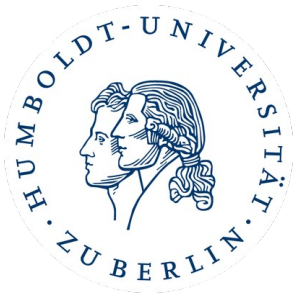
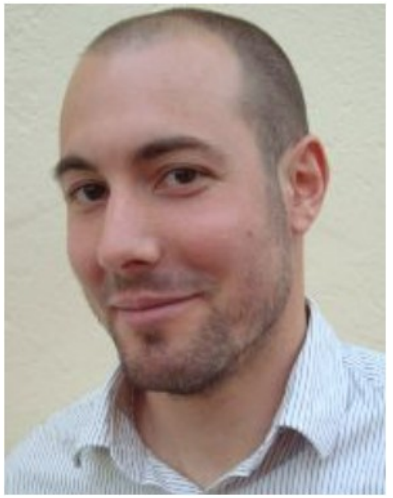


Quantifying the Potential to Automate the Synchronization of Variants in Clone-and-Own

Alexander Schultheiß

In collaboration with

Paul Maximilian Bittner

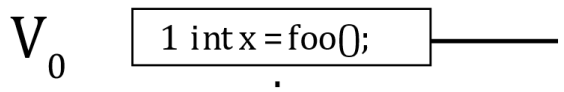


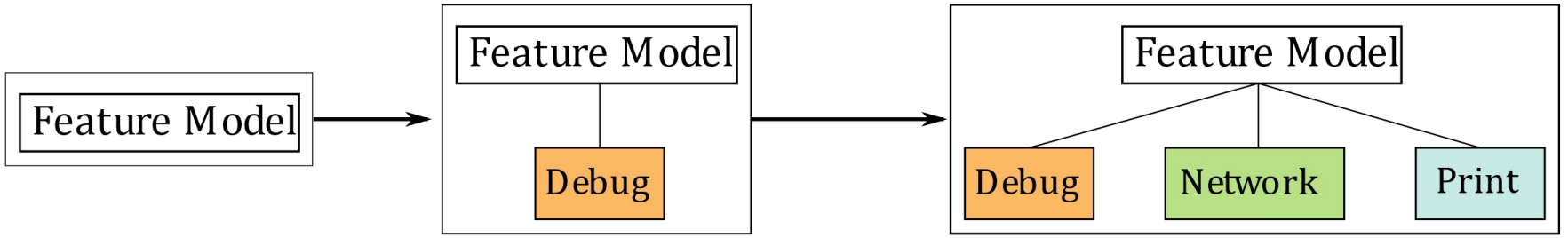
Timo Kehrer

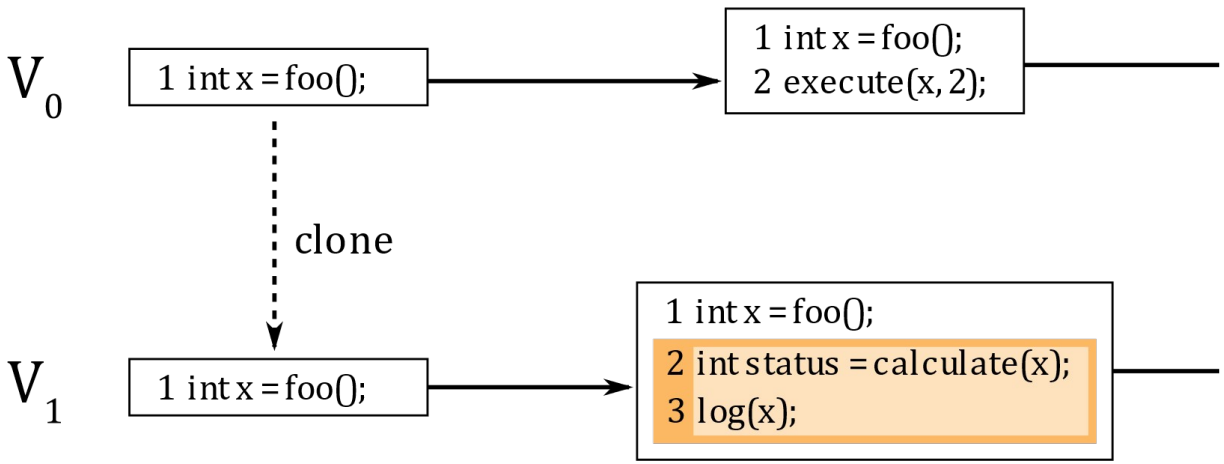
Thomas Thüm

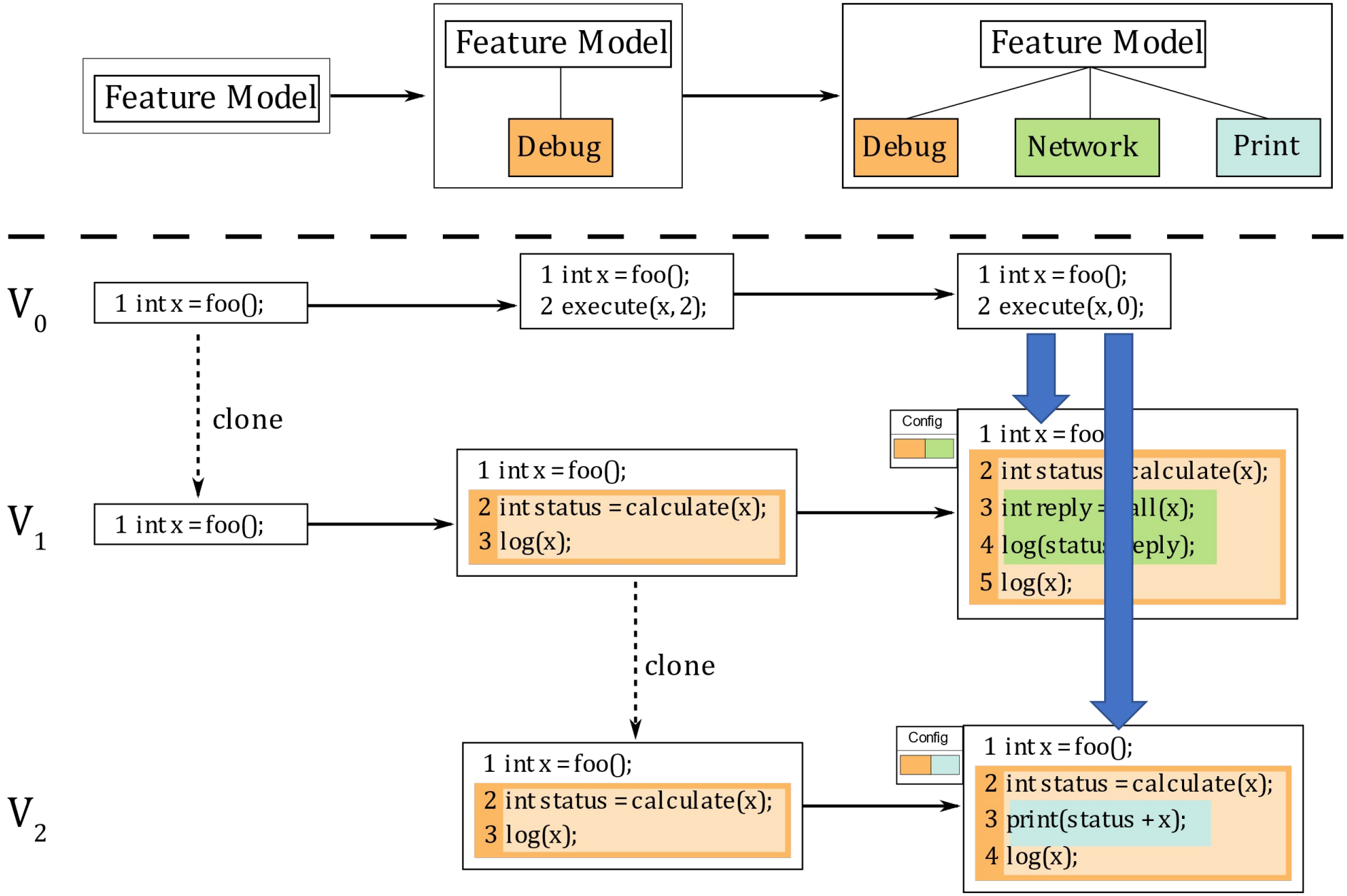
Clone-and-Own Development

Variants are created by copying and adapting existing variant

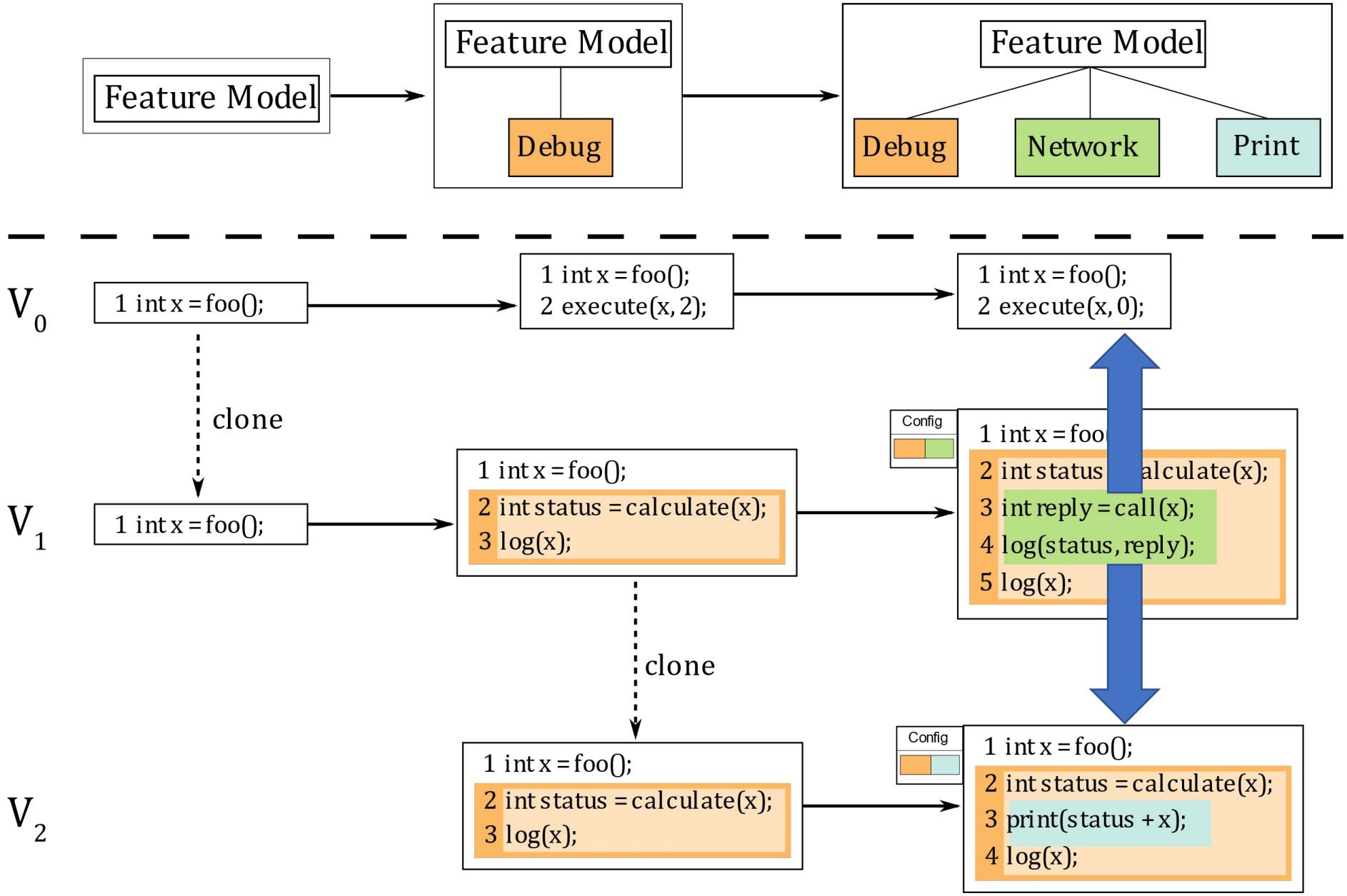



Feature Model
Feature Model
Debug
Feature Model
Debug
Network
Print


V0
1 int x = foo();
1 int x = foo();
2 execute(x, 2);
clone
V1
1 int x = foo();
1 int x = foo();
2 int status = calculate(x);
3 log(x);



Where to apply a change?



Which changes to propagate?

Quantifying the Automation Potential

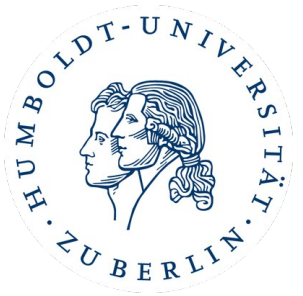
Goals of the study

Goals:

- Simulate automated change propagation
- Evaluate applicability and correctness
- Assess the benefit of knowledge about features

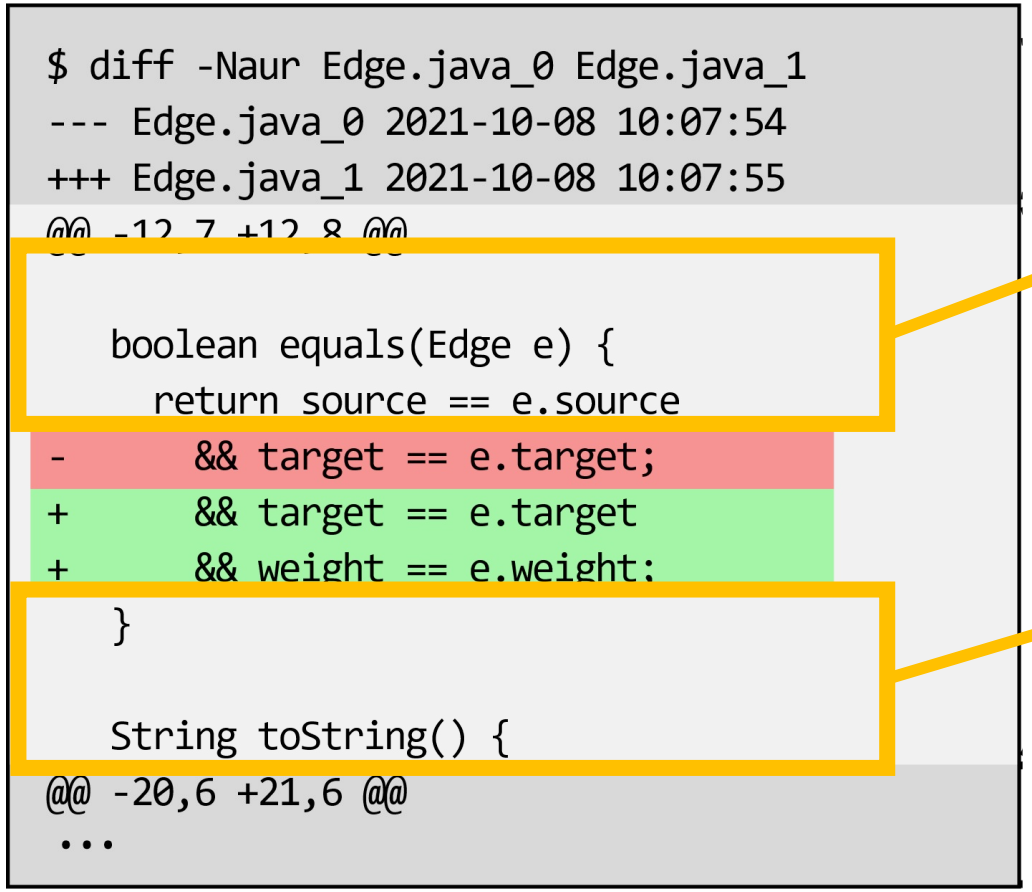
What we need:

- Change propagation technique
- Software variants
- History of the variants



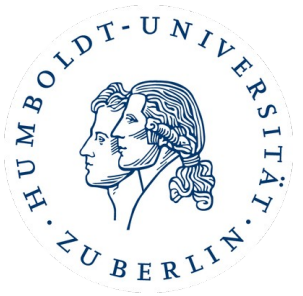
Change Propagation with *Diff* and *Patch*

Patch applies changes based on a *diff*



```
$ diff -Naur Edge.java_0 Edge.java_1
--- Edge.java_0 2021-10-08 10:07:54
+++ Edge.java_1 2021-10-08 10:07:55
@@ -12,7 +12,8 @@

    boolean equals(Edge e) {
      return source == e.source
-         && target == e.target;
+         && target == e.target
+         && weight == e.weight;
    }

    String toString() {
@@ -20,6 +21,6 @@
 ...
```

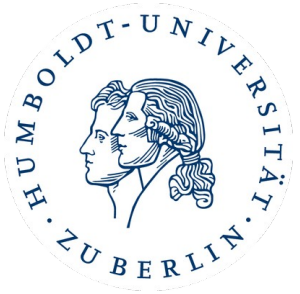
```
11  ...
12
13  boolean equals(Edge e) {
14    return source == e.source
15        && target == e.target;
16  }
17
18  String getLabel() {
19  ...
```



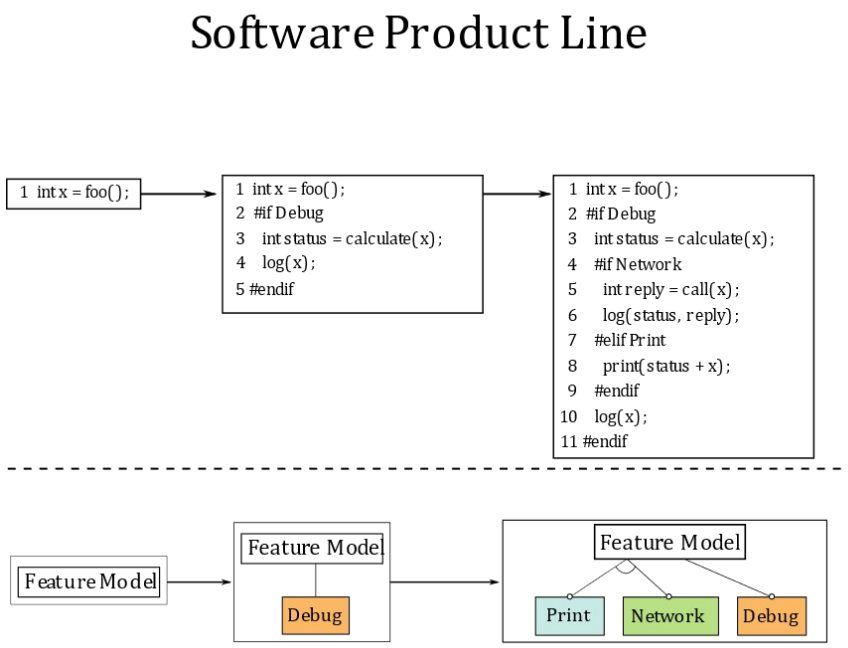
Variants and their histories

We consider the evolution of variants in BusyBox

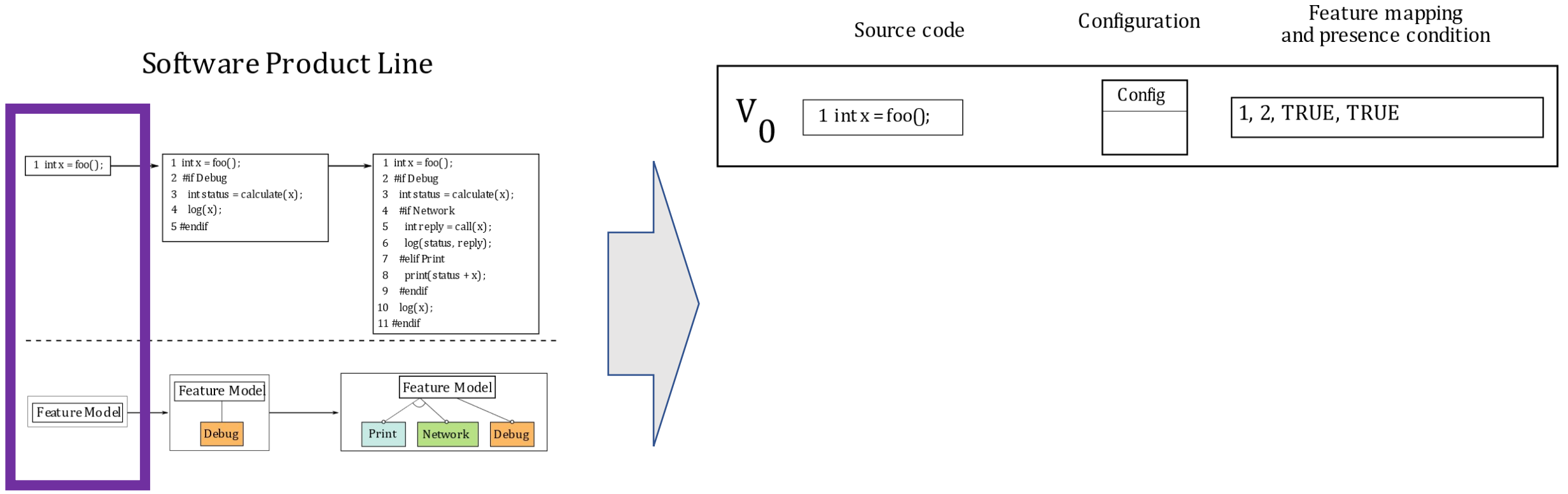
Software product line development



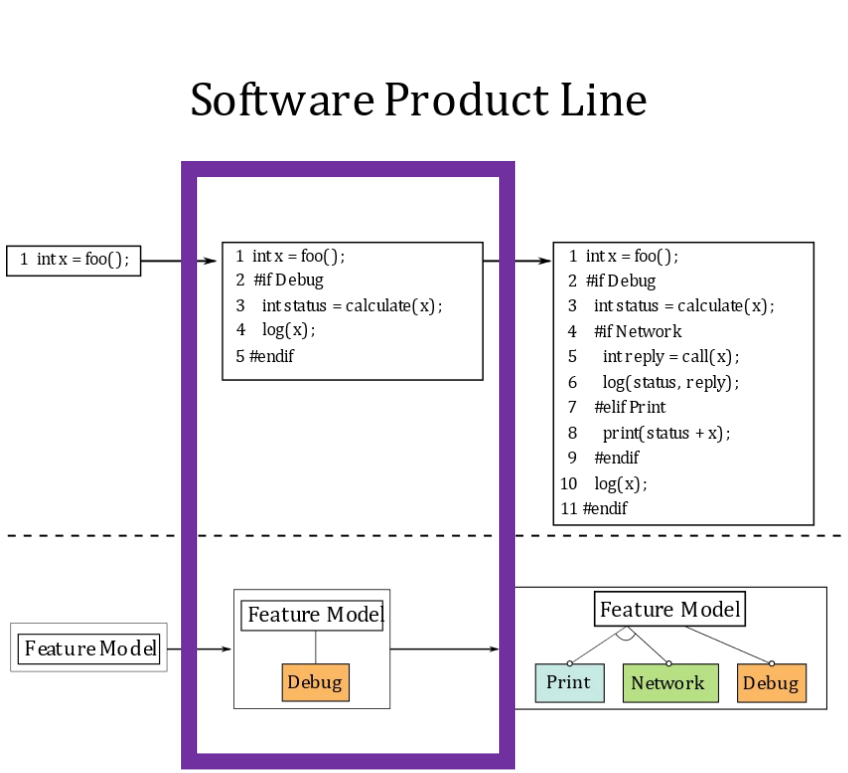
We can analyze the product line…

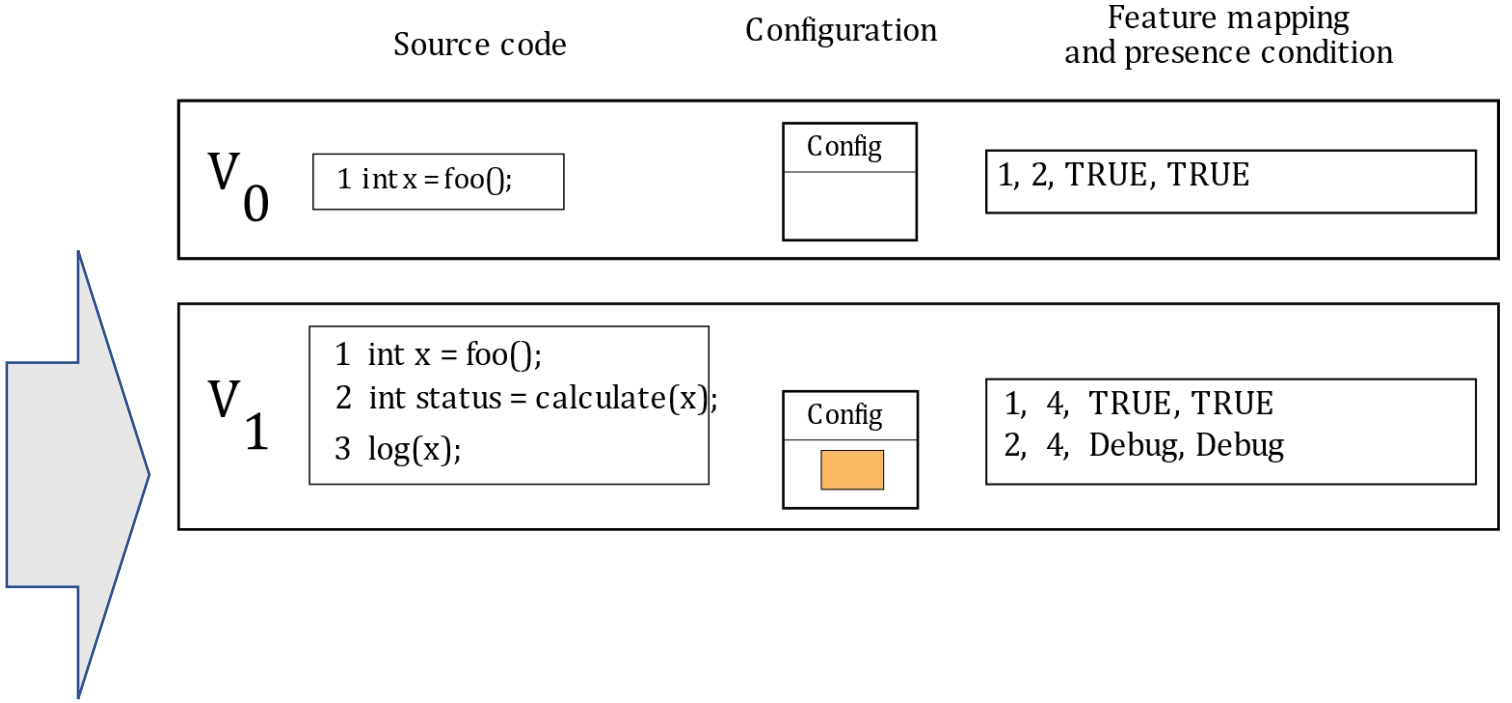


… to generate variants…

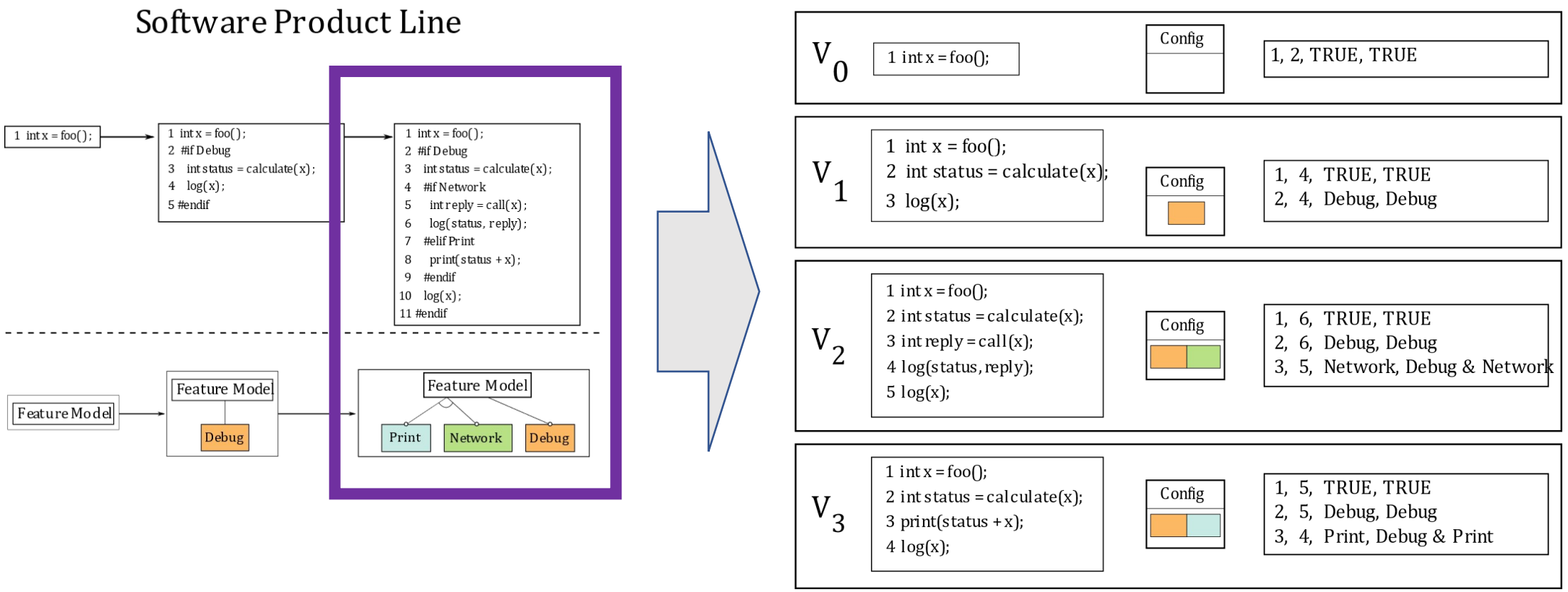


… to generate variants for each revision

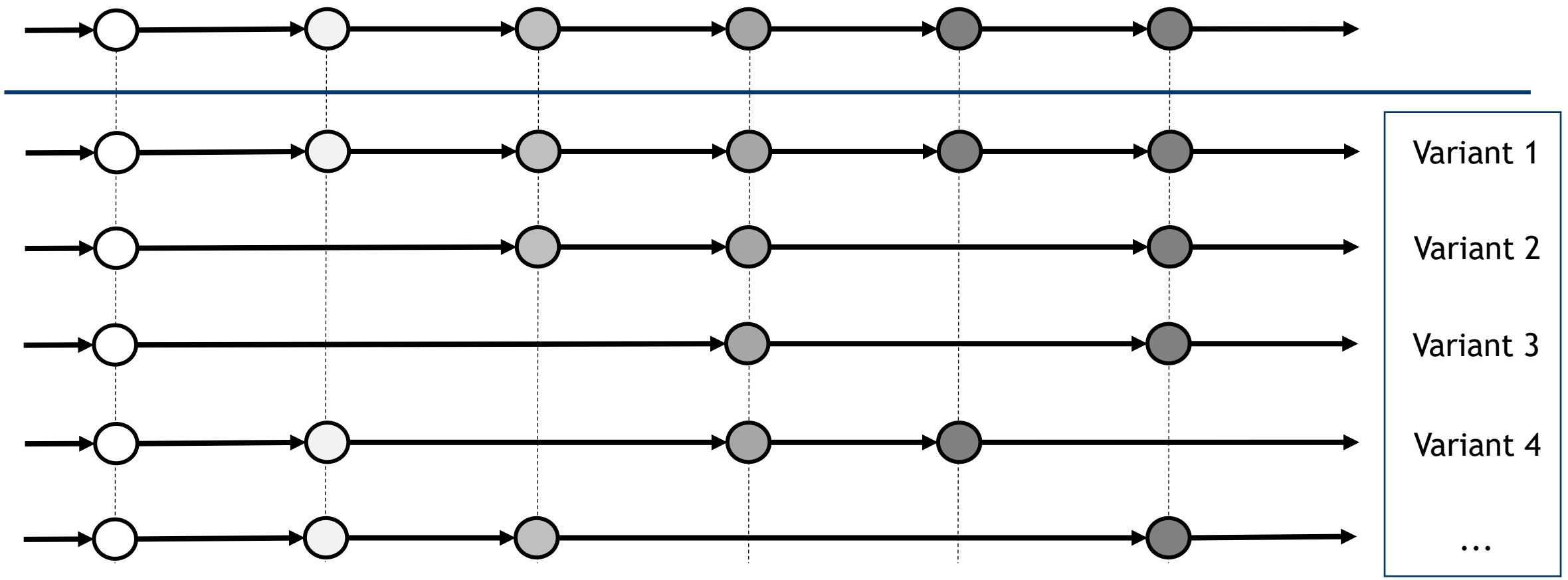




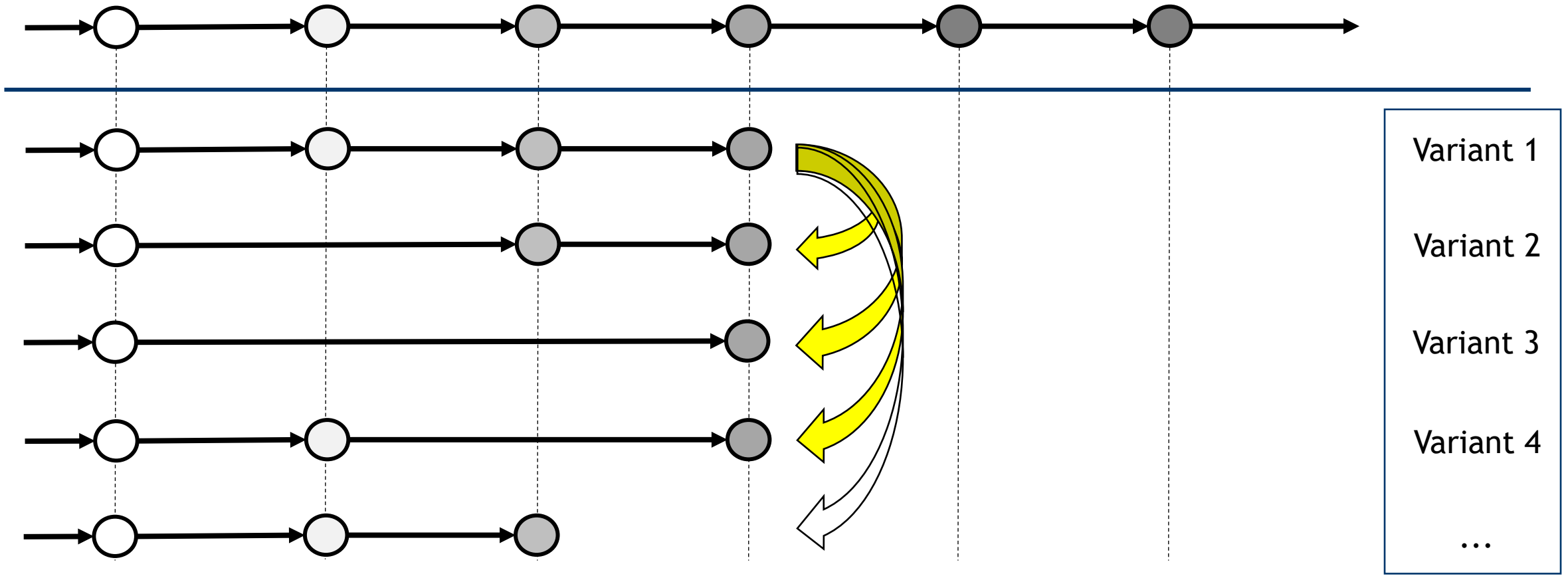
… to generate variants for each revision



We can simulate a history

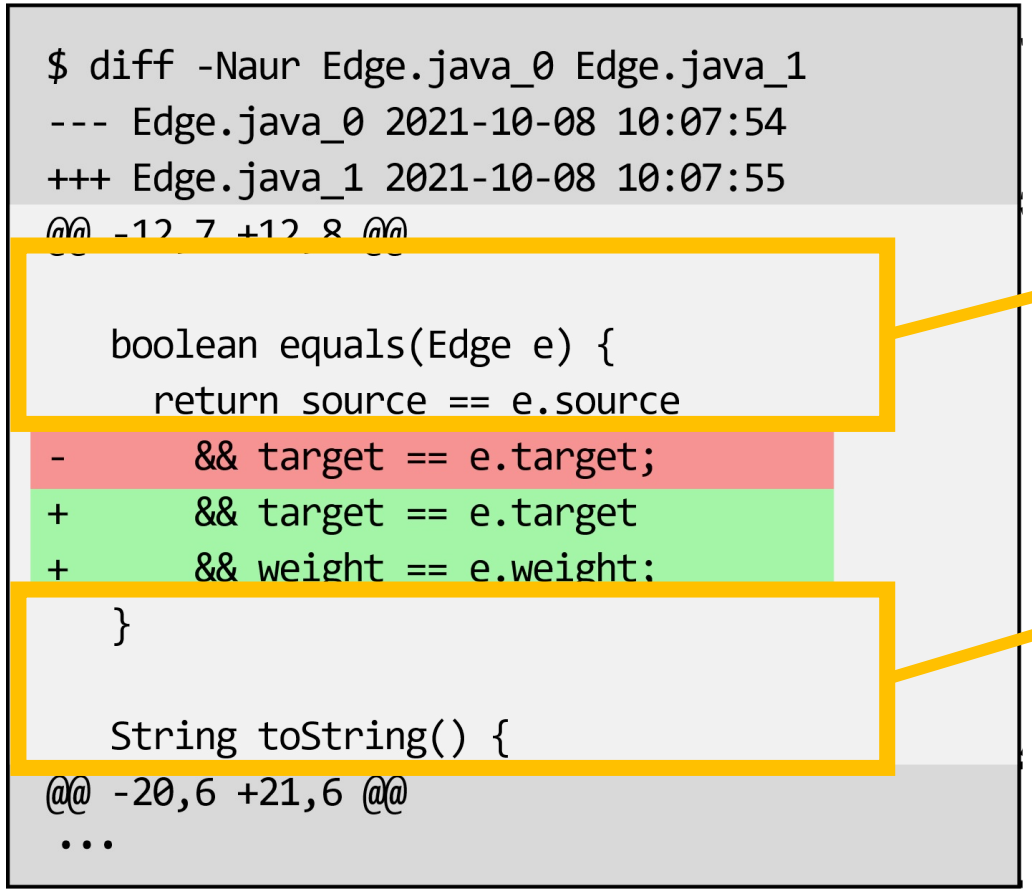


We can conduct our study

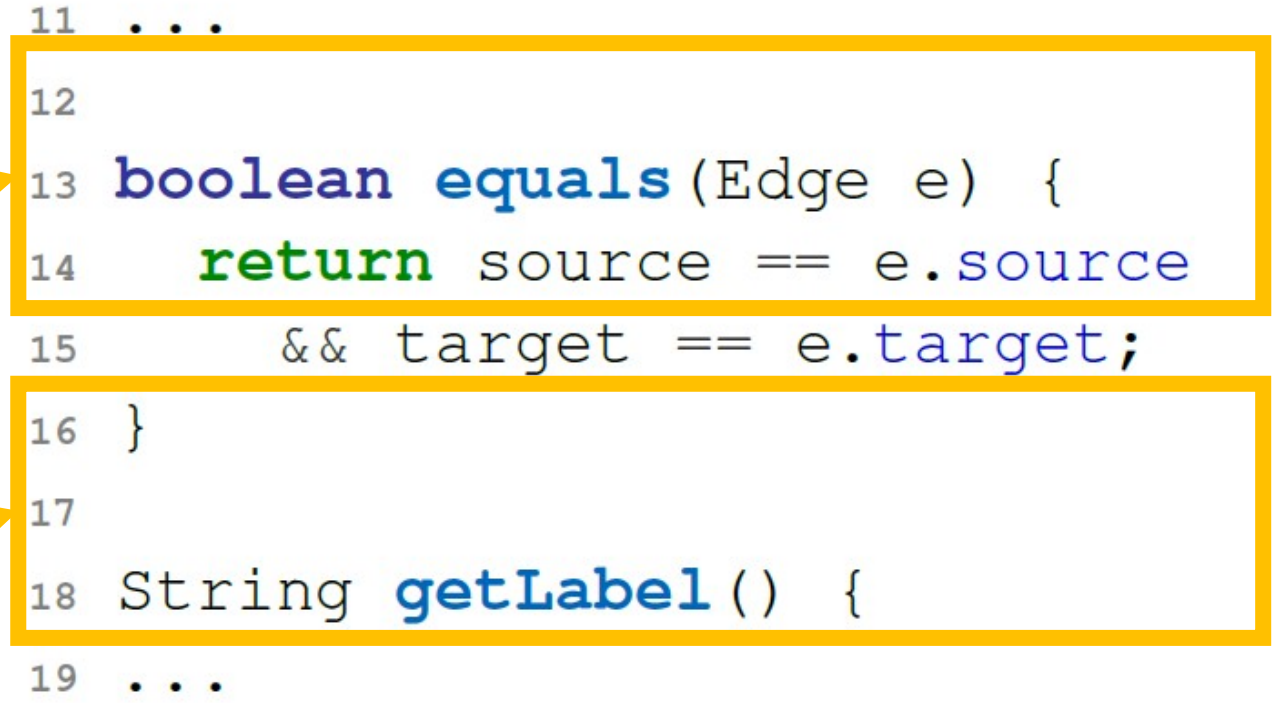


Results

RQ1: How often can changes be applied?

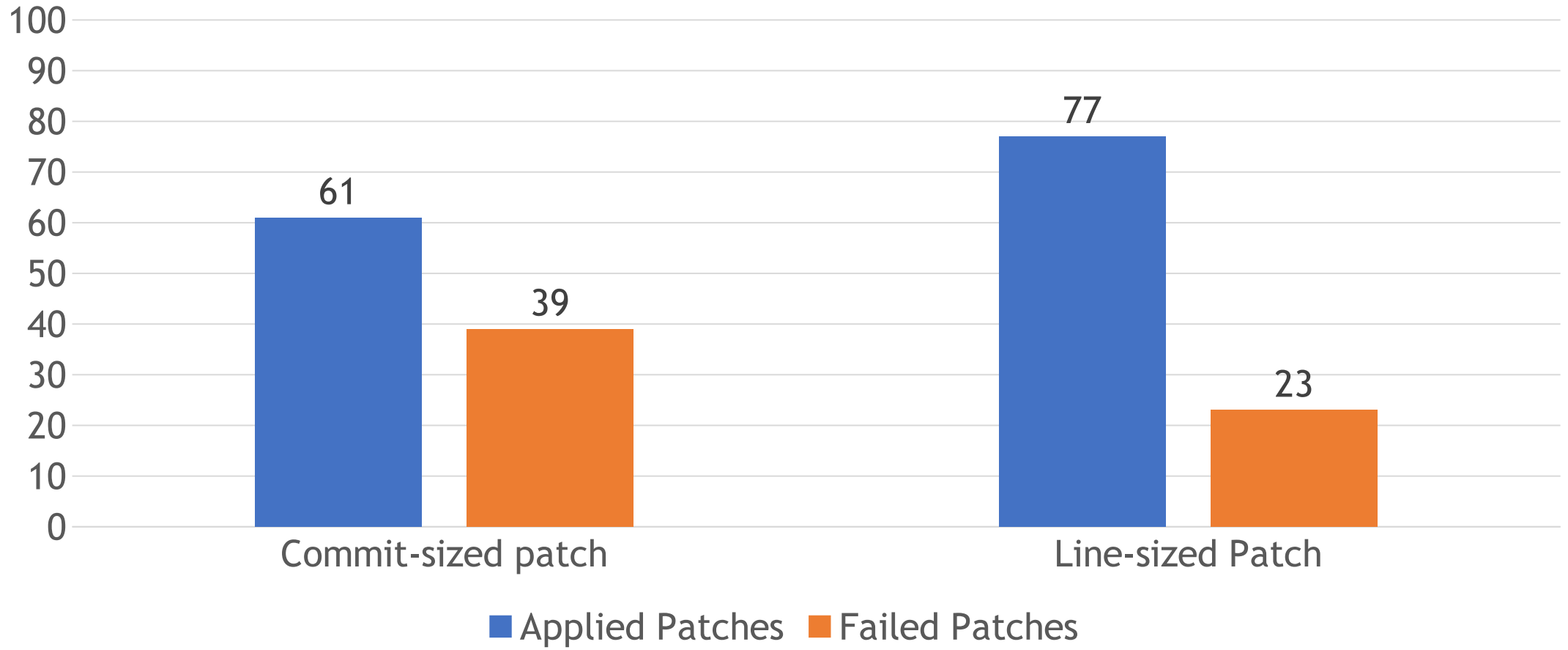


```
$ diff -Naur Edge.java_0 Edge.java_1
--- Edge.java_0 2021-10-08 10:07:54
+++ Edge.java_1 2021-10-08 10:07:55
@@ -12,7 +12,8 @@

   boolean equals(Edge e) {
     return source == e.source
-      && target == e.target;
+      && target == e.target
+      && weight == e.weight;
   }

   String toString() {
@@ -20,6 +21,6 @@
 ...
```

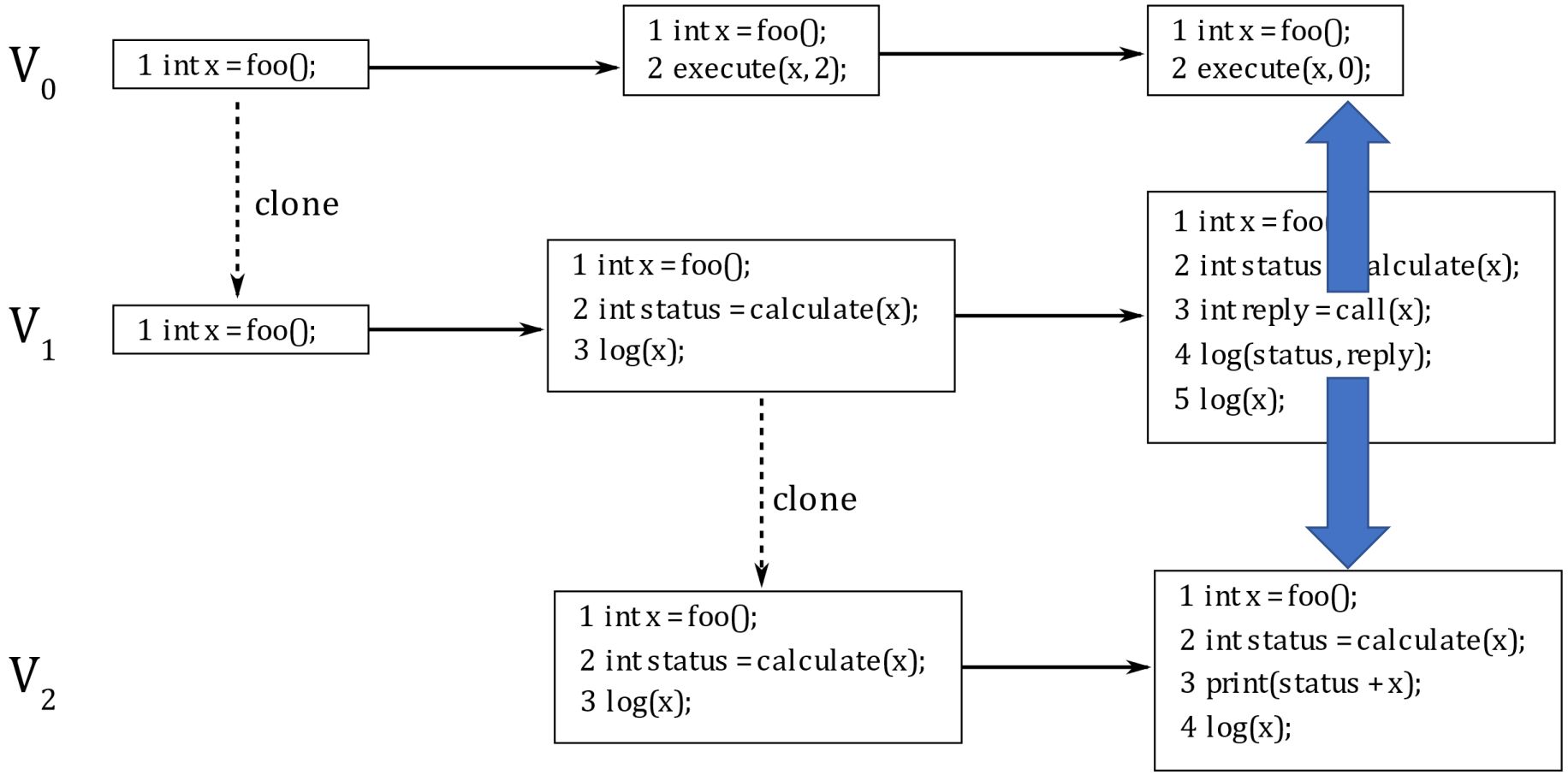


```
11 ...
12
13 boolean equals(Edge e) {
14   return source == e.source
15       && target == e.target;
16 }
17
18 String getLabel() {
19 ...
```

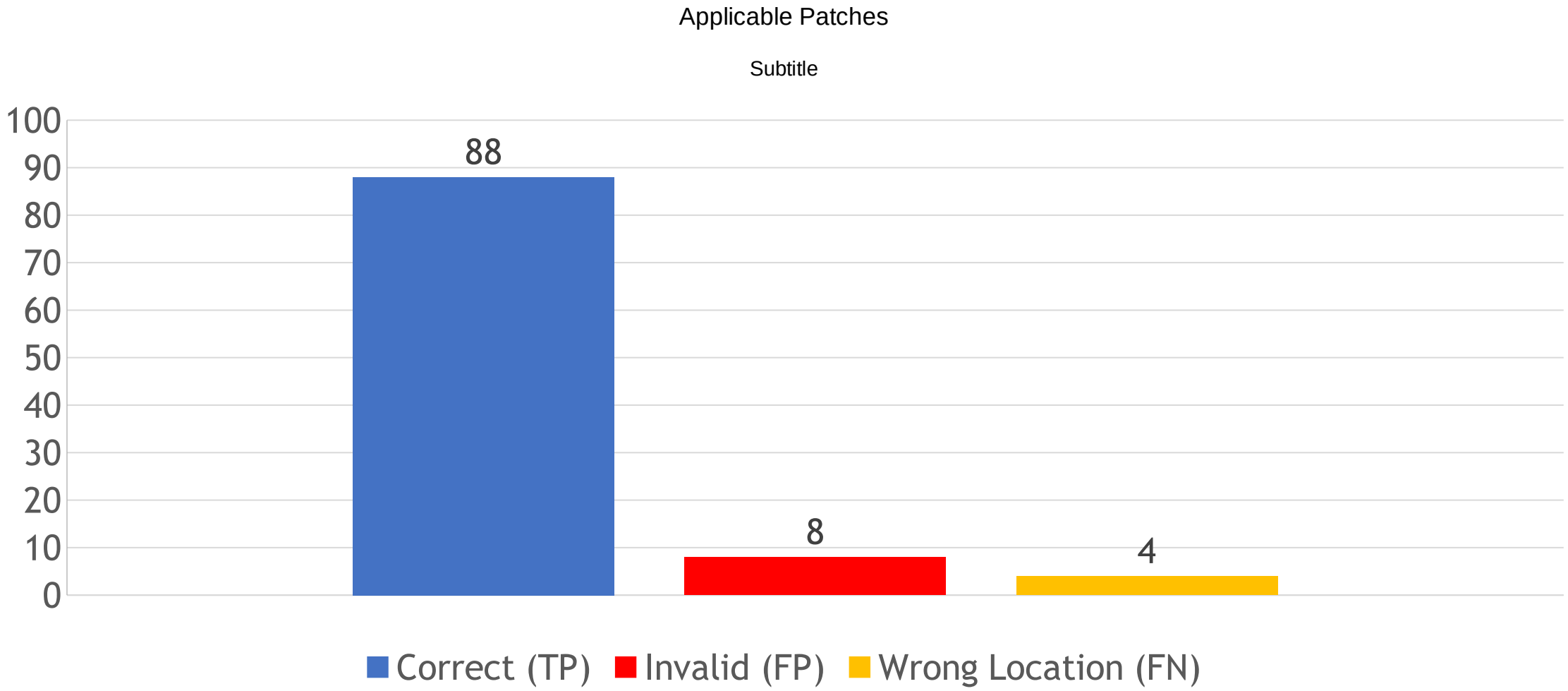
RQ1: Applicability of Blindly Propagated Changes (in %)



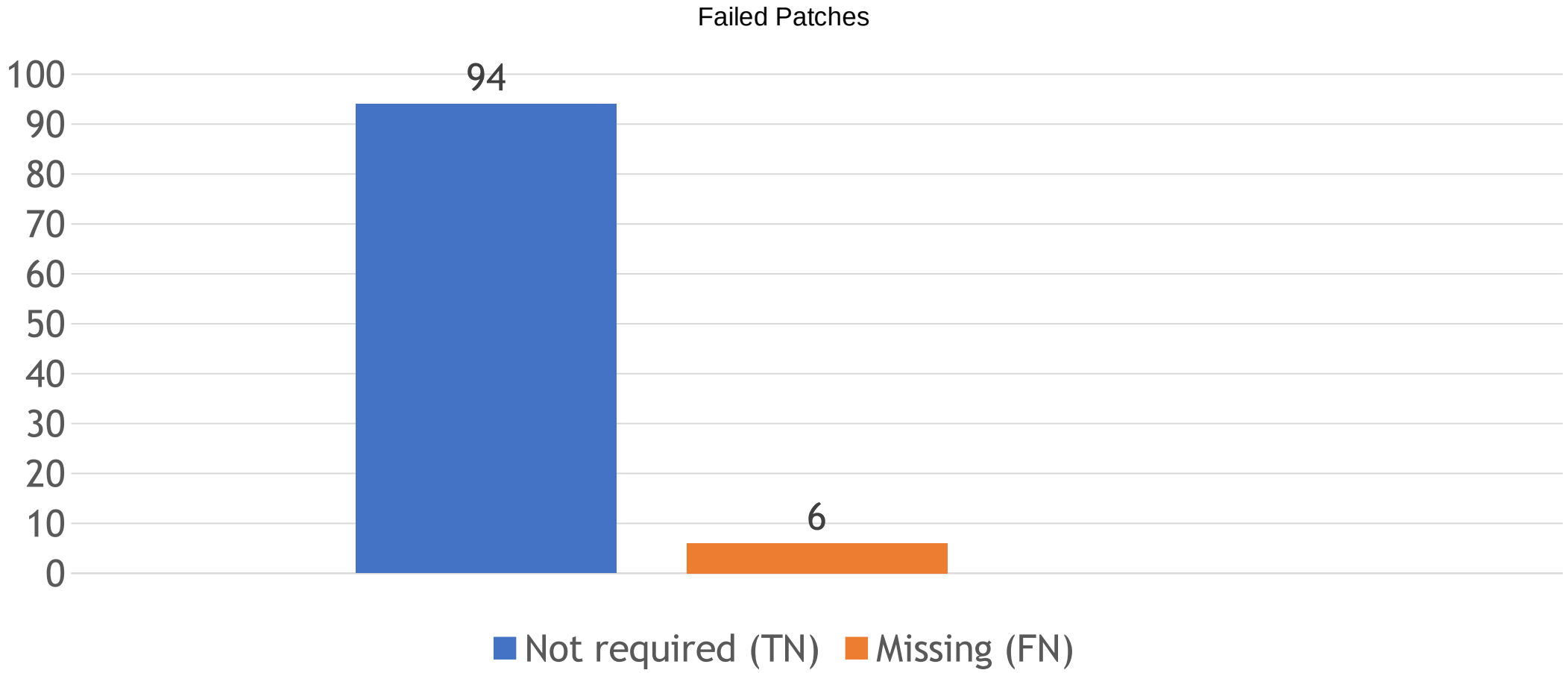
RQ2: How often is blind change propagation correct?



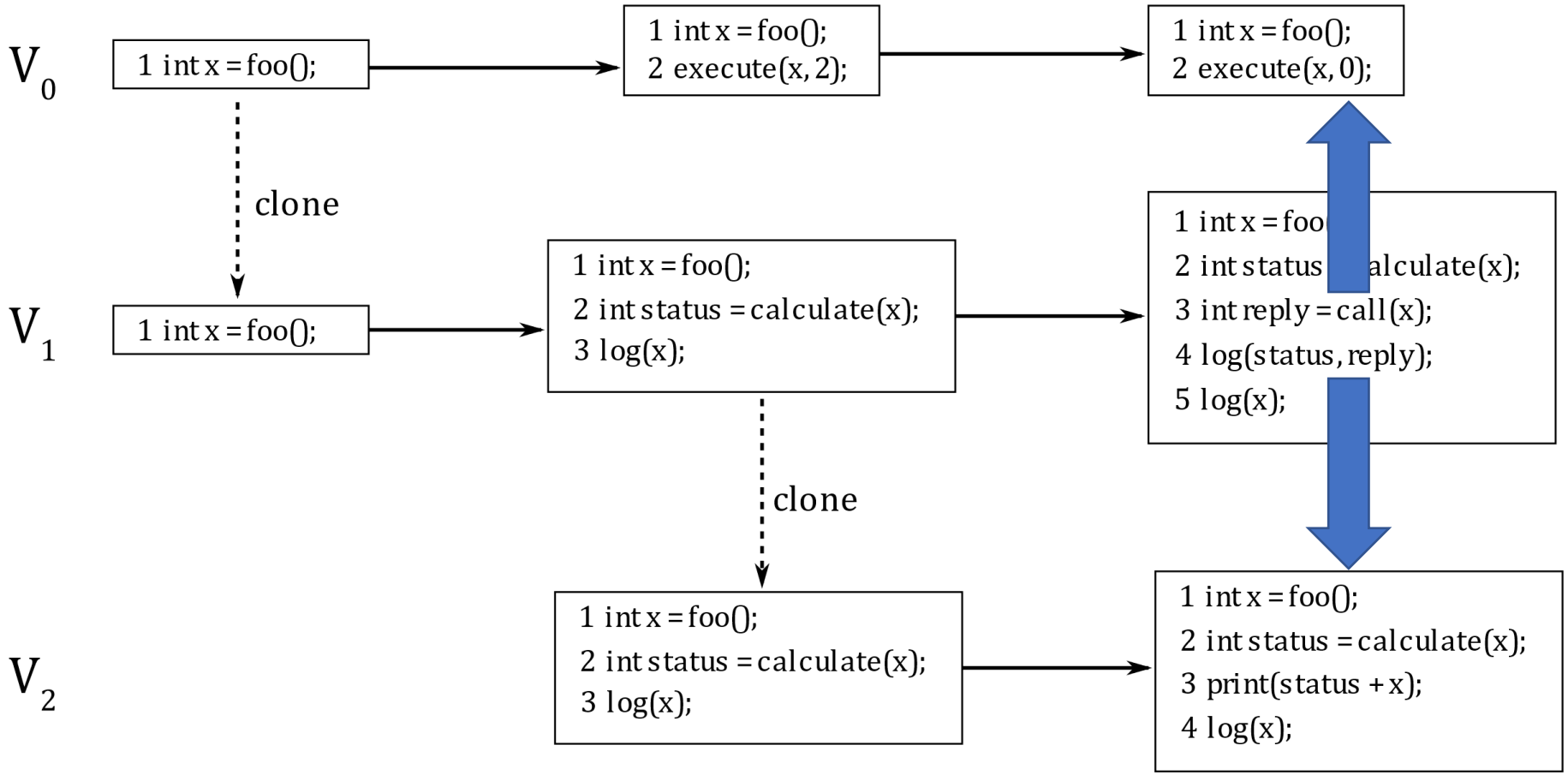
RQ2: Correctness of Blindly Propagated Changes (in %)



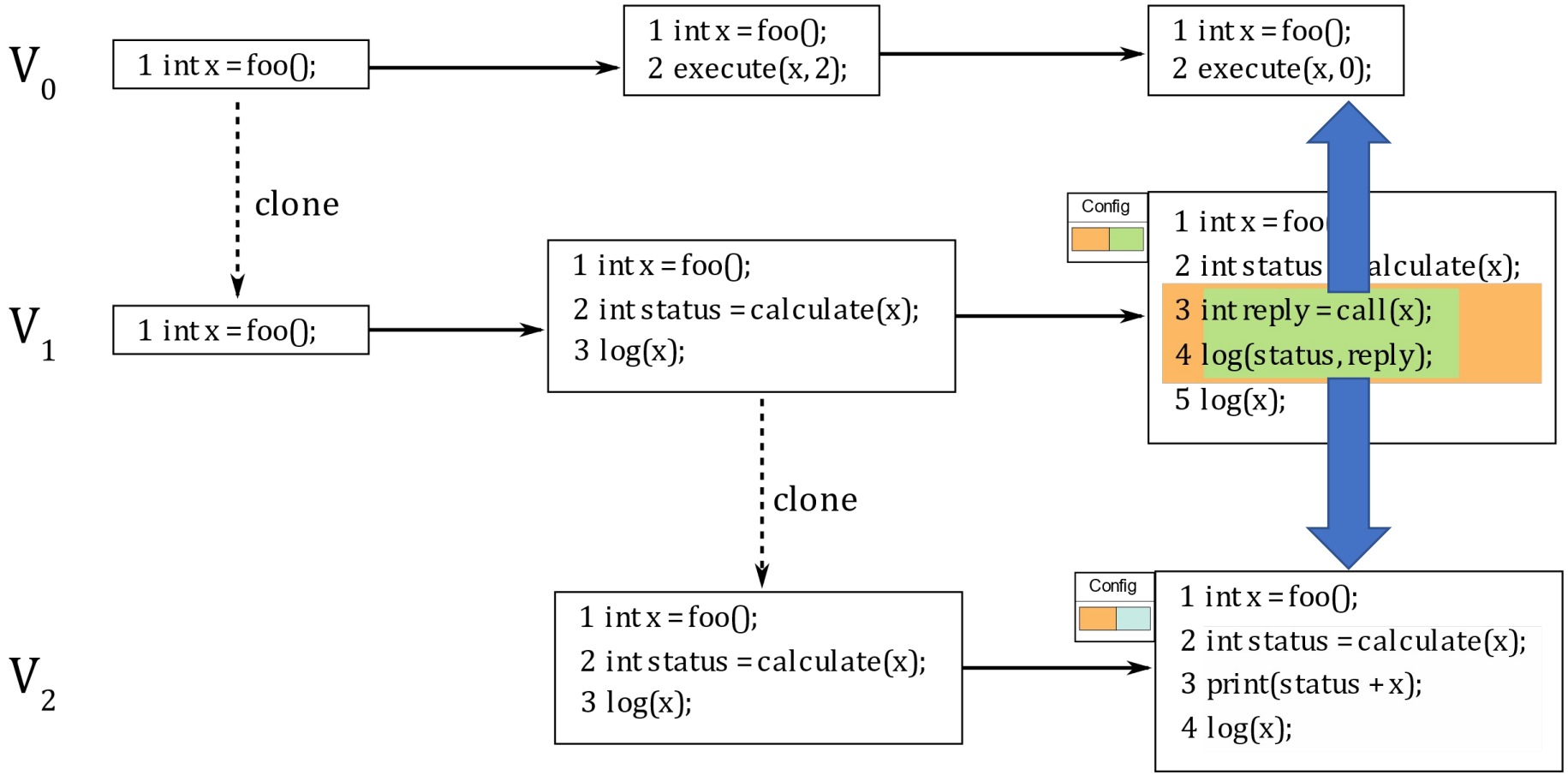
RQ2: Correctness of Blindly Propagated Changes (in %)



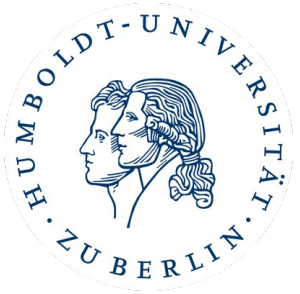
RQ3: What is the impact of considering knowledge about features?



RQ3: What is the impact of considering knowledge about features?

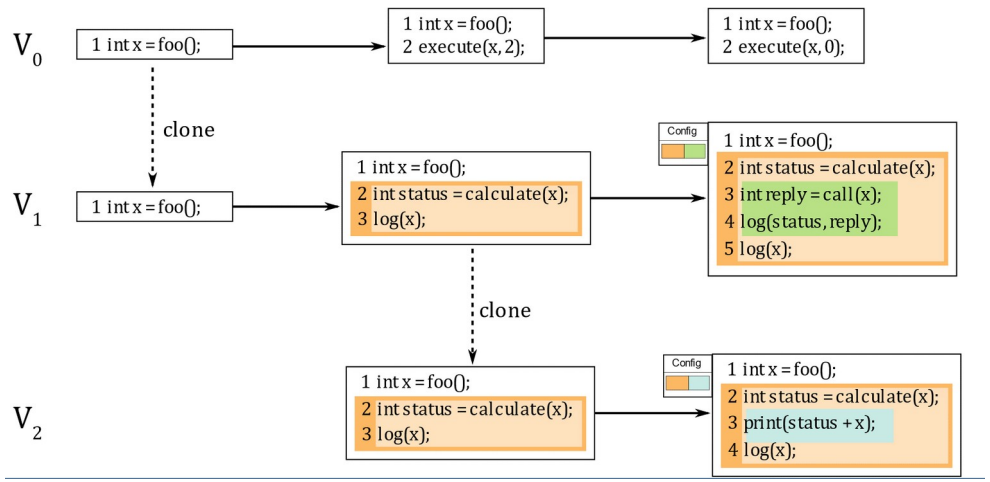


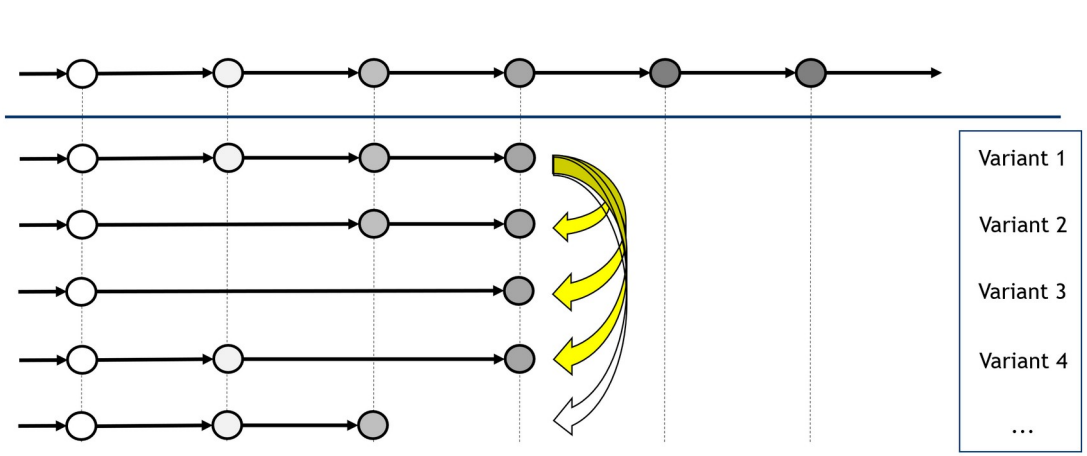
RQ3: Impact of considering knowledge about features

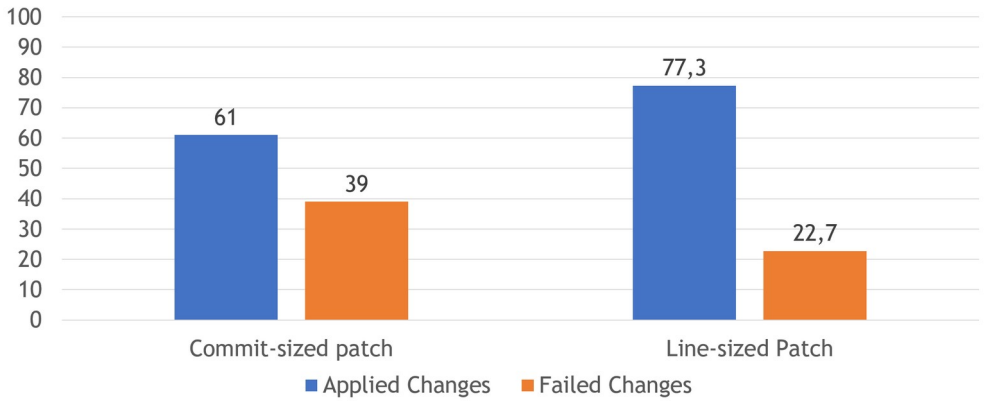


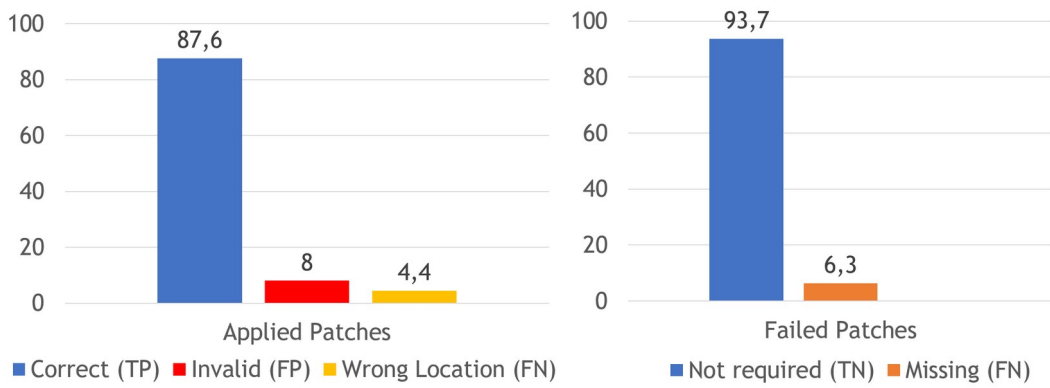
	Precision (are applied changes correct?)	Recall (have relevant changes been applied?)	Balanced Accuracy (degree of correct results)
Blind propagation	0.92	0.93	0.85
Feature-aware propagation	0.97	0.93	0.93

In summary...

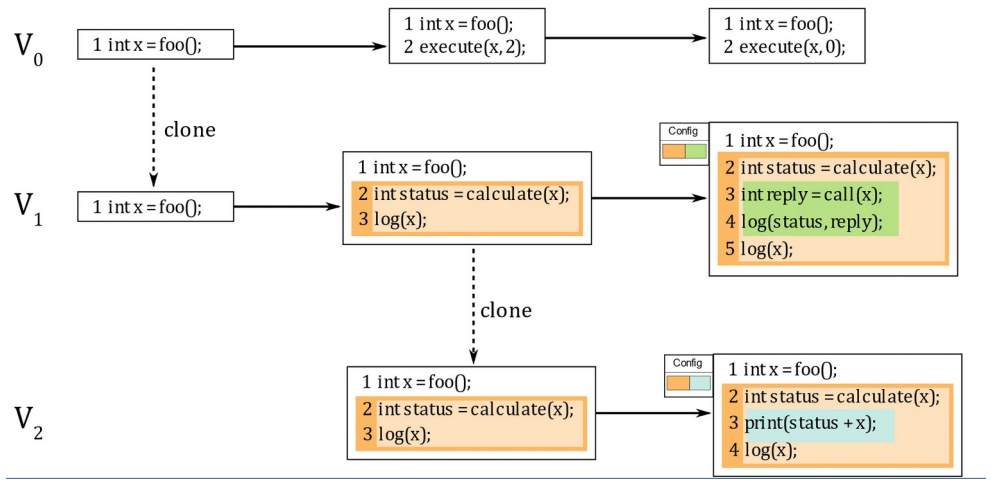

V0
1 int x = foo();
1 int x = foo();
2 execute(x, 2);
1 int x = foo();
2 execute(x, 0);
clone
V1
1 int x = foo();
1 int x = foo();
2 int status = calculate(x);
3 log(x);
Config
1 int x = foo();
2 int status = calculate(x);
3 int reply = call(x);
4 log(status, reply);
5 log(x);
clone
V2
1 int x = foo();
2 int status = calculate(x);
3 log(x);
Config
1 int x = foo();
2 int status = calculate(x);
3 print(status + x);
4 log(x);

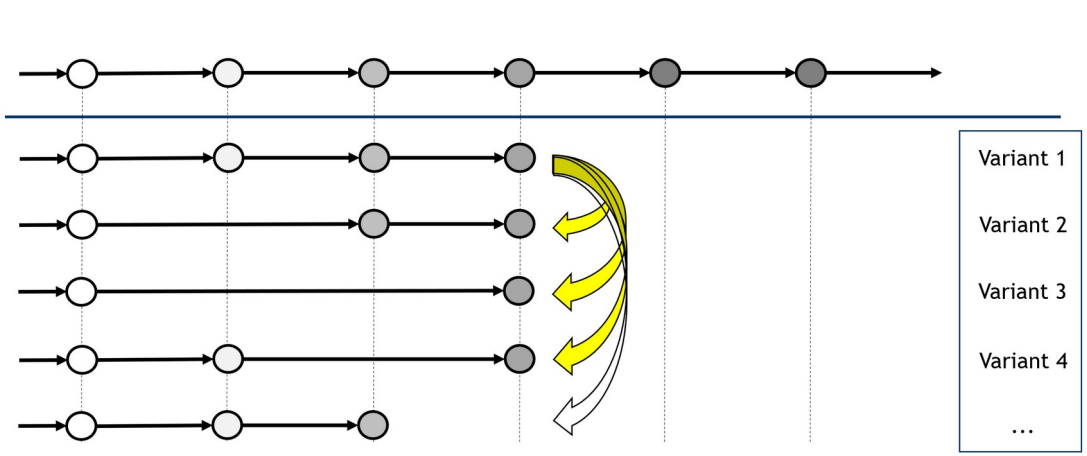

Variant 1
Variant 2
Variant 3
Variant 4
...

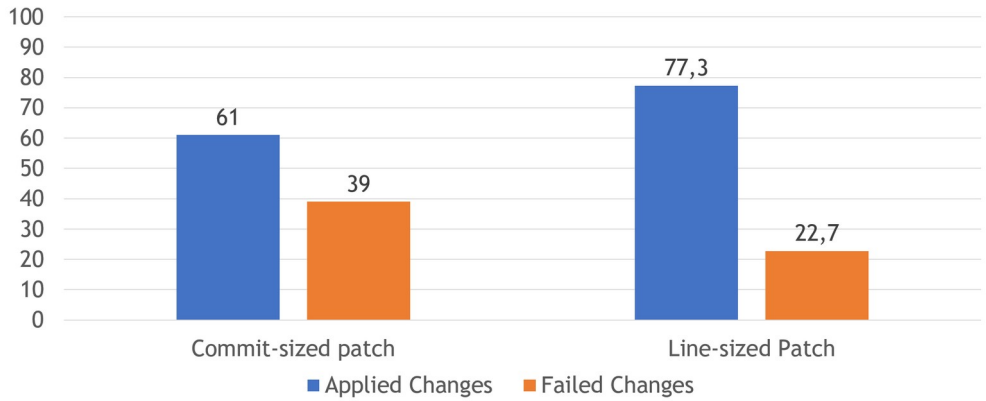

100
90
80
70
60
50
40
30
20
10
0
61
39
77,3
22,7
Commit-sized patch
Line-sized Patch
Applied Changes
Failed Changes

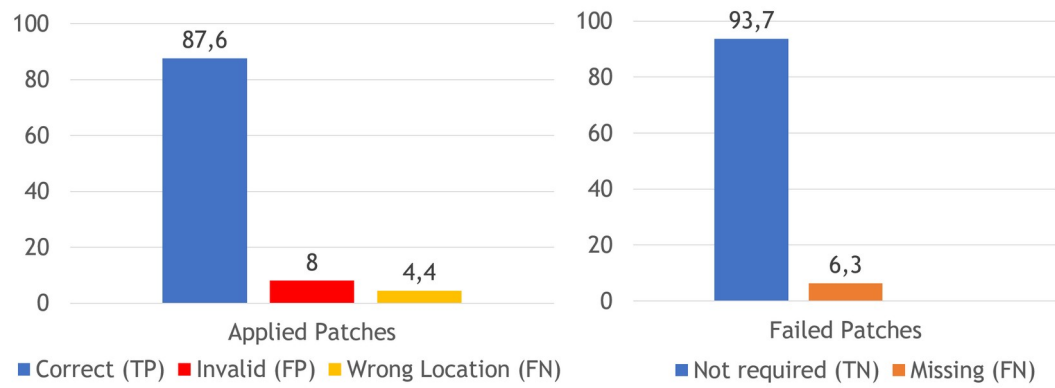

100
80
60
40
20
0
87,6
8
4,4
Applied Patches
Correct (TP)
Invalid (FP)
Wrong Location (FN)
100
80
60
40
20
0
93,7
6,3
Failed Patches
Not required (TN)
Missing (FN)

Questions?

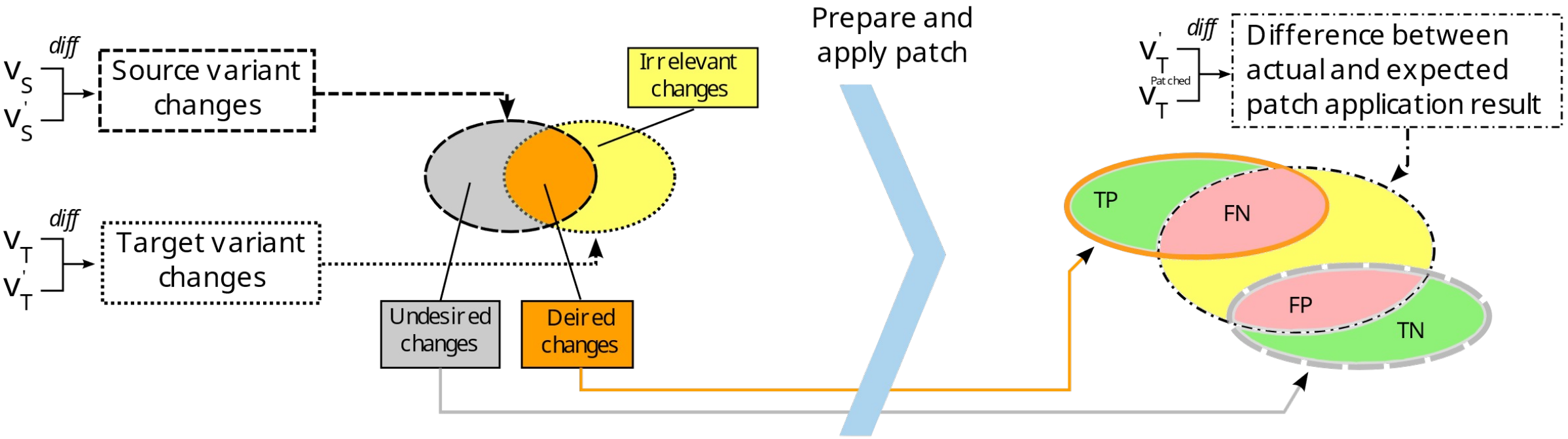

V0
1 int x = foo();
1 int x = foo();
2 execute(x, 2);
1 int x = foo();
2 execute(x, 0);
clone
V1
1 int x = foo();
1 int x = foo();
2 int status = calculate(x);
3 log(x);
Config
1 int x = foo();
2 int status = calculate(x);
3 int reply = call(x);
4 log(status, reply);
5 log(x);
clone
V2
1 int x = foo();
2 int status = calculate(x);
3 log(x);
Config
1 int x = foo();
2 int status = calculate(x);
3 print(status + x);
4 log(x);


Variant 1
Variant 2
Variant 3
Variant 4
...

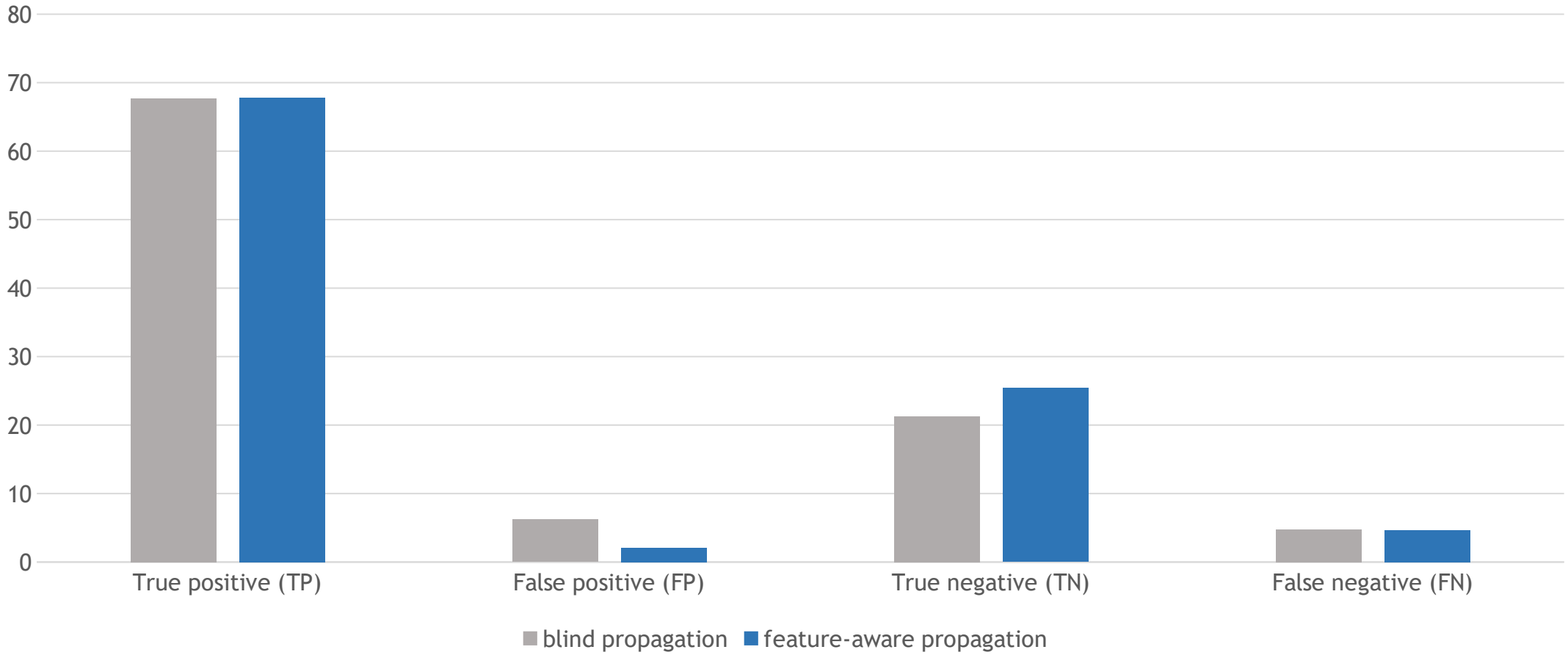

100
90
80
70
60
50
40
30
20
10
0
61
39
77,3
22,7
Commit-sized patch
Line-sized Patch
Applied Changes
Failed Changes


100
80
60
40
20
0
87,6
8
4,4
Applied Patches
Correct (TP)
Invalid (FP)
Wrong Location (FN)
100
80
60
40
20
0
93,7
6,3
Failed Patches
Not required (TN)
Missing (FN)

Evaluation of patch outcomes



RQ3: Impact of considering knowledge about features



The VariantSync project

Central Goal:

- Automatic synchronization of variants

Basic Idea:

- Trace features to their implementation
- Propagate feature changes to other variants automatically

Simulation of Automated Change Propagation

Not all variants are changed in a commit