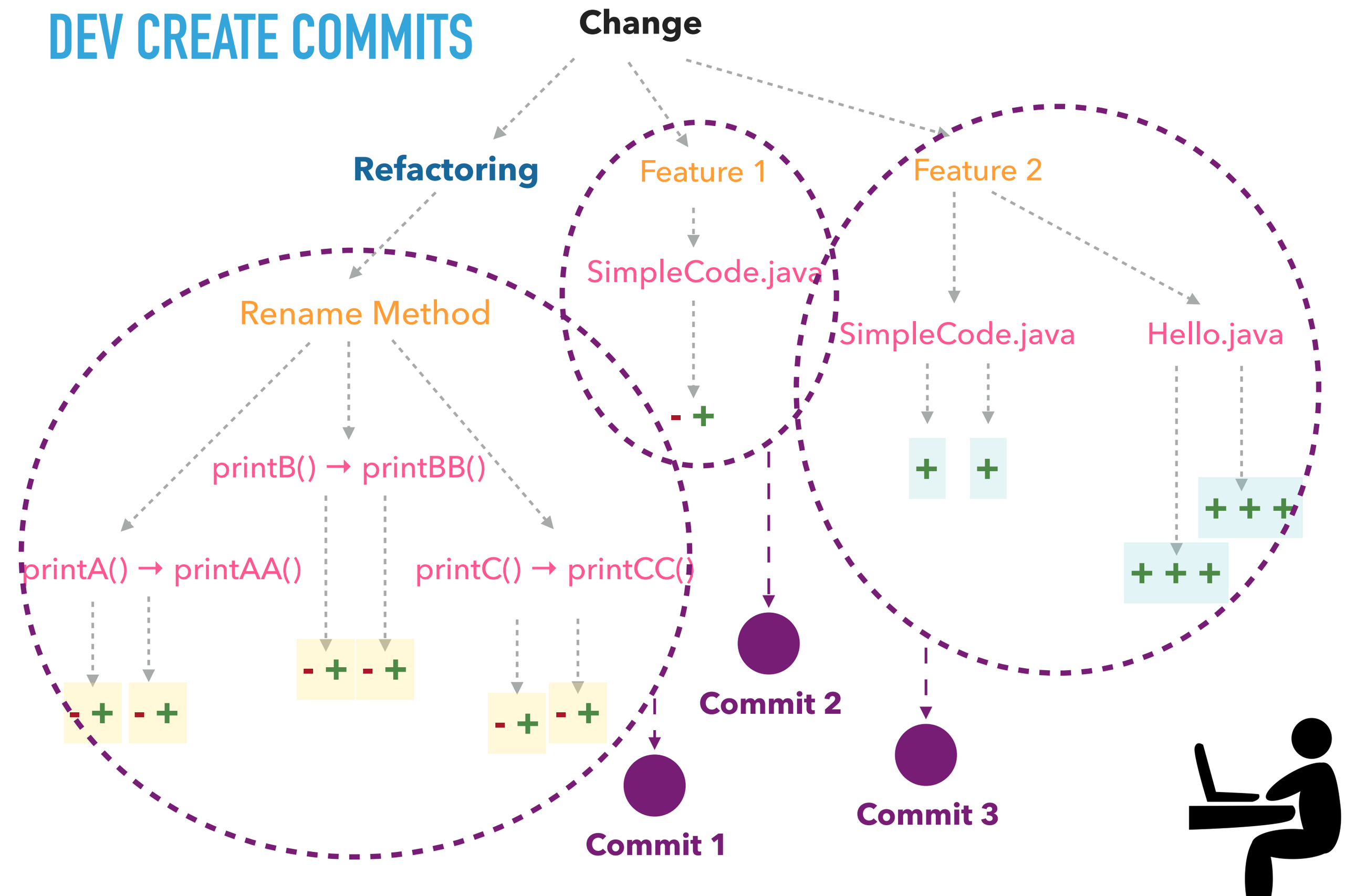
RELEVANCE BASED TREE RESULT



FINAL TREE

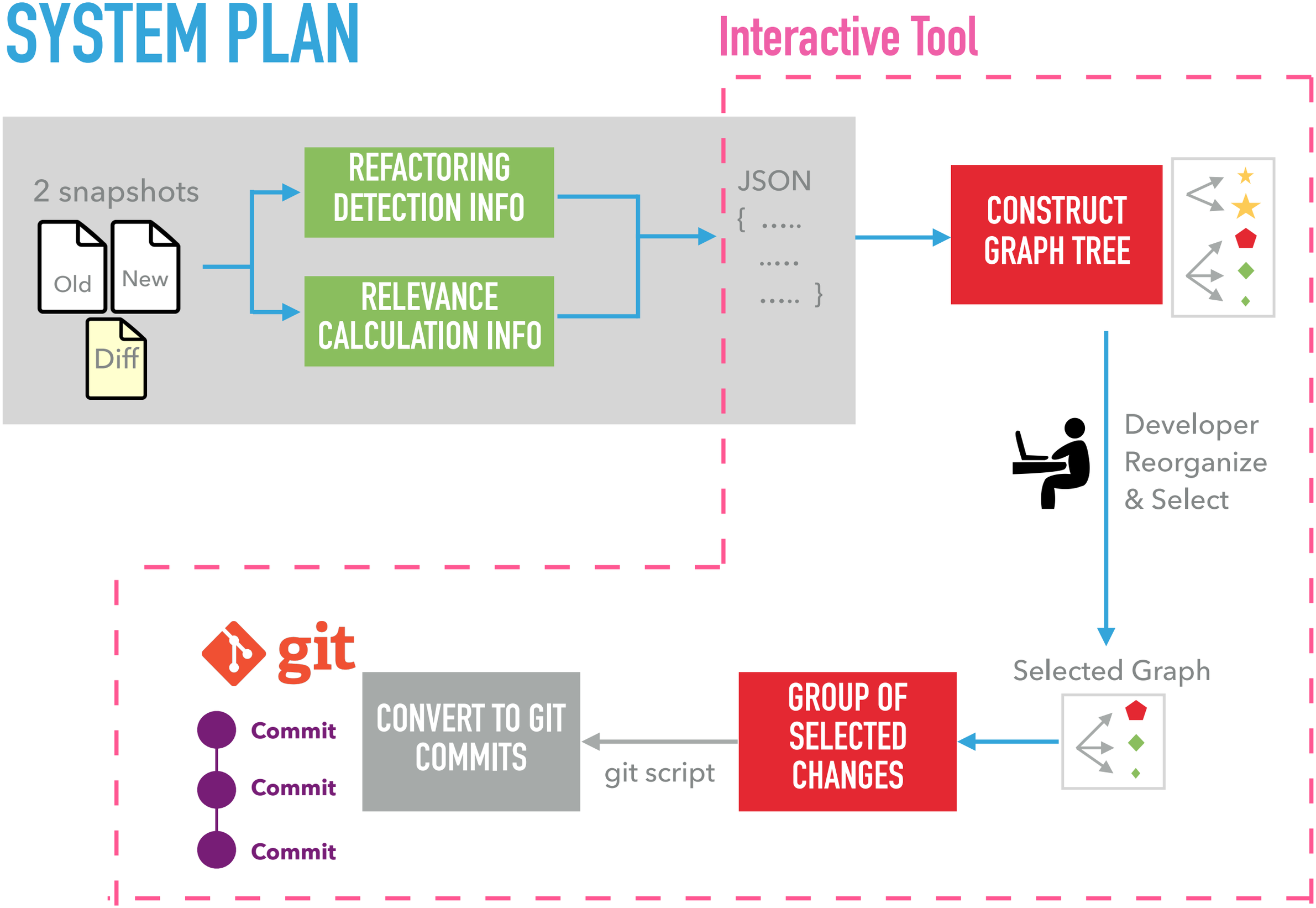

Change
Refactoring
Feature 1
Feature 2
Rename Method
SimpleCode.java
- +
SimpleCode.java
+ +
Hello.java
+ + +
+ + +
printB() → printBB()
printA() → printAA()
printC() → printCC()
- + - +
- + - +
- + - +

DEV CREATE COMMITS



TOOL

SYSTEM PLAN


Interactive Tool
2 snapshots
Old
New
Diff
REFACTORING DETECTION INFO
RELEVANCE CALCULATION INFO
JSON
{ .....
.....
..... }
CONSTRUCT GRAPH TREE
Developer Reorganize & Select
Selected Graph
GROUP OF SELECTED CHANGES
git script
CONVERT TO GIT COMMITS
git
Commit
Commit
Commit

3rd Problem ▾ | Switch Mode | Show File Detail

Changes

- Refactor
  - Rename Method
    - printA() -> printAA()
      - (DEF) SimpleCode.printA()
      - SimpleCode - printA();
    - printB() -> printBB()
      - (DEF) SimpleCode.printB()
      - SimpleCode - printB();
    - printC() -> printCC()
      - (DEF) SimpleCode.printC()
      - SimpleCode - printC();
- Feature 1
  - SimpleCode.java
    - System.out.println("Hello World B!");
- Feature 2
  - SimpleCode.java
    - Hello h = new Hello("Tua");
    - h.printHello();
  - Hello.java
    - Hello()
    - printHello()

SimpleCode.java

```
5   5       printB();
6   6       printC();
7   7   }
8           public static void printA() {
        8   public static void printAA() {
9   9       System.out.println("Hello World A!");
10  10  }
11  11  public static void printB() {
```

```
        7       Hello h = new Hello("Tua");
        8       h.printHello();
7   9   }
8           public static void printA() {
        10  public static void printAA() {
9   11      System.out.println("Hello World A!");
10  12  }
11          public static void printB() {
        13  public static void printBB() {
12          System.out.println("Hello World B!");
        14  System.out.println("Hello World BB!");
```

Commit

Print Selected Node

develop ▴ | Commit Message | Commit!

3rd Problem ▾ | Switch Mode | ☑ Show File Detail

Changes

- ▼ ☐ Refactor
  - ▼ ☐ Rename Method
    - ▼ ☑ printA() -> printAA()
      - ☑ (DEF) SimpleCode.printA()
      - ☑ SimpleCode - printA();
    - ▼ ☐ printB() -> printBB()
      - ☐ (DEF) SimpleCode.printB()
      - ☐ SimpleCode - printB();
    - ▼ ☐ printC() -> printCC()
      - ☐ (DEF) SimpleCode.printC()
      - ☐ SimpleCode - printC();
- ▼ ☐ Feature 1
  - ▼ ☐ SimpleCode.java
    - ☐ System.out.println("Hello World B!");
- ▼ ☐ Feature 2
  - ▼ ☐ SimpleCode.java
    - ☐ Hello h = new Hello("Tua");
    - ☐ h.printHello();
  - ▼ ☐ Hello.java
    - ☐ Hello()
    - ☐ printHello()

Staged Files Diff

SimpleCode.java

```
 1   1
 2   2  public class SimpleCode {
 3   3      public static void main(String args[]) {
 4          -      printA();
     4      +      printAA();
 5          -      printB();
     5      +      printBB();
 6          -      printC();
     6      +      printCC();
     7      +      Hello h = new Hello("Tua");
     8      +      h.printHello();
 7   9      }
 8          -  public static void printA() {
     10     +  public static void printAA() {
 9   11         System.out.println("Hello World A!");
 10  12     }
 11         -  public static void printB() {
     13     +  public static void printBB() {
 12         -      System.out.println("Hello World B!");
     14     +      System.out.println("Hello World BB!");
```

Commit

Print Selected Node

develop ▴ | Commit Message | Commit!

EVALUATION

1. RQ 1: Is Tree easier to understand?
   - **Time** spent for tangled change understanding
   - **Users feeling** when they try to understand changes
2. RQ 2: Is Tree easier to separate changes?
   - **Time** spent for tangled change separation
   - **Users feeling** when they try to separate changes
   - **Correctness** of task level changes grouping
3. RQ 3: Is tool's usability good enough?
   - **Users feeling** when they use the tool

COMPARATIVE

TREE VS LIST

CONTROLLED EXPERIMENT (UNDERSTANDING & SEPARATION SESSIONS) + QUESTIONNAIRE (FEELING & OPINION SCALE & COMMENTS)

Baseline = File list + Refactoring detection info + Relevance info

3rd Problem ▾ Switch Mode Show File Detail

Changes

- [x] **Select All Files**
- [x] SimpleCode.java
- [x] Hello.java

Staged Files Diff

- [ ] SimpleCode.java

```
   1   1
   2   2  public class SimpleCode {
   3   3      public static void main(String args[]) {
- [x] 4          printA();
+ [x]     4      printAA();
- [ ] 5          printB();
+ [ ]     5      printBB();
- [ ] 6          printC();
+ [ ]     6      printCC();
+ [ ]     7      Hello h = new Hello("Tua");
+ [ ]     8      h.printHello();
   7   9  }
- [x] 8      public static void printA() {
+ [x]     10     public static void printAA() {
   9   11         System.out.println("Hello World A!");
   10  12 }
- [ ] 11     public static void printB() {
+ [ ]     13     public static void printBB() {
- [ ] 12         System.out.println("Hello World B!");
```

Commit

Print Selected Node

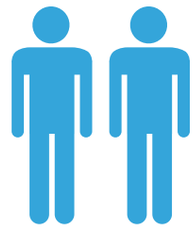
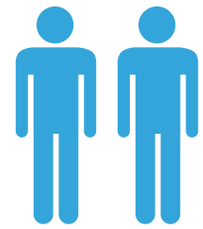
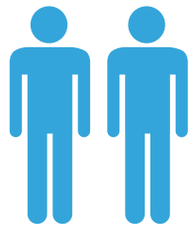
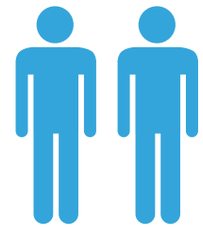
develop ▲ | Commit Message | Commit!

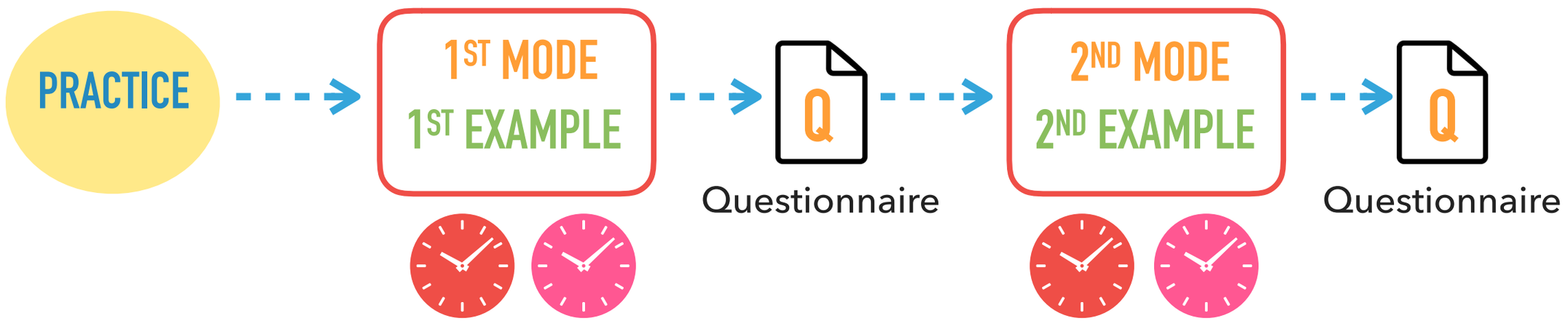
STEP

8 Programmers = Subject

Random order

- 2 Mode
  (Tree VS File List)
- 2 Examples

	Tree Mode First	List Mode First
Example 1 First		
Example 2 First		

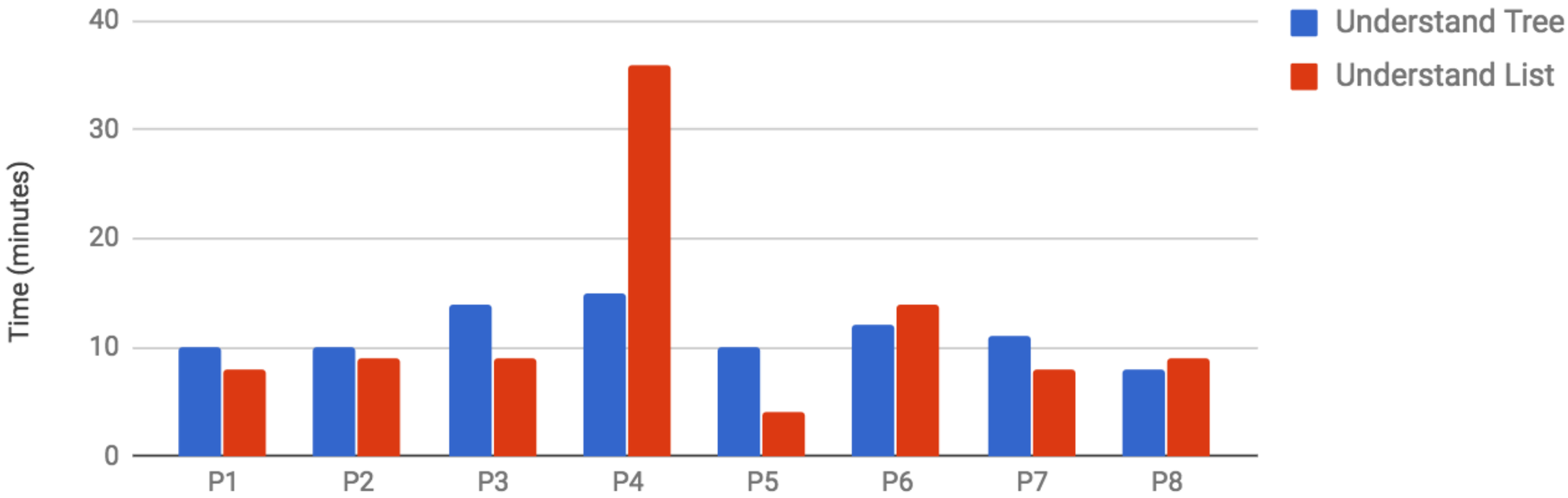


CONTROLLED EXPERIMENT

RESULTS

TIME SPENT TO UNDERSTAND CHANGES (MINUTES)

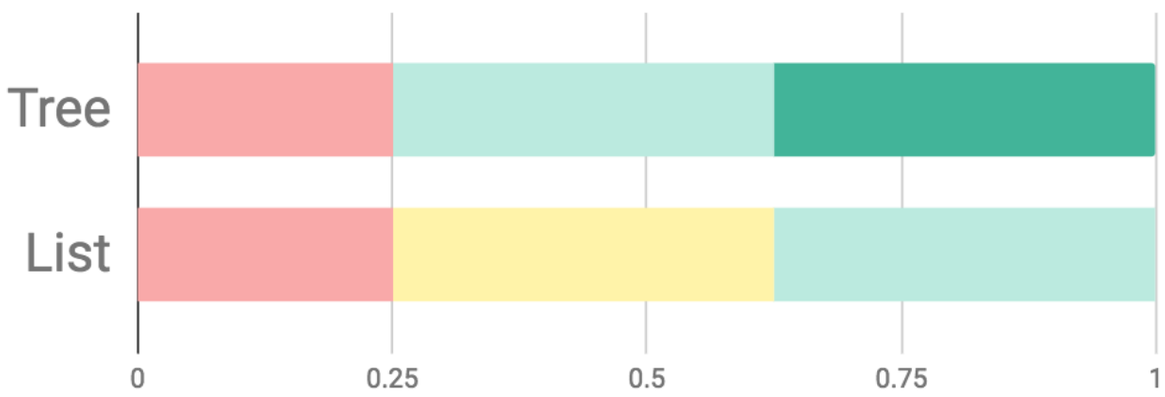
	Understand	Separate	Total
Tree	11.25	7.38	18.63
List	12.13	10.00	22.13



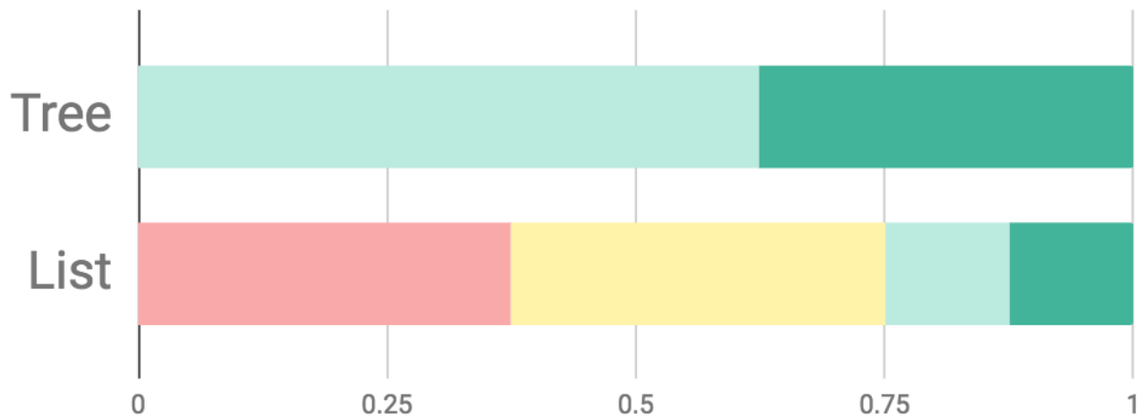
- Time spent to understand tangled changes using informative tree visualization is not much different from ordinary file list visualization

RQ 1: IS TREE EASIER TO UNDERSTAND ?

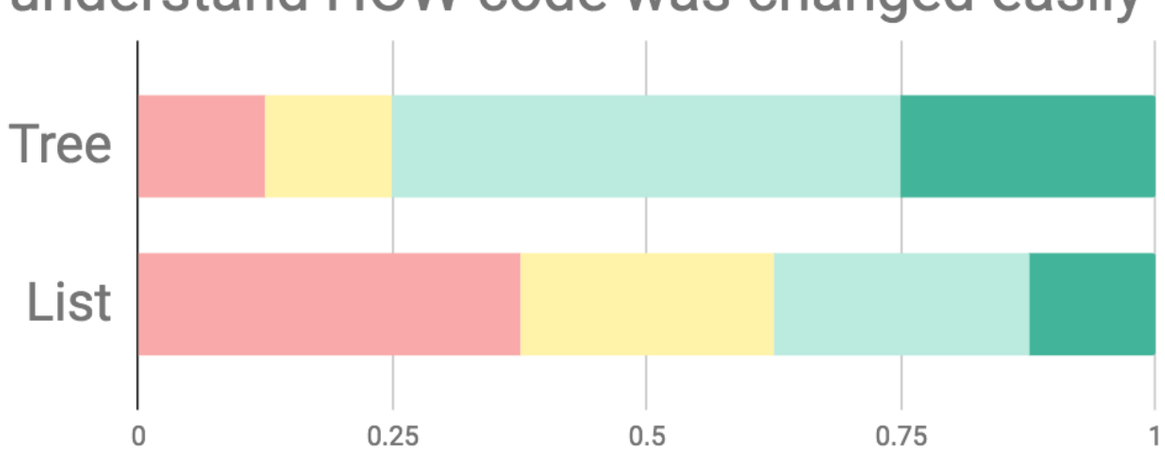
see OVERVIEW of change easily



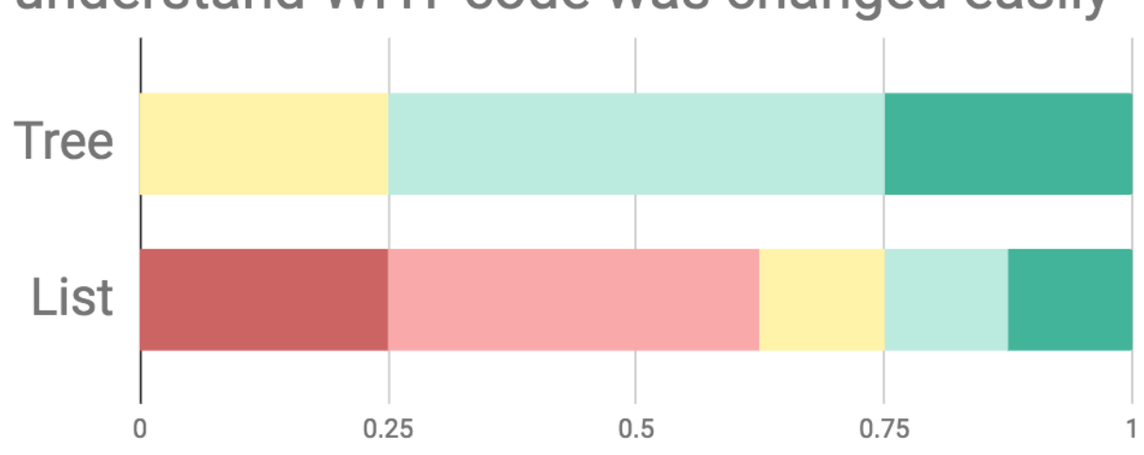
understand PURPOSE of each sub task easily



understand HOW code was changed easily



understand WHY code was changed easily



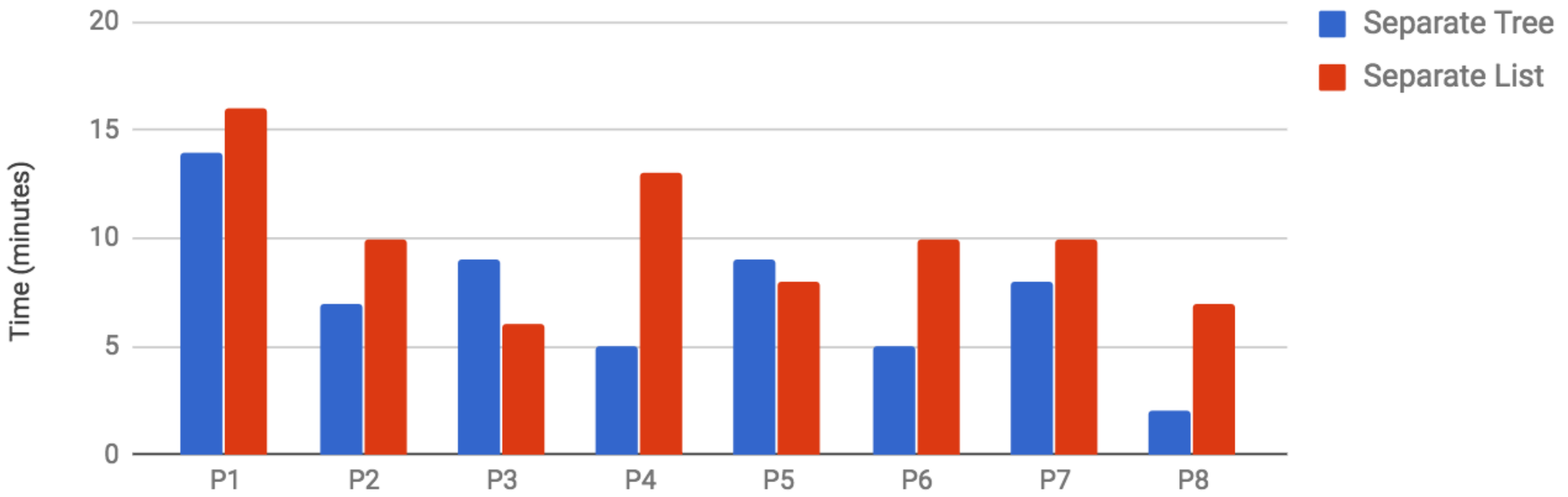
Strongly disagree Disagree Neither agree nor disagree Agree Strongly agree

- Users felt that understanding tangled change using informative tree visualization is easier than using ordinary file list visualization.

RQ 2: IS TREE EASIER TO SEPARATE CHANGES ?

TIME SPENT TO SEPARATE CHANGES (MINUTES)

	Understand	Separate	Total
Tree	11.25	7.38	18.63
List	12.13	10.00	22.13



- Time spent to separate tangled changes using informative tree visualization is faster than using ordinary file list visualization

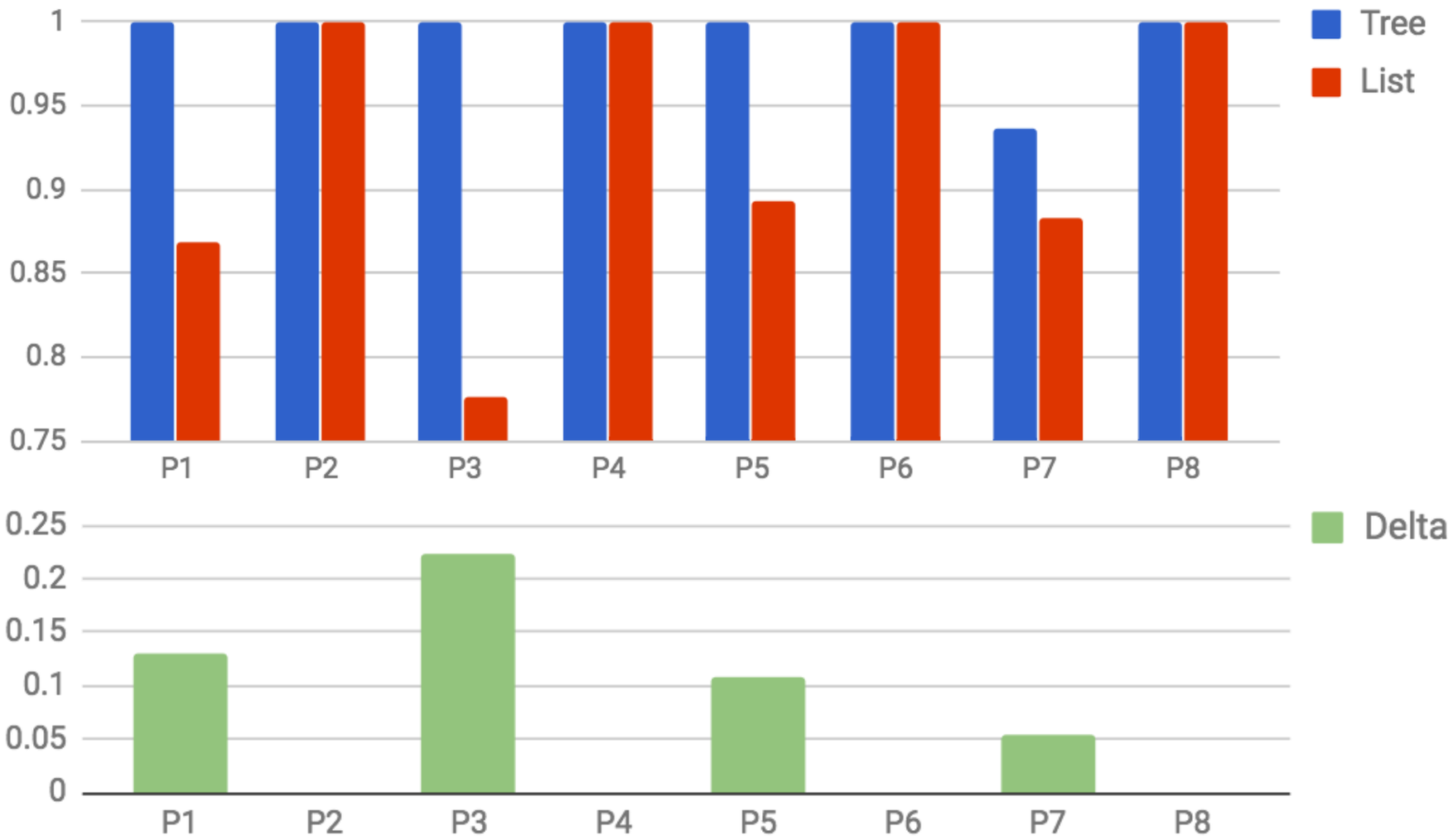
RQ 2: IS TREE EASIER TO SEPARATE CHANGES ?

- Users felt when separating tangled change using informative tree visualization is easier than using ordinary file list visualization

RQ 2: IS TREE EASIER TO SEPARATE CHANGES ?

GROUPING PRECISION

	Precision	Recall	Rand Index
Tree	0.99	0.94	0.97
List	0.92	0.91	0.93

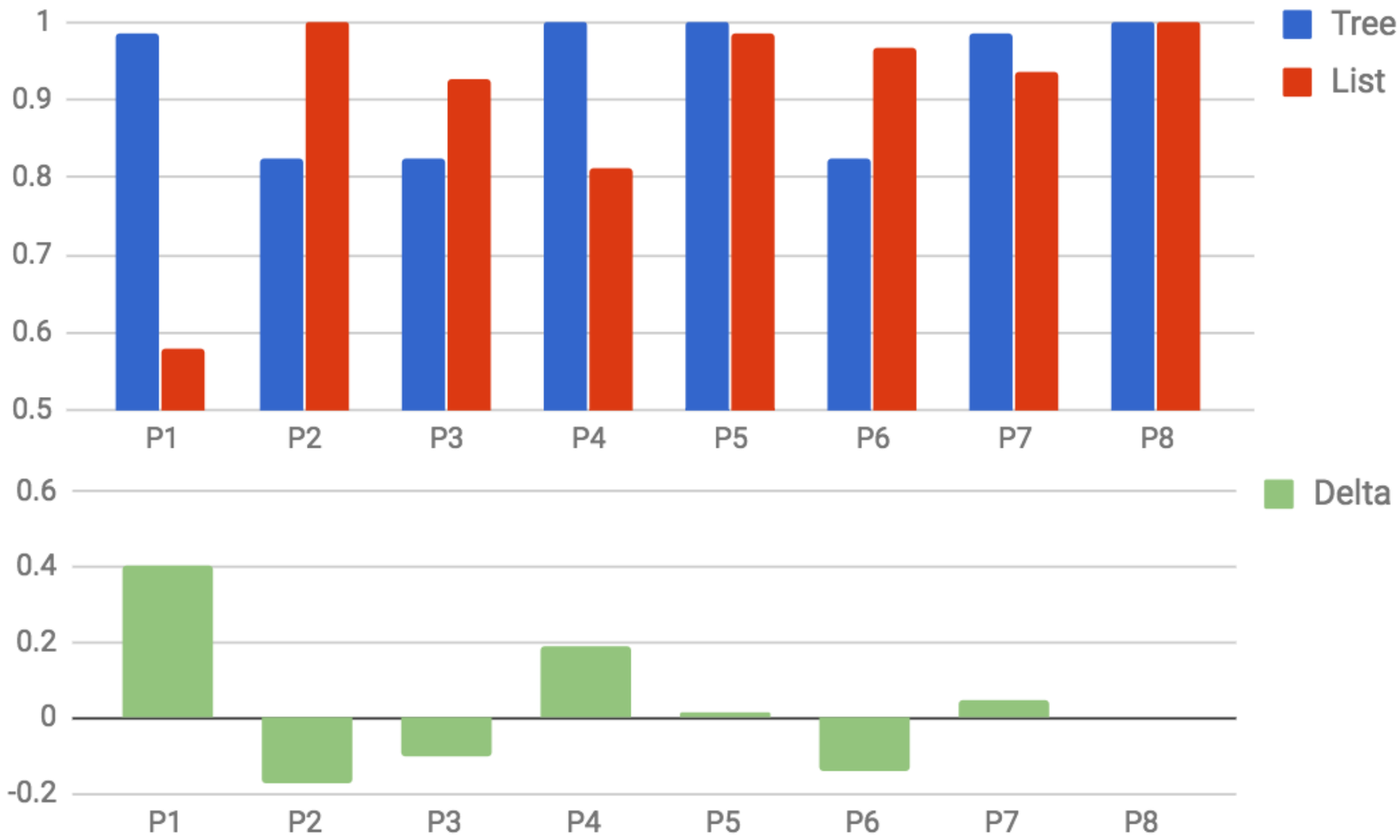


- Precision of grouping accuracy using informative tree visualization is better than ordinary file list visualization

RQ 2: IS TREE EASIER TO SEPARATE CHANGES ?

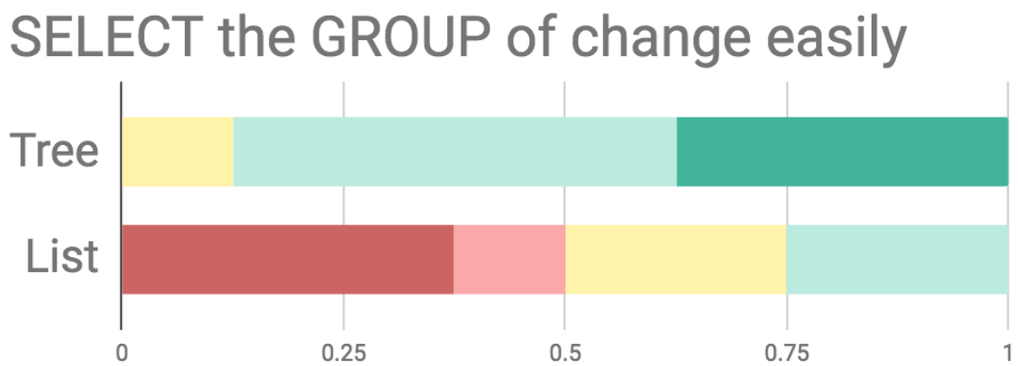
GROUPING RECALL

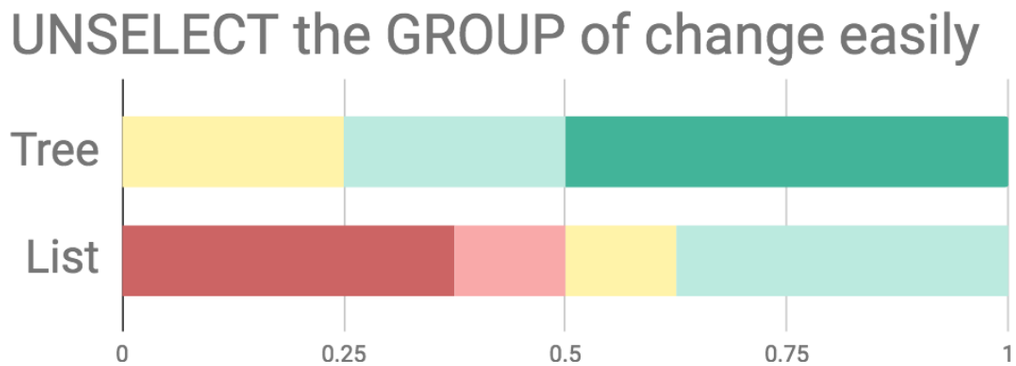
	Precision	Recall	Rand Index
Tree	0.99	0.94	0.97
List	0.92	0.91	0.93

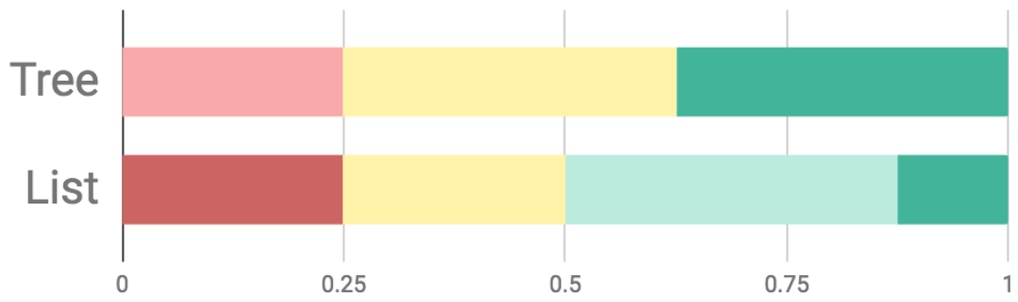


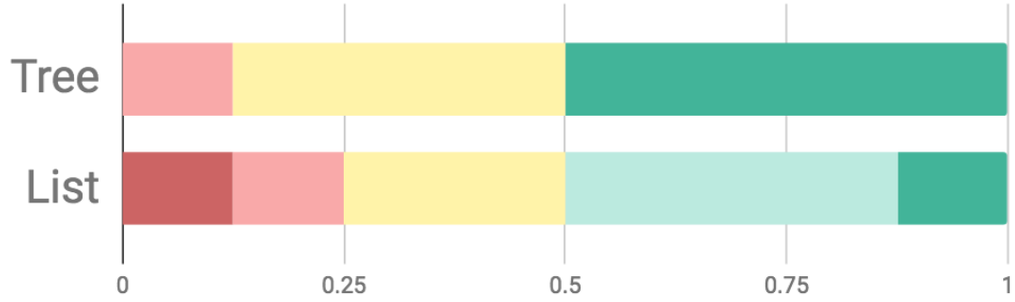
- Recall of grouping accuracy using informative tree visualization is better than ordinary file list visualization

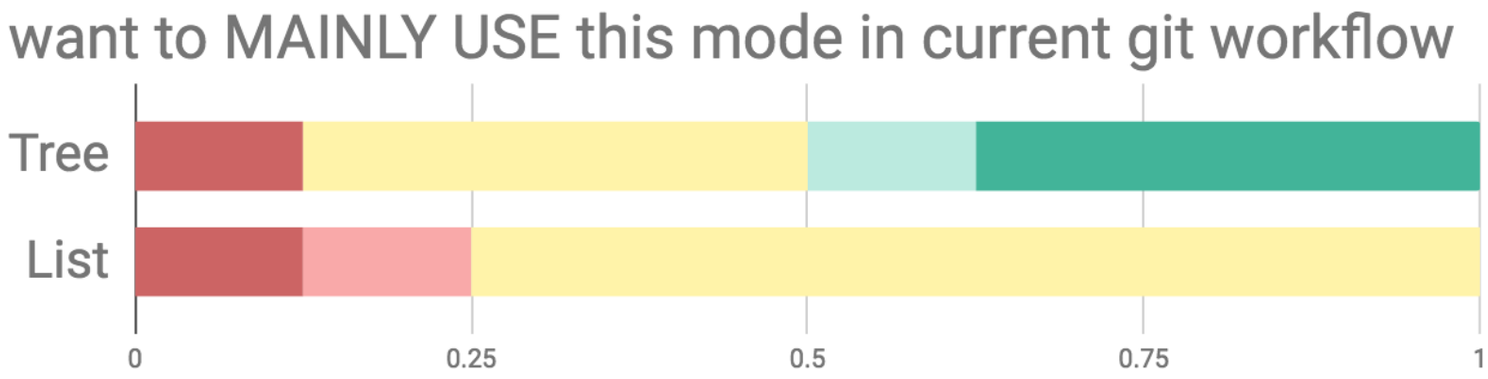
RQ 3: IS TOOL'S USABILITY GOOD ENOUGH ?











- Users felt that usability in informative tree visualization mode is better than ordinary file list visualization mode

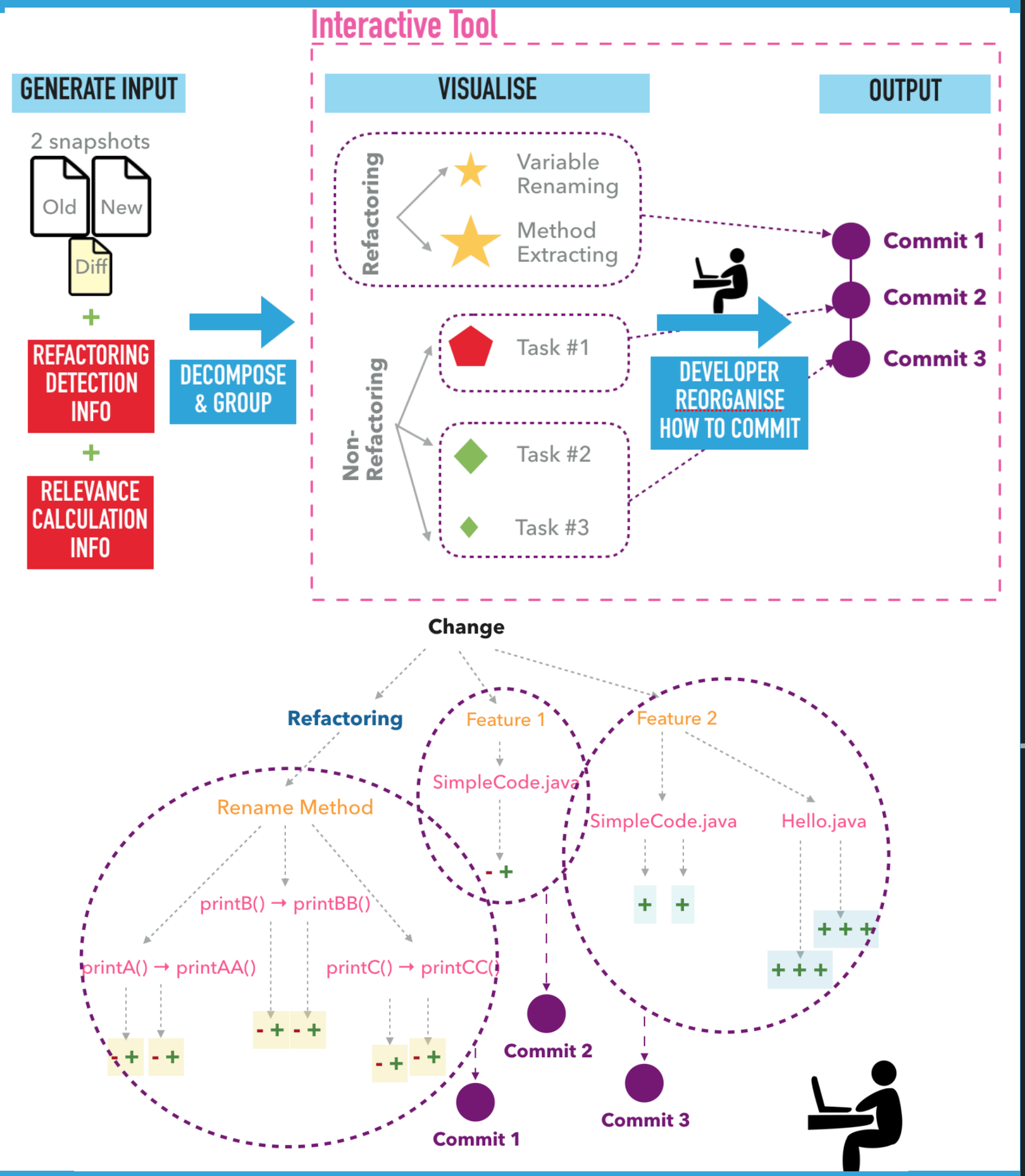
CONCLUSION

CONCLUSION

- Using an informative tree to visualize the tangled change
- Refactoring detection + Relevance calculation
  = Tangled change decomposition
- **Evaluation** : Compared to ordinary file list based
  - Reduce averaged time spent to understand changes
    12.13 min → 11.25 min (7%)
  - Reduce averaged time spent to separate changes
    10 min → 7.38 min (26%)
  - Increase grouping accuracy precision 0.92 → 0.99
- **RQ Answers :** Tree is **easier** to understand + separate tangled change & Tool's usability is **good enough**

FUTURE WORK

- Evaluate with more participants $\Longrightarrow$ Statistical significant
- Real world projects testing
- Grouping algorithm $\Longrightarrow$ Support more cases
- Tool maturity



Thank you for your kind attention

Q&A