Regression Testing

- Introduction
- Test Selection
- Test Minimization
- Test Prioritization
- Summary

What is it?

- Regression testing refers to the portion of the test cycle in which a program is tested to ensure that changes do not affect features that are not supposed to be affected.
  - Corrective regression testing is triggered by corrections made to the previous version
  - Progressive regression testing is triggered by new features added to the previous version.

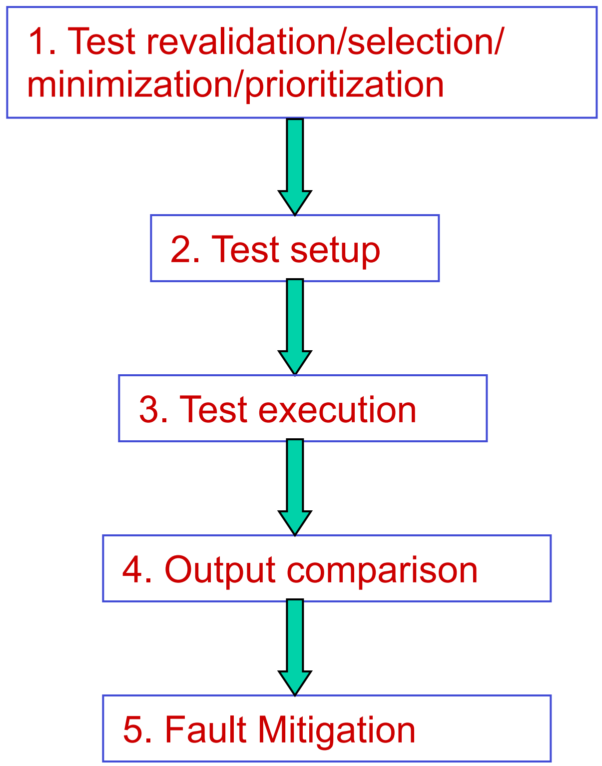
Develop-Test-Release Cycle

Version 1	Later Versions
1. Develop P 2. Test P 3. Release P	1. Modify P to P' 2. Test P' for new functionality 3. Perform regression testing on P' to ensure that the code carried over from P behaves correctly 4. Release P'

A Simple Approach

- Can we simply re-execute all the tests that are developed for the previous version?

Regression-Test Process



Major Tasks

- Test revalidation refers to the task of checking which tests for P remain valid for P' .
- Test selection refers to the identification of tests that traverse the modified portions in P' .
- Test minimization refers to the removal of tests that are seemingly redundant with respect to some criteria.
- Test prioritization refers to the task of prioritizing tests based on certain criteria.

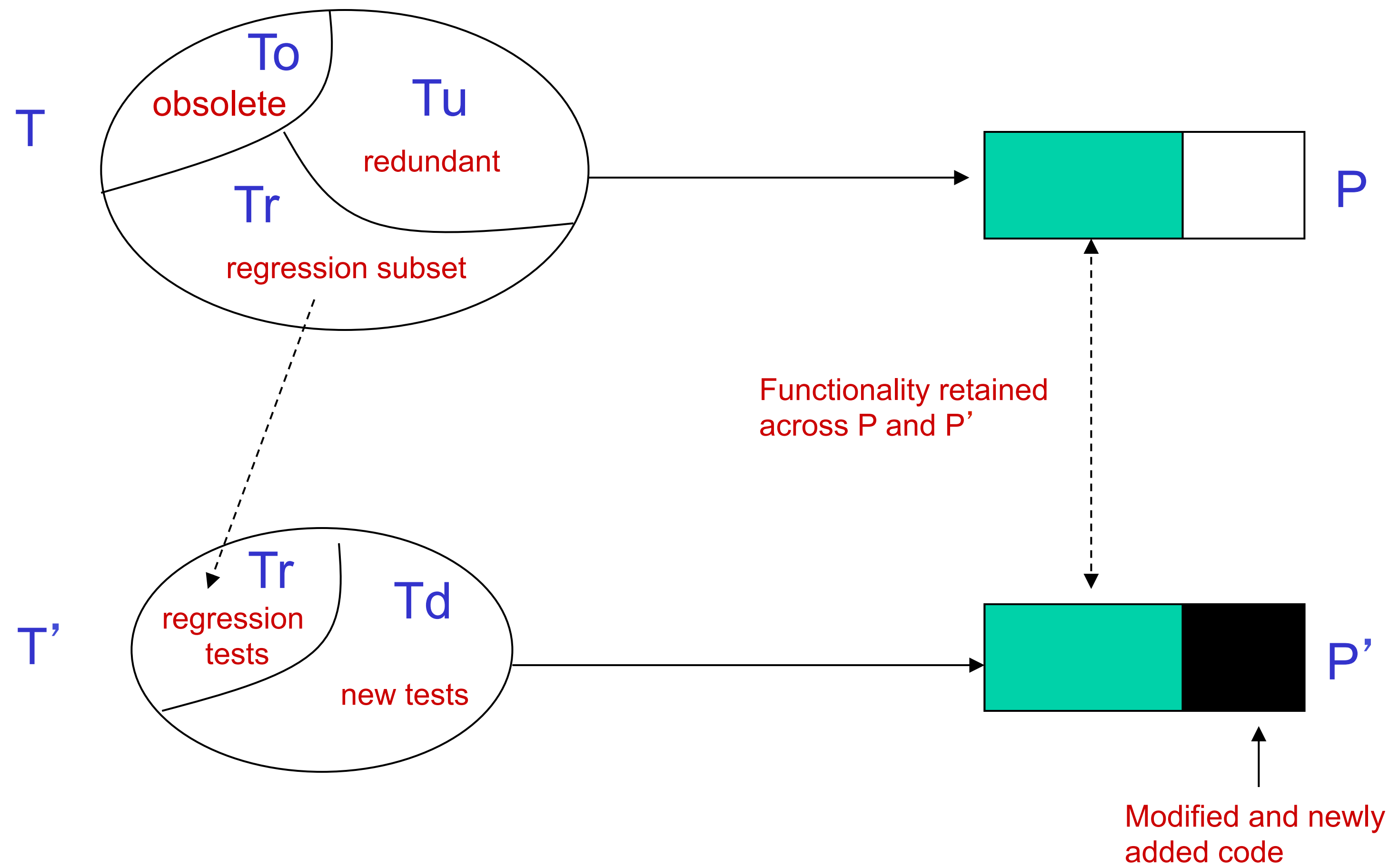
Example (1)

- Consider a web service ZipCode that provides two services:
  - ZtoC: returns a list of cities and the state for a given zip code
  - ZtoA: returns the area code for a given zip code
- Assume that ZipCode only serves the US initially, and then is modified as follows:
  - ZtoC is modified so that a user must provide a given country as well as a zip code.
  - ZtoT, a new service, is added that inputs a country and a zip code and return the time-zone.

Example (2)

- Consider the following two tests used for the original version:
  - t1: <service = ZtoC, zip = 47906>
  - t2: <service = ZtoA, zip = 47906>
- Can the above two tests be applied to the new version?

The RTS Problem (1)



The RTS Problem (2)

- The RTS problem is to find a minimal subset Tr of non-obsolete tests from T such that if P' passes tests in Tr then it will also pass tests in Tu.
- Formally, Tr shall satisfy the following property: $\forall t \in Tr$ and $\forall t' \in Tu \cup Tr$, $P(t) = P'(t) \Rightarrow P(t') = P'(t')$.

Regression Testing

- Introduction
- Test Selection
- Test Minimization
- Test Prioritization
- Summary

Main Idea

- The goal is to identify test cases that traverse the modified portions.
- Phase 1: P is executed and the trace is recorded for each test case in $Tno = Tu \cup Tr$.
- Phase 2: Tr is isolated from Tno by a comparison of P and P' and an analysis of the execution traces
  - Step 2.1: Construct CFGs and syntax trees
  - Step 2.2: Compare CFGs and select tests

Obtain Execution Traces

```
main () {
int x, y, p;
input (x, y);
if (x < y)
    p = g1(x, y);
else
    p = g2(x, y);
endif
output (p);
end
}
```

```
int g1 (int a, b) {
int a, b;
if (a + 1 == b)
     return (a*a);
else
     return (b*b);
```

```
int g2 (int a, b) {
int a, b;
if (a == (b + 1))
     return (b*b);
else
     return (a*a);
```

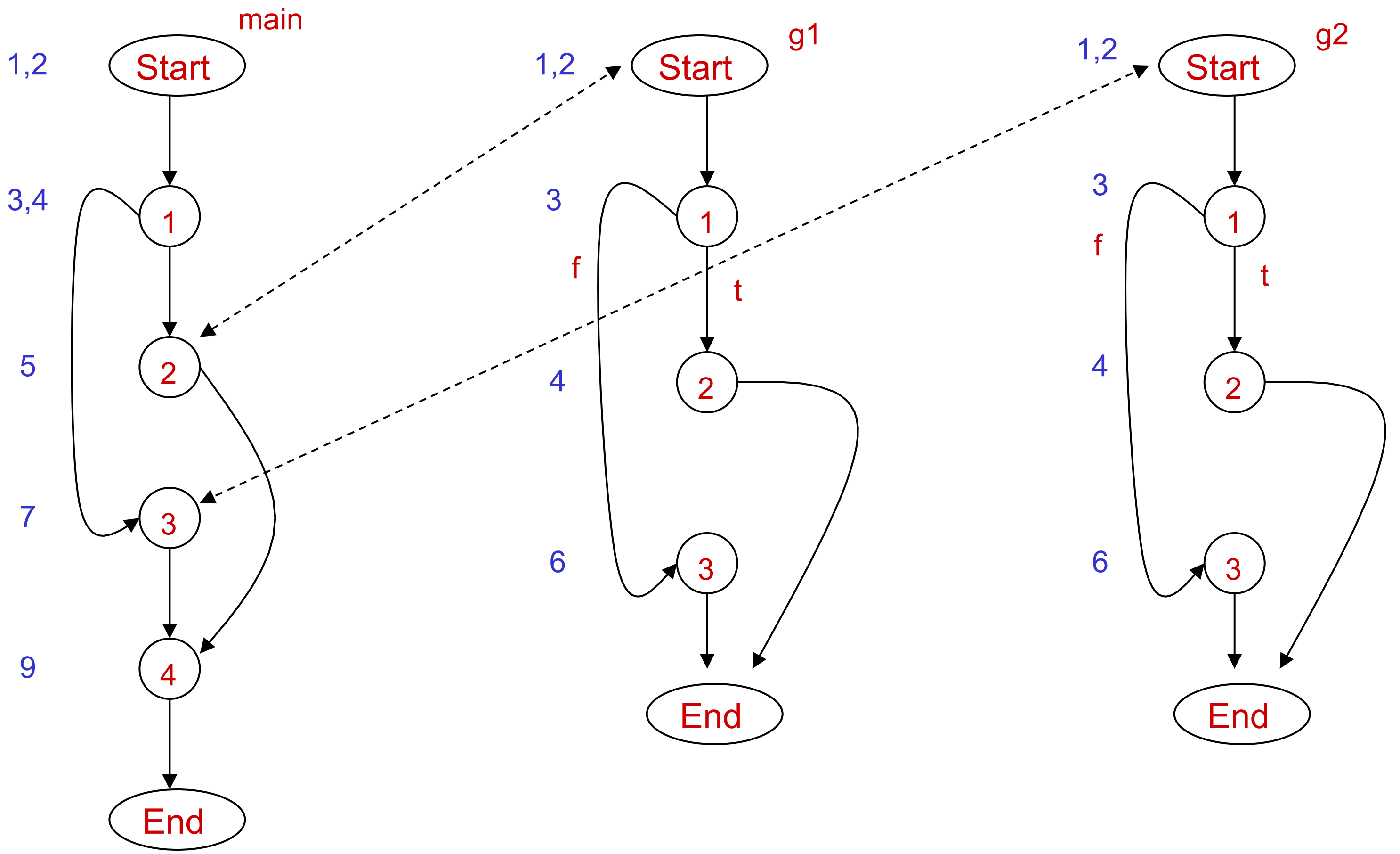
Consider the following test set:

t1: <x=1, y=3>

t2: <x=2, y=1>

t3: <x=1, y=2>

CFG



main
g1
g2
1,2
Start
3,4
1
5
2
7
3
9
4
End
1,2
Start
3
1
f
t
4
2
6
3
End
1,2
Start
3
1
f
t
4
2
6
3
End

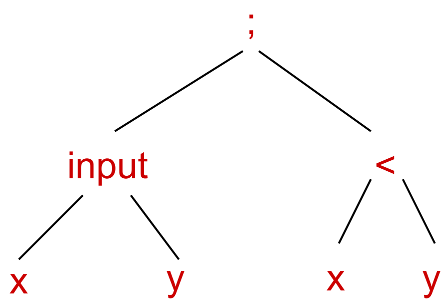
Execution Trace

Test (t)	Execution Trace (trace(t))
t1	main.Start, main.1, main.2, g1.Start, g1.1, g1.3, g1.End, main.2, main.4, main.End
t2	main.Start, main.1, main.3, g2.Start, g2.1, g2.2, g2.End, main.3, main.4, main.End
t3	main.Start, main.1, main.2, g1.Start, g1.1, g1.2, g1.End, main.2, main.4, main.End

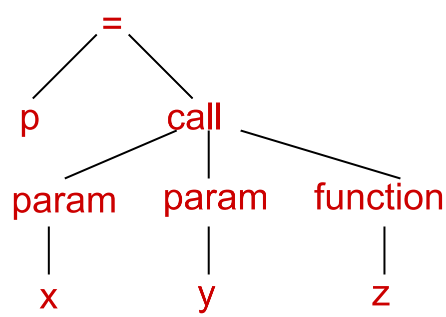
Test Vector

	Test vector (test(n)) for node n			
Function	1	2	3	4
main	t1, t2, t3	t1, t3	t2	t1, t2, t3
g1	t1, t3	t3	t1	-
g2	t2	t2	None	-

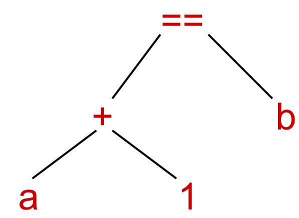
Syntax Tree



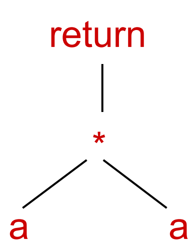
main.1



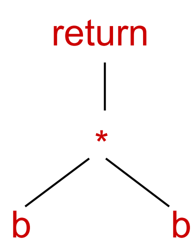
main.2



g1.1



g1.2 and g2.3



g1.3 and g2.2

Selection Strategy

- The CFGs for P and P' are compared to identify nodes that differ in P and P'.
  - Two nodes are considered equivalent if the corresponding syntax trees are identical.
  - Two syntax trees are considered identical when their roots have the same labels and the same corresponding descendants.
- Tests that traverse those nodes are selected.

Procedure SelectTestsMain

Input: (1) G and G', including syntax trees; (2) Test vector test(n) for each node n in G and G'; and (3) Set T of non-obsolete tests
Output: A subset T' of T

Procedure SelectTestsMain
  Step 1: Set T' = ∅. Unmark all nodes in G and in its child CFGs
  Step 2: Call procedure SelectTests (G.Start, G'.Start')
  Step 3: Return T' as the desired test set

Procedure SelectTests (N, N')
  Step 1: Mark node N
  Step 2: If N and N' are not equivalent, T' = T' ∪ test(N) and return, otherwise go to the next step.
  Step 3: Let S be the set of successor nodes of N
  Step 4: Repeat the next step for each n ∈ S.
    4.1 If n is marked then return else repeat the following steps:
      4.1.1 Let l = label(N, n). The value of l could be t, f or ε
      4.1.2 n' = getNode(l, N').
      4.1.3 SelectTests(n, n')
  Step 5: Return from SelectTests

Example

- Consider the previous example. Suppose that function g1 is modified as follows:

```
int g1 (int a, b) {
int a, b;
if (a - 1 == b) ← Predicate modified
    return (a*a);
else
    return (b*b);
```

Regression Testing

- Introduction
- Test Selection
- Test Minimization
- Test Prioritization
- Summary

Motivation

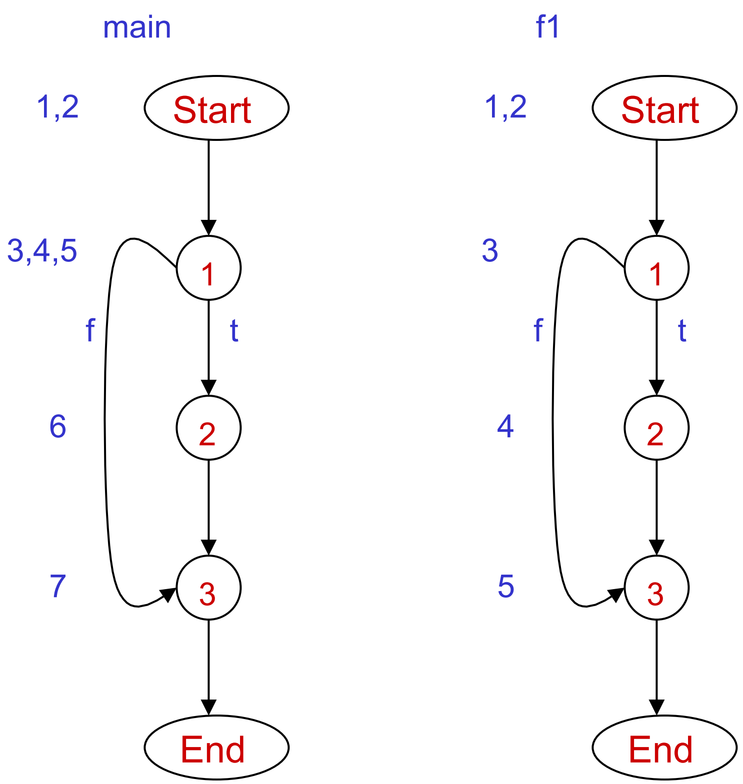
- The adequacy of a test set is usually measured by the coverage of some testable entities, such as basic blocks, branches, and du-paths.
- Given a test set T, is it possible to reduce T to T' such that $T' \subseteq T$ and T' still covers all the testable entities that are covered by T?

Example (1)

```
main () {
 int x, y, z;
 input (x, y);
 z = f1(x);
 if (z > 0)
    z = f2(x);
 output (z);
end
}
```

```
int f1(int x) {
 int p;
 if (x > 0)
    p = f3(x, y);
 return (p);
}
```

Example (2)



Consider the following test set:
t1: main: 1, 2, 3; f1 : 1, 3
t2: main: 1, 3; f1: 1, 3
t3: main: 1, 3; f1: 1, 2, 3

The Set-Cover Problem

- Let E be a set of entities and TE a set of subsets of E.
- A set cover is a subset (of subsets) $C \subseteq TE$ such that the union of all the subsets in C is E.
- The set-cover problem is to find a minimal C.

Example

- Consider the previous example:
  - E = {main.1, main.2, main.3, f1.1, f1.2, f1.3}
  - TE = {{main.1, main.2, main.3, f1.1, f1.3}, {main.1, main.3, f1.1, f1.3}, {main.1, main.3, f1.1, f1.2, f1.3}}
- The solution to the set cover problem is:
  - C = {{main.1, main.2, main.3, f1.1, f1.3}, {main.1, main.3, f1.1, f1.2, f1.3}}

A Greedy Approach

- Find a test t in T that covers the maximum number of entities in E.
- Add t to the return set, and remove it from T and the entities it covers from E
- Repeat the same procedure until all entities in E have been covered.

Procedure CMIMX

Input: An $n \times m$ matrix C, where each column corresponds to an entity to be covered, and each row to a distinct test. C(i,j) is 1 if test t_i covers entity j.

Output: Minimal cover minCov = $\{i_1, i_2, \ldots, i_k\}$ such that for each column in C, there is at least one nonzero entry in at least one row with index in minCov.

Step 1: Set minCov = ϕ, yetToCover = m.

Step 2: Unmark each of the n tests and m entities.

Step 3: Repeat the following steps while yetToCover > 0

3.1. Among the unmarked entities (columns) in C find those containing the least number of 1s. Let LC be the set of indices of all such columns.

3.2. Among all the unmarked tests (rows) in C that also cover entities in LC, find those that have the max number of nonzero entries that correspond to unmarked columns. Let s be any one of those rows.

3.3. Mark test s and add it to minCov. Mark all entities covered by test s. Reduce yetToCover by the number of entities covered by s.

Example

- Consider the previous example:

	1	2	3	4	5	6
t1	1	1	1	0	0	0
t2	1	0	0	1	0	0
t3	0	1	0	0	1	0
t4	0	0	1	0	0	1
t5	0	0	0	0	1	0

Step 1: LC = {4, 6}. t2 and t4 has two 1s. Select t2. minCov = {t2}.
Step 2: LC = {6}. Select t4, as t4 is the only one covers 6. minCov = {t2, t4}.
Step 3: LC = {2, 5}. Select t3. minCov = {t2, t4, t3}.

Regression Testing

- Introduction
- Test Selection
- Test Minimization
- Test Prioritization
- Summary

Motivation

- In practice, sufficient resources may not be available to execute all the tests.
- One way to solve this problem is to prioritize tests and only execute those high-priority tests that are allowed by the budget.
- Typically, test prioritization is applied to a reduced test set that are obtained, e.g., by the test selection and/or minimization process.

Residual Coverage

- Residual coverage refers to the number of entities that remain to be covered w.r.t. a given coverage criterion.
- One way to prioritize tests is to give higher priority to tests that lead to a smaller residual coverage.

Procedure PrTest

Input: (1) T' : a regression test set to be prioritized; (2) entitiesCov: set of entities covered by tests in T' ; (2) cov: Coverage vector such that for each test $t \in T'$, cov(t) is the set of entities covered by t.

Output: PrT: A prioritized sequence of tests in T'

Step 1: X' = T' . Find $t \in X'$ such that $|cov(t)| \geq |cov(u)|$ for all $u \in X'$.

Step 2: PrT = <t>, X' = x' \ {t}, entitiesCov = entitiesCov \ cov(t)

Step 3: Repeat the following steps while $X' \neq \phi$ and entitiesCov $\neq \phi$.

3.1. resCov(t) = |entitiesCov \ (cov(t) $\cap$ entitiesCov)|

3.2. Find test $t \in X'$ such that resCov(t) $\leq$ resCov(u) for all $u \in X'$, $u \neq t$.

3.3. Append t to Prt, X' = X' \ {t}, and entitiesCov = entitiesCov \ cov(t)

Step 4: Append to PrT any remaining tests in X' in an arbitrary order.

Example

- Consider a program P consisting of four classes C1, C2, C3, and C4. Each of these classes has one or more methods as follows: C1 = {m_1, m_{12}, m_{16}}, C2 = {m_2, m_3, m_4}, C3 = {m_5, m_6, m_{10}, m_{11}}, and C4 = {m_7, m_8, m_9, m_{13}, m_{14}, m_{15}}.

Test(t)	Methods covered (cov(t))	|cov(t)|
t1	1,2,3,4,5,10,11,12,13,14,16	11
t2	1,2,4,5,12,13,15,16	8
t3	1,2,3,4,5,12,13,14,16	9
t4	1,2,4,5,12,13,14,16	8
t5	1,2,4,5,6,7,8,9,10,11,12,13,15,16	14

1: PrT = <t5>. entitiesCov = {3, 14}
2: resCov(t1) = {}, resCov(t2) = {3, 14}, resCov(t3)= {}, resCov(t4) = {3}
PrT = <t5, t1>. entitiesCov = {}
3: PrT = <t5, t1, t2, t3, t4>

Regression Testing

- Introduction
- Test Selection
- Test Minimization
- Test Prioritization
- Summary

Summary

- Regression testing is about ensuring new changes do not adversely affect existing functionalities.
- Test selection is to select tests that execute at least one line of code that has been changed.
- Test minimization is to reduce the number of tests while preserving the same coverage.
- Test prioritization is to determine an order in which tests should be executed so that we could spend limited resources on the most important tests.