# Inspecting Code Churns to Prioritize Test Cases

Luigi Libero Lucio Starace

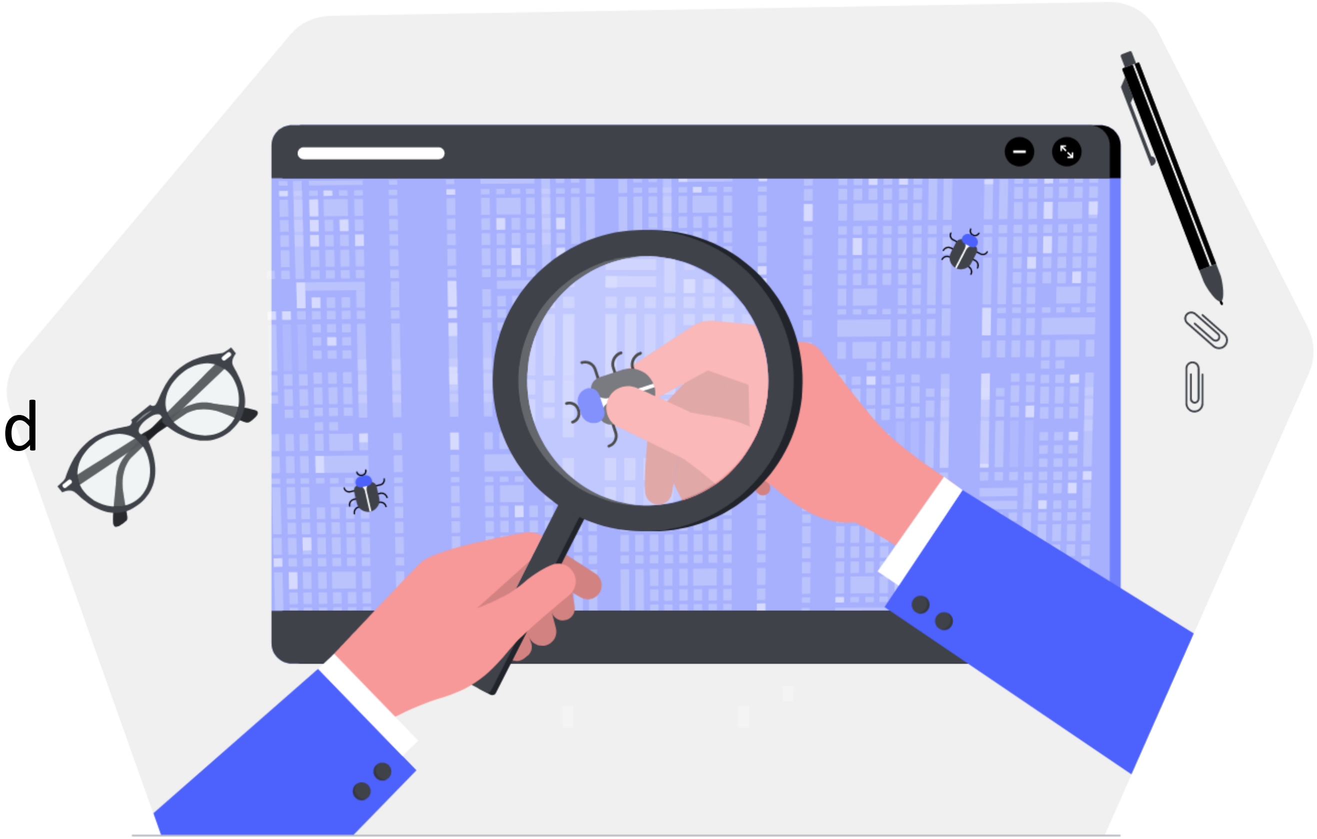
University of Naples *Federico II,* dept. of Electrical Engineering and Information Technology

luigiliberolucio.starace@unina.it

December 9, 2020

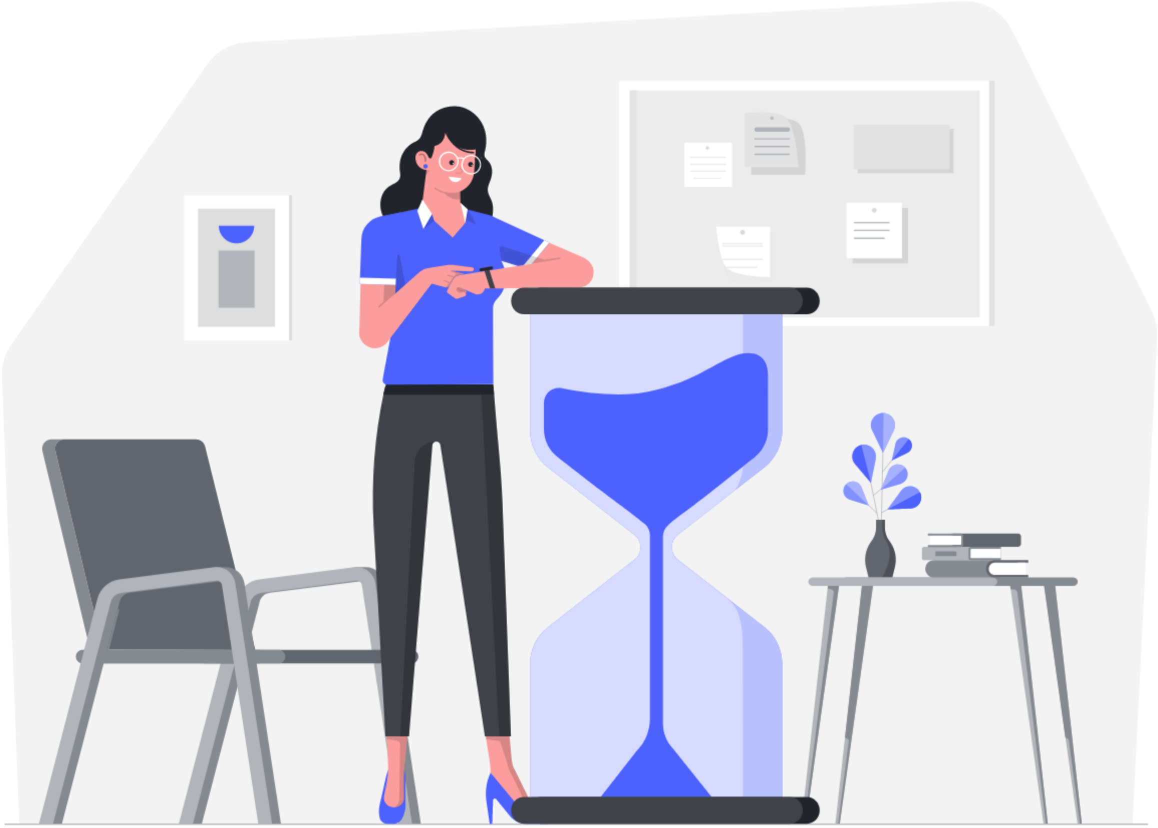
# Regression Testing

- Within Software Maintenance, changes can impact previously validated functionalities
- **Regression Testing** aims at making sure that the unchanged parts have not been adversely affected by the changes



# Regression Testing: Challenges

- Test suites can take up to **days** to execute
- Sometimes there's not enough time or resources!
- And even if there were, why waste them?

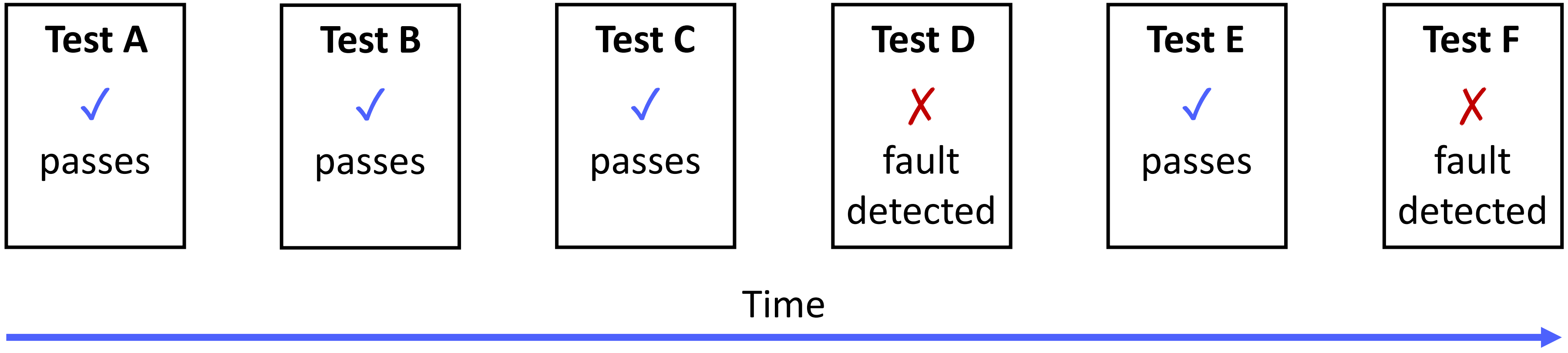


# Test Prioritization

The **order** in which test cases in a test suite are executed matters!

A «good» ordering can improve **fault detection** and **coverage rates**.

- Faults are detected earlier;
- Test suite gives satisfactory confidence in the system's reliability earlier.



**Test A** ✓ passes

**Test B** ✓ passes

**Test C** ✓ passes

**Test D** ✗ fault detected

**Test E** ✓ passes

**Test F** ✗ fault detected

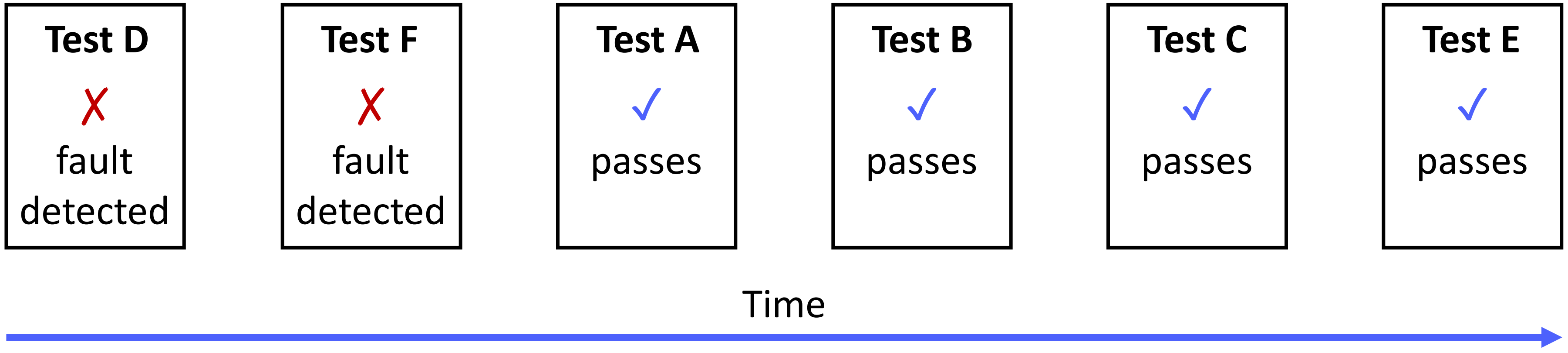
Time

# Test Prioritization

The **order** in which test cases in a test suite are executed matters!

A «good» ordering can improve **fault detection** and **coverage rates**.

- Faults are detected earlier;
- Test suite gives satisfactory confidence in the system’s reliability earlier.



**Test D** ✗ fault detected

**Test F** ✗ fault detected

**Test A** ✓ passes

**Test B** ✓ passes

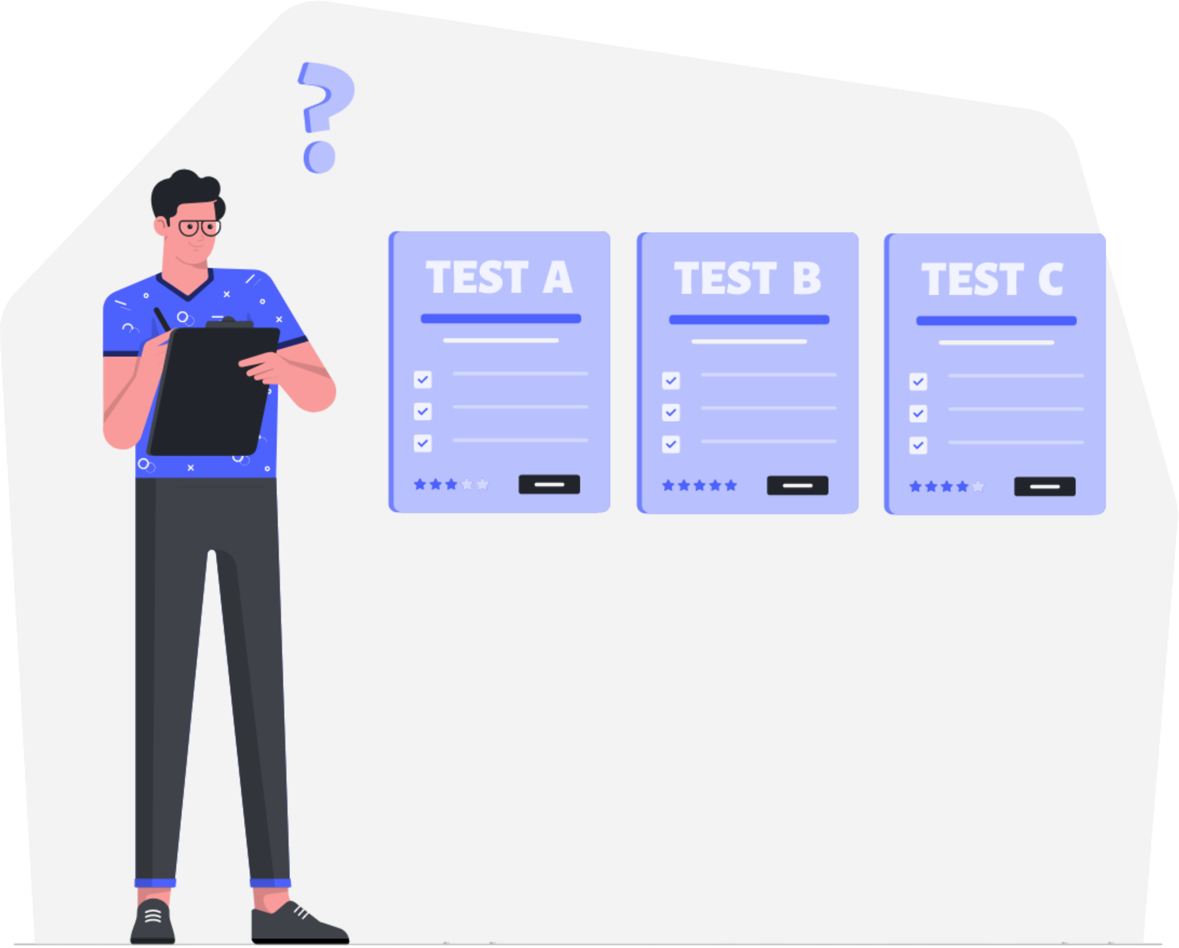
**Test C** ✓ passes

**Test E** ✓ passes

Time

# Test Prioritization

- We don’t know which tests will reveal faults before executing them
- How can we find a «good» order?
- Several **Heuristics** have been proposed
  - Code Coverage
  - History Information
  - **Code Churns**



# Code Churns

**Code churn**[1] measures the source code changes between two versions of a Software.

Version **V**

```
void hello() {
    String s = "ICTSS";
    print("Hello "+s);
}

```

Version **V’**

```
void hello() {
    print("Hello ICTSS!");
    print("How’s it going?");
}

```

[1] Nagappan, N., & Ball, T. (2005, May). Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering* (pp. 284-292).

# Code Churns

**Code churn**[1] measures the source code changes between two versions of a Software.

Version **V**

```
void hello() {
    String s = "ICTSS";
    print("Hello "+s);
}
```

Version **V’**

```
  void hello() {
-     String s = "ICTSS";
      print("Hello ICTSS!");
+     print("How’s it going?");
  }
```

[1] Nagappan, N., & Ball, T. (2005, May). Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering* (pp. 284-292).

# Our proposal

# Introducing Churn Coverage

Given a test case $t$, we defined its **churn coverage** w.r.t. version $V$ as the tuple:

$$\langle c, d, u \rangle$$

- $c$ is the number of **changed** code units (w.r.t. $V$’) which are covered by $t$
- $d$ is the number of **deleted** code units (w.r.t. $V$’) which are covered by $t$
- $u$ is the number of **unchanged** code units (w.r.t. $V$’) which are covered by $t$ (i.e. covered code units that are neither changed nor deleted)

# Churn Coverage: Example

Version **V**

```
 1 int foo(int a) {
 2       int x = 0;
 3       x = 5;
 4       if(a > 10) {
 5             a = a + 1;
 6             x = a + 2;
 7       }
 8       else {
 9             x = a * 2;
10       }
11       return x;
12 }
```

Version **V’**

```
 1 int foo(int a) {
 2       int x = 5;
-        x = 5;
 3       if(a > 10) {
 4             a = a + 1;
 5             x = a + 2;
 6       }
 7       else {
 8             x = a * 3;
 9       }
10       return x - 4;
11 }
```

# Churn Coverage: Example

Version **V**

```
int foo(int a) {
    int x = 0;
    x = 5;
    if(a > 10) {
        a = a + 1;
        x = a + 2;
    }
    else {
        x = a * 2;
    }
    return x;
}
```

Test case **t**

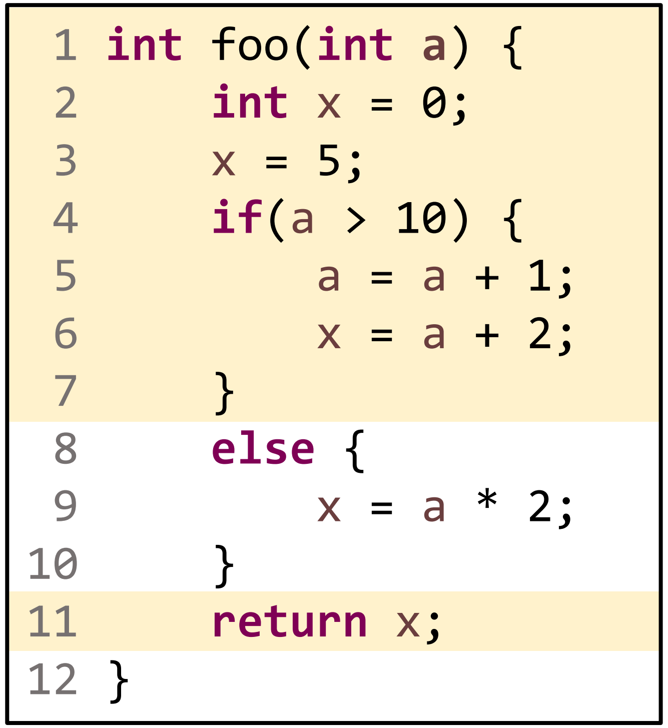
$$\langle c, d, u \rangle$$

Version **V’**

```
 int foo(int a) {
     int x = 5;
-    x = 5;
     if(a > 10) {
         a = a + 1;
         x = a + 2;
     }
     else {
         x = a * 3;
     }
     return x - 4;
 }
```

# Churn Coverage: Example

### Version V



```
int foo(int a) {
    int x = 0;
    x = 5;
    if(a > 10) {
        a = a + 1;
        x = a + 2;
    }
    else {
        x = a * 2;
    }
    return x;
}
```

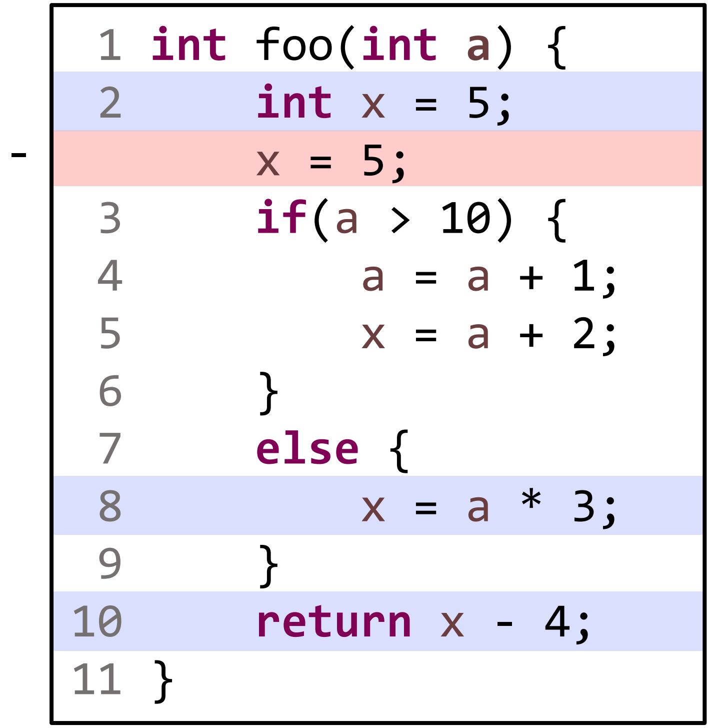
### Test case t

$$\langle c, d, u \rangle$$

$$||$$

$$\langle 2,1,3 \rangle$$

### Version V’



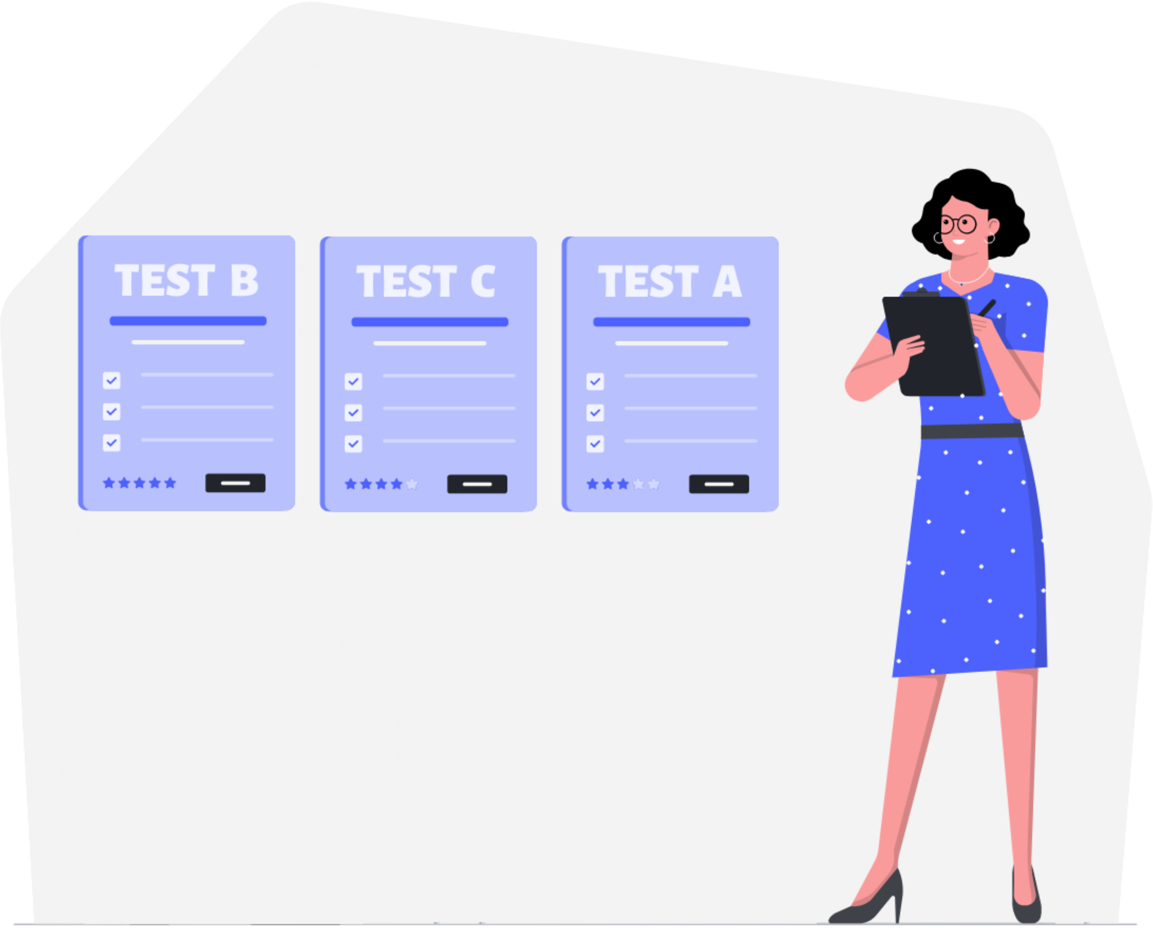
```
  int foo(int a) {
      int x = 5;
-     x = 5;
      if(a > 10) {
          a = a + 1;
          x = a + 2;
      }
      else {
          x = a * 3;
      }
      return x - 4;
  }
```

# Test Prioritization based on Churn Coverage

A **prioritization strategy** is an **order relation** on the test cases.

We can prioritize by sorting the test cases according to these relations.

**Main contribution:** We designed and experimentally evaluated three novel prioritization strategies.



# Baseline Strategy: Total Coverage[1]

Let $t$ and $t'$be two tests, and let $\langle c, d, u \rangle$ and $\langle c', d', u' \rangle$ be the respective churn coverage.

$$t \preccurlyeq_{Tot} t' \Leftrightarrow \underbrace{c + d + u}_{\substack{\text{Total number of}\\ \text{code units}\\ \text{covered by } t}} \leq \underbrace{c' + d' + u'}_{\substack{\text{Total number of}\\ \text{code units}\\ \text{covered by } t'}}$$

[1] Hao, D., Zhang, L., Zhang, L., Rothermel, G., & Mei, H. (2014). A unified test case prioritization approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *24*(2), 1-31.

# Prioritize Churn Strategy

Let $t$ and $t'$be two tests, and let $\langle c, d, u\rangle$ and $\langle c', d', u'\rangle$ be the respective churn coverage.

$$t \preccurlyeq_{Churn} t' \Leftrightarrow \frac{c+d}{c+d+u} \leq \frac{c'+d'}{c'+d'+u'}$$

# Prioritize Unchanged Strategy

Let $t$ and $t'$be two tests, and let $\langle c, d, u\rangle$ and $\langle c', d', u'\rangle$ be the respective churn coverage.

$$t \preccurlyeq_{Unch} t' \Leftrightarrow \frac{u}{c+d+u} \leq \frac{u'}{c'+d'+u'}$$

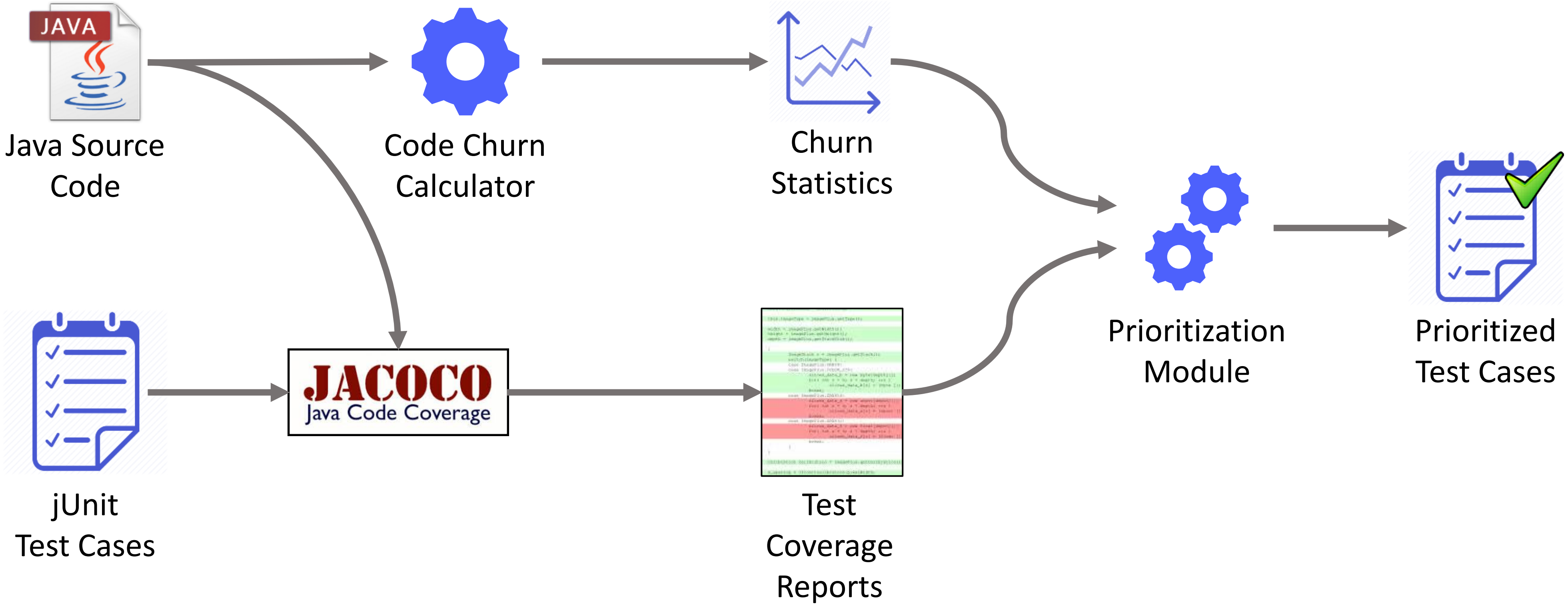
# Combined Strategy

Let $t$ and $t'$be two tests, and let $\langle c, d, u\rangle$ and $\langle c', d', u'\rangle$ be the respective churn coverage.

$$t \preccurlyeq_{Comb} t' \Leftrightarrow \begin{array}{c} (c+d) < (c'+d') \\ \vee \\ (c+d) = (c'+d') \wedge u \leq u' \end{array}$$

# Empirical Evaluation

To assess the **effectiveness** of the proposed strategies, we implemented them in a prioritization toolchain

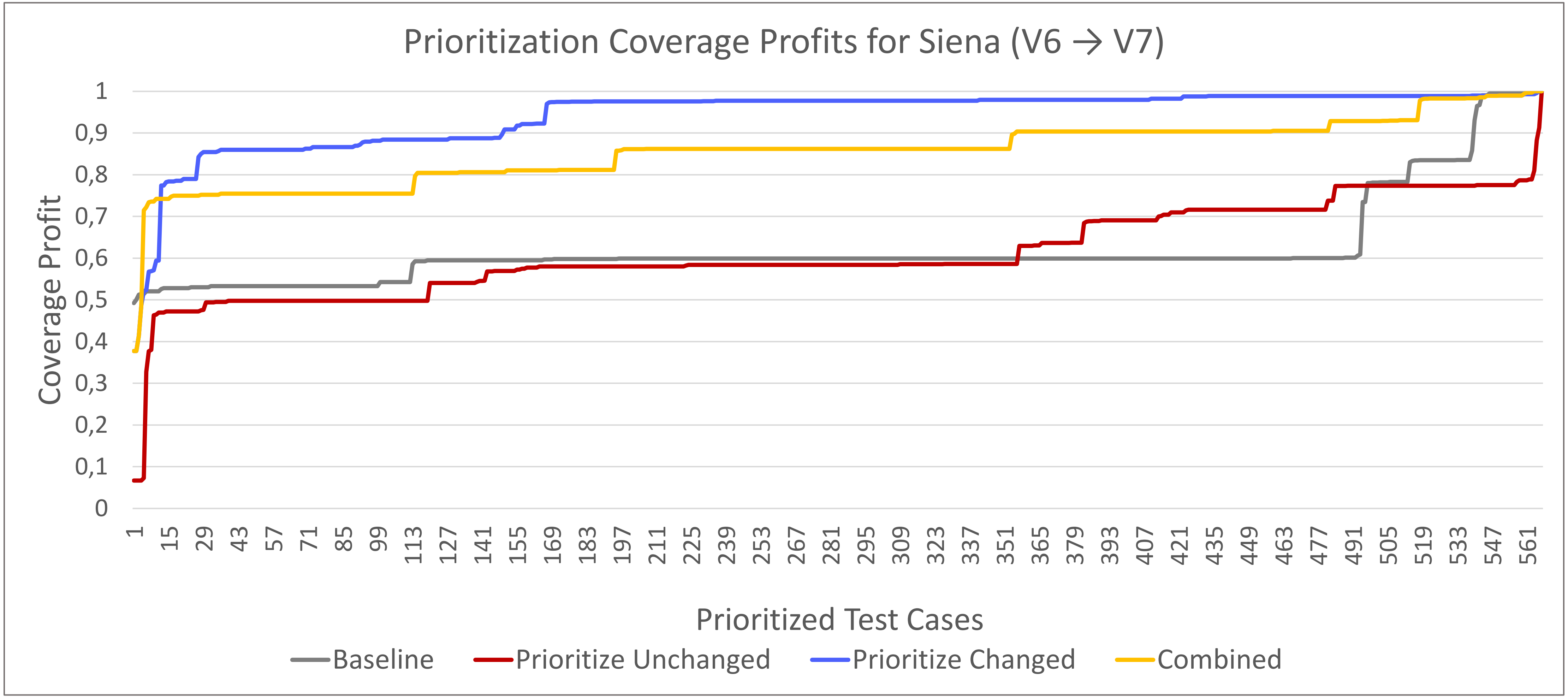


# Empirical Evaluation: Subject

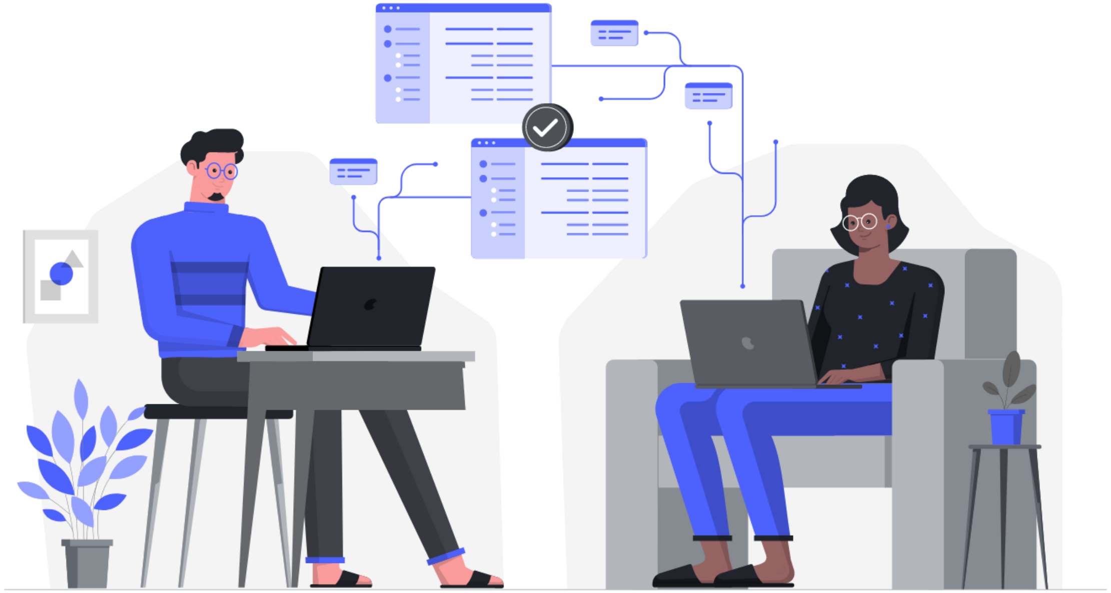
As a subject for our experiments, we selected ***Siena*** (**S**calable **I**nternet **E**vent **N**otification **A**rchitecture)

- 7 versions
- ~11k lines of code
- ~500 test cases

# Results


Prioritization Coverage Profits for Siena (V6 → V7)
Coverage Profit
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393
407
421
435
449
463
477
491
505
519
533
547
561
Prioritized Test Cases
Baseline
Prioritize Unchanged
Prioritize Changed
Combined

# Are all code changes equal?

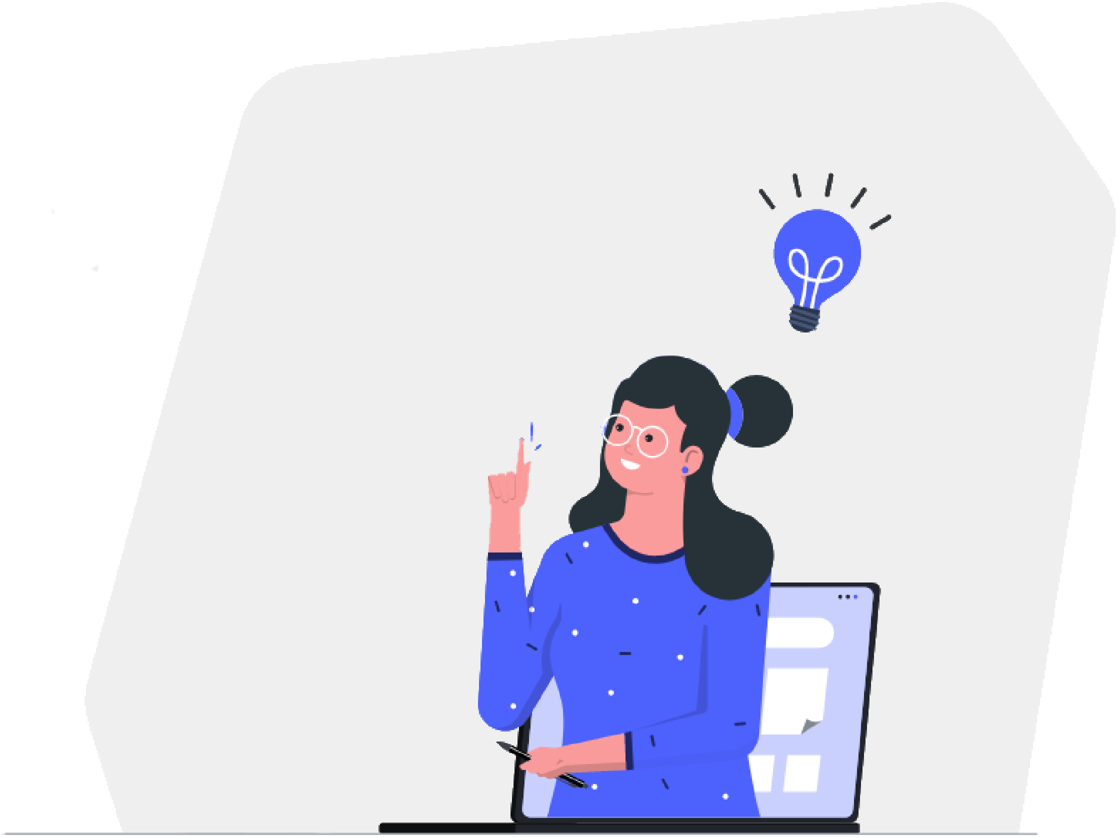


# Some changes are more critical

- Renaming a local variable in a method is less critical
- Changing the condition in a branching statement on in a loop, on the other hand…

↓

We should prioritize tests covering code with more **critical changes**



# A New Approach to Code Churn Evaluation

## Standard Approach

Did this code unit change?

- Yes/No

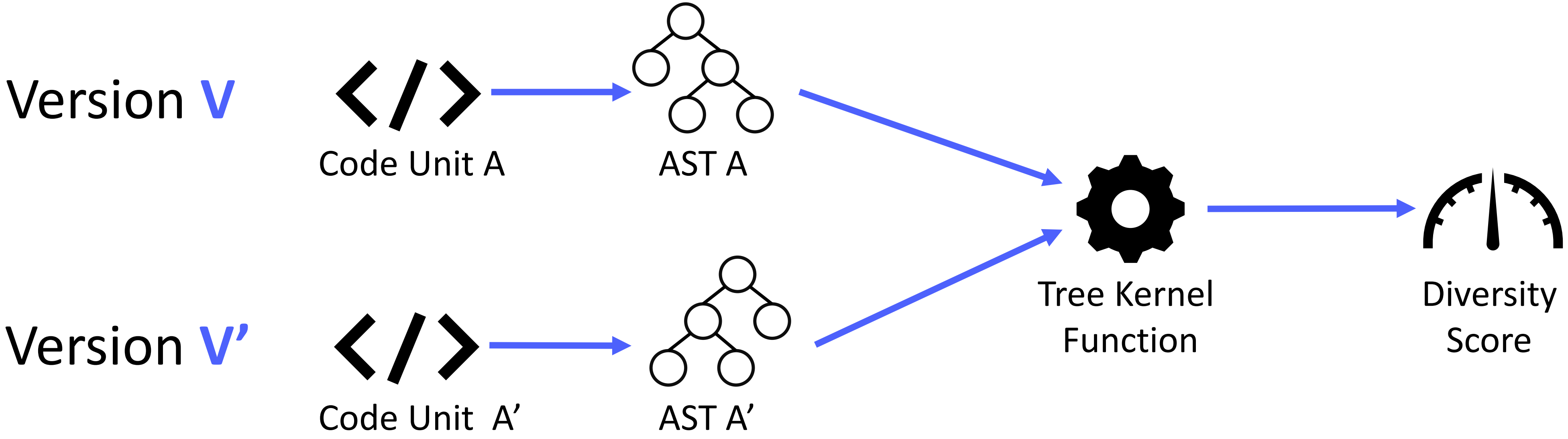
## New Approach

**How much** did this code unit change?

- Score in [0,1]
  - 0 if the unit is unchanged
  - 1 if the unit changed significantly
  - Every value in between!

# How do we do it?

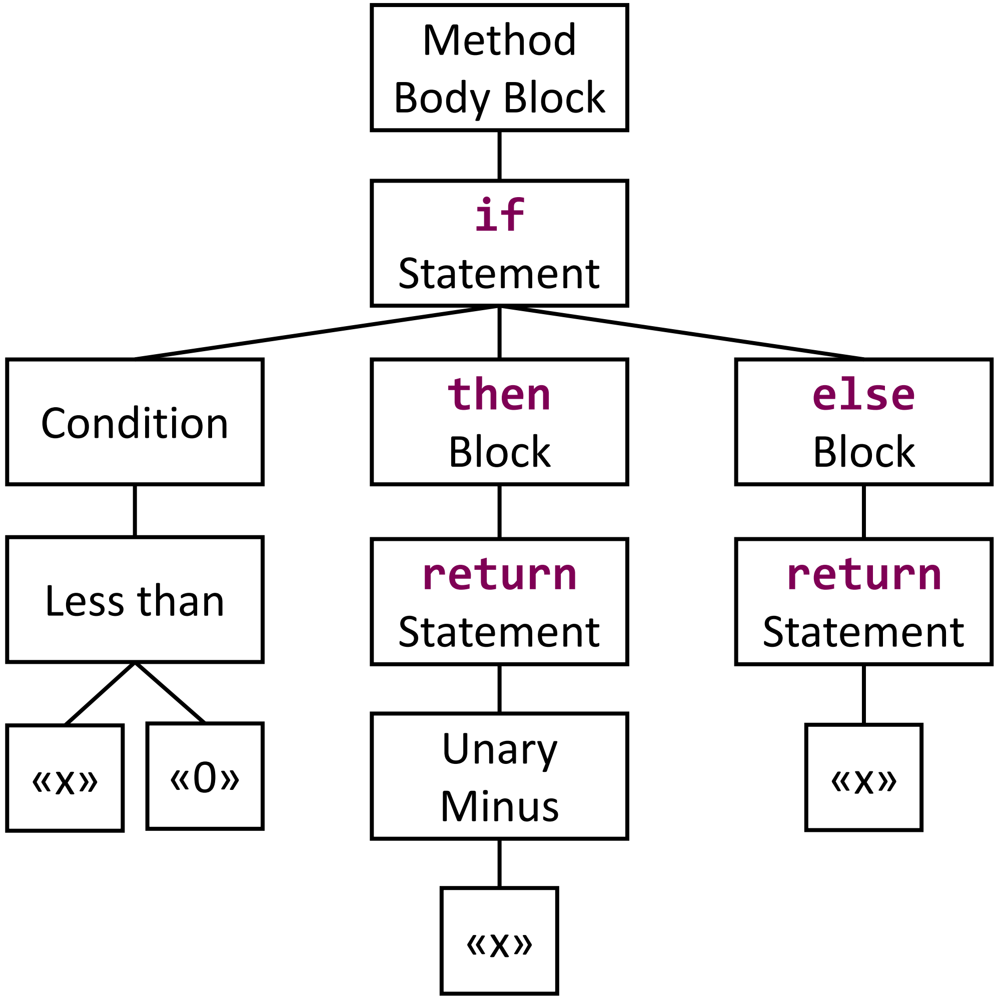
- We use a **Abstract Syntax Tree** (AST) representation for the two versions of a code unit
- We use suitably-designed **Tree Kernel Functions** to compute a diversity score.



# Abstract Syntax Tree Representation

```
public float abs(float x) {
    if(x < 0)
        return -x;
    else
        return x;
}
```

- **Structured** information
- Ignores indentation, whitespaces, etc…

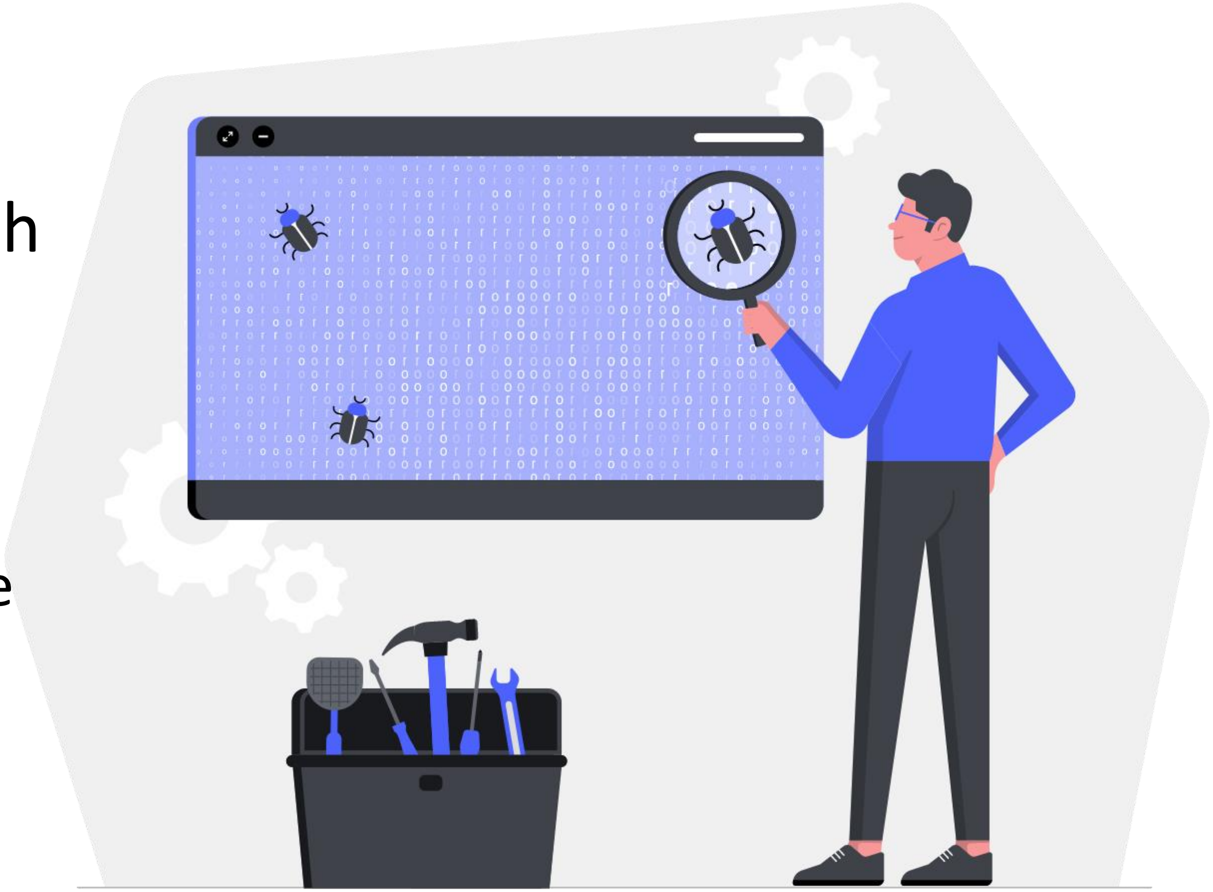


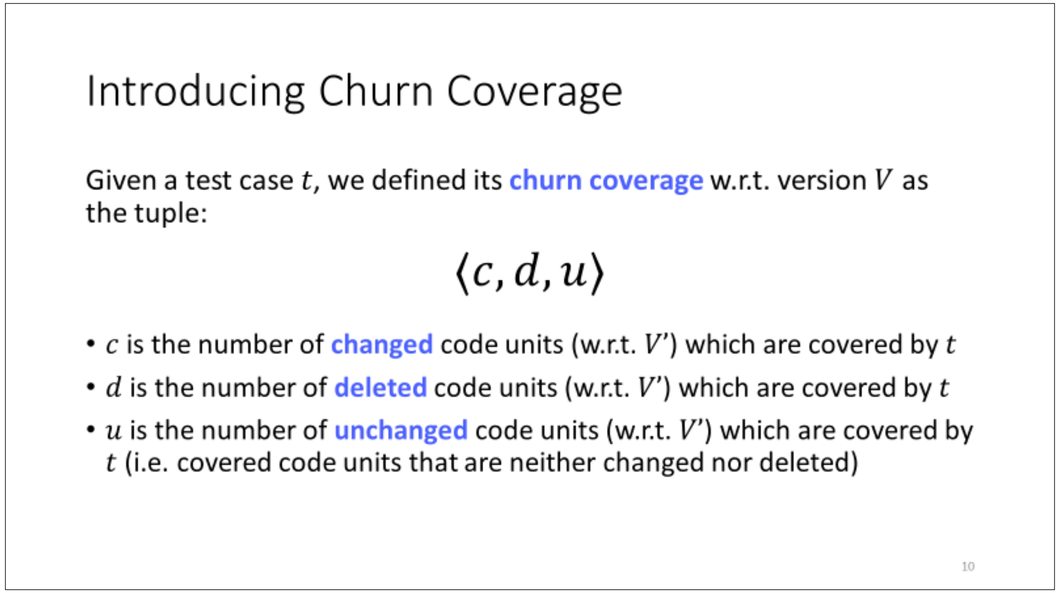
# Tree Kernel Functions

- New class of functions successfully applied in **Natural Language Processing**.
- Compute **similarity** between tree structures.
- **Highly Customizable**
  - Easy to customize which tree parts have a greater impact on similarity
- Can be computed efficiently using **Dynamic Programming** and **memoization**.
- Recently used in Software Engineering for clone detection, but never in test case prioritization

# Future Works

- We are working on extending our prioritization toolchain with this refined approach
- We plan to conduct a more extensive evaluation
  - on several open source software projects
  - using **fault detection**-related metrics



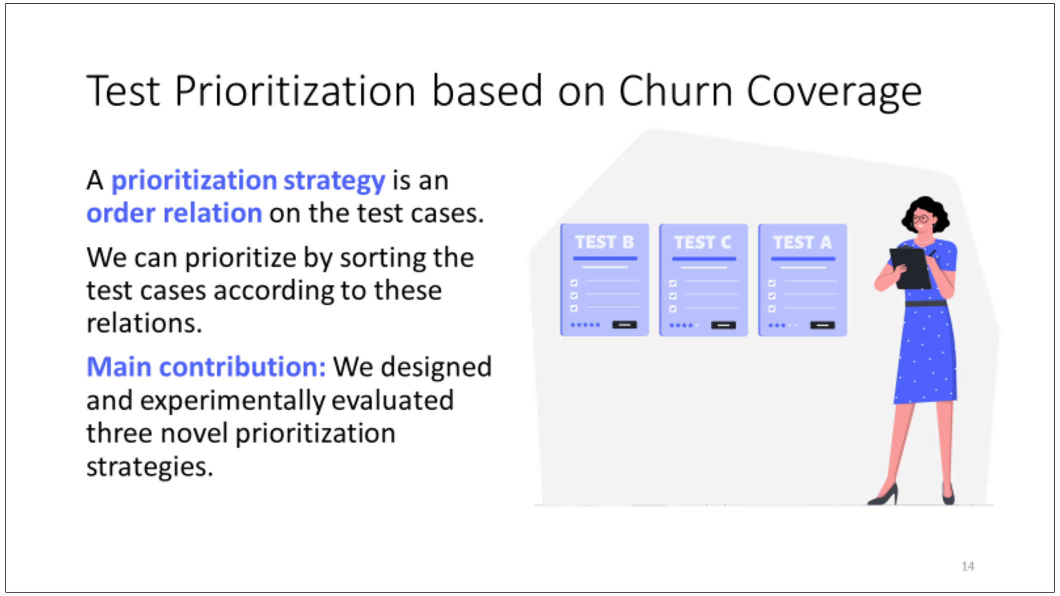


## Introducing Churn Coverage

Given a test case $t$, we defined its **churn coverage** w.r.t. version $V$ as the tuple:

$$\langle c, d, u \rangle$$

- $c$ is the number of **changed** code units (w.r.t. $V'$) which are covered by $t$
- $d$ is the number of **deleted** code units (w.r.t. $V'$) which are covered by $t$
- $u$ is the number of **unchanged** code units (w.r.t. $V'$) which are covered by $t$ (i.e. covered code units that are neither changed nor deleted)



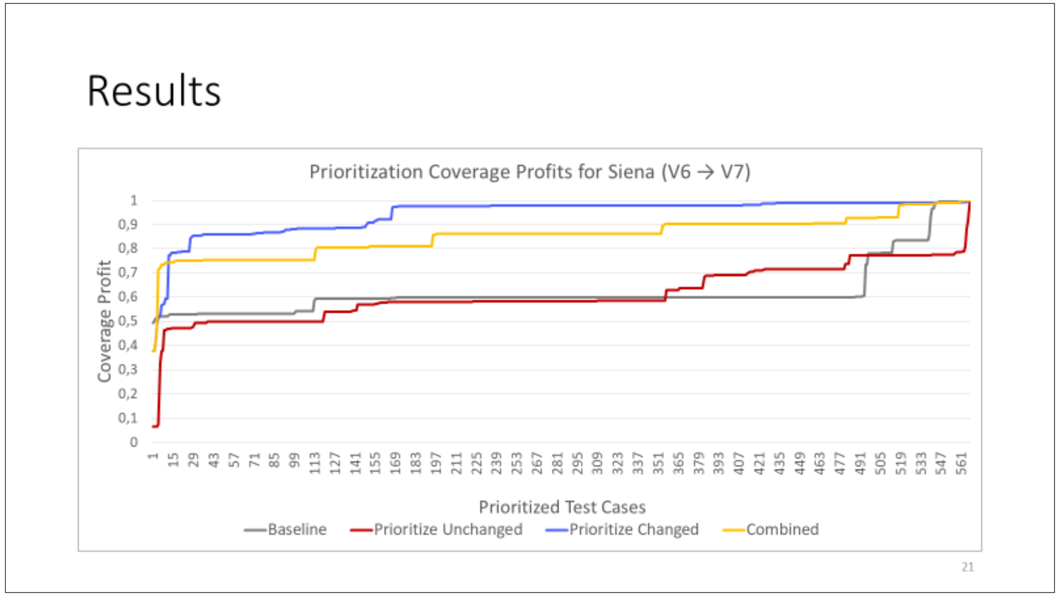
## Test Prioritization based on Churn Coverage

A **prioritization strategy** is an **order relation** on the test cases.

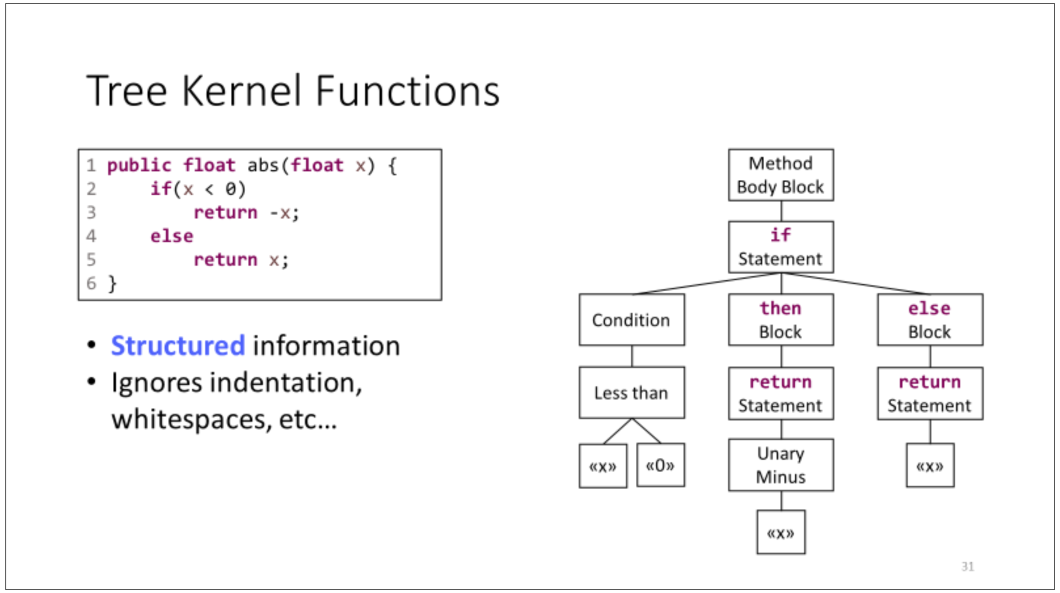
We can prioritize by sorting the test cases according to these relations.

**Main contribution:** We designed and experimentally evaluated three novel prioritization strategies.

TEST B
TEST C
TEST A



## Results



## Tree Kernel Functions

```
public float abs(float x) {
    if(x < 0)
        return -x;
    else
        return x;
}
```

- **Structured** information
- Ignores indentation, whitespaces, etc...

Inspecting Code Churns to Prioritize Test Cases

Luigi Libero Lucio Starace
luigiliberolucio.starace@unina.it