UTA

## Regression Testing

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



UTA

## What is it?

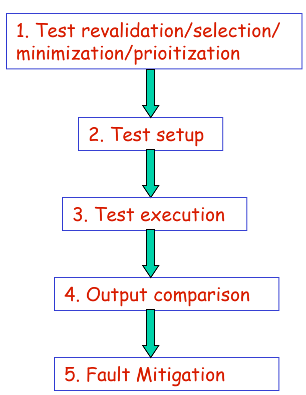
- Regression testing refers to the portion of the test cycle in which a program is tested to ensure that changes do not affect features that are not supposed to be affected.
- Corrective regression testing is triggered by corrections made to the previous version; progressive regression testing is triggered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

# Develop-Test-Release Cycle

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



# Regression-Test Process



UTA

## A Simple Approach

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



UTA

## 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.

UTA

## 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.

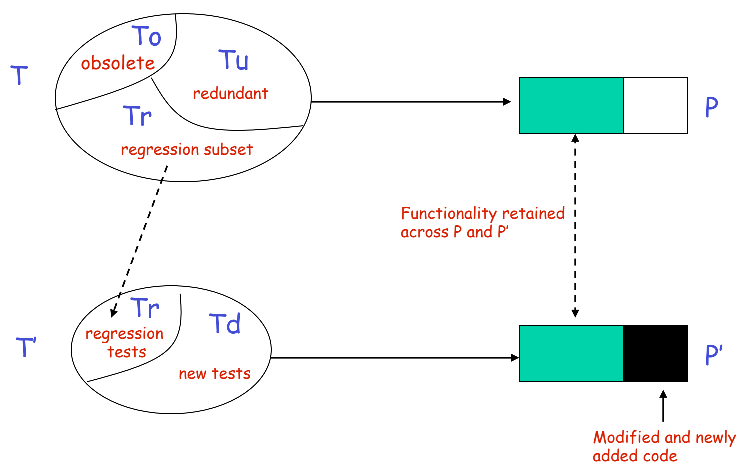


UTA

## 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 ∪ 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 CFG and syntax trees
  - Step 2.2: Compare CFGs and select tests

UTA

# Obtain Execution Traces

```
1.  main () {
2.   int x, y, p;
3.   input (x, y);
4.   if (x < y)
5.      p = g1(x, y);
6.   else
7.      p = g2(x, y);
8.   endif
9.   output (p);
10. end
11. }
```

```
1.  int g1 (int a, b) {
2.  int a, b;
3.  if (a + 1 == b)
4.      return (a*a);
5.  else
6.      return (b*b);
```

```
1.  int g2 (int a, b) {
2.  int a, b;
3.  if (a == (b + 1))
4.      return (b*b);
5.  else
6.      return (a*a);
```

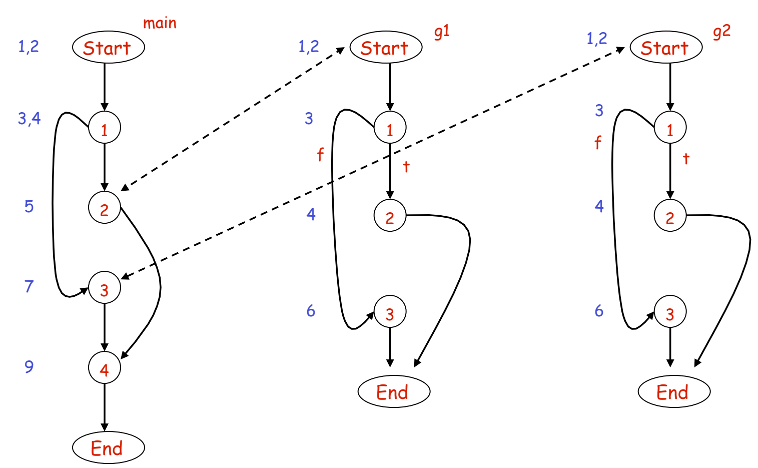
Consider the following test set:

t1: <x=1, y=3>
t2: <x=2, y=1>
t3: <x=1, y=2>



UTA

# CFG



UTA

## 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 |

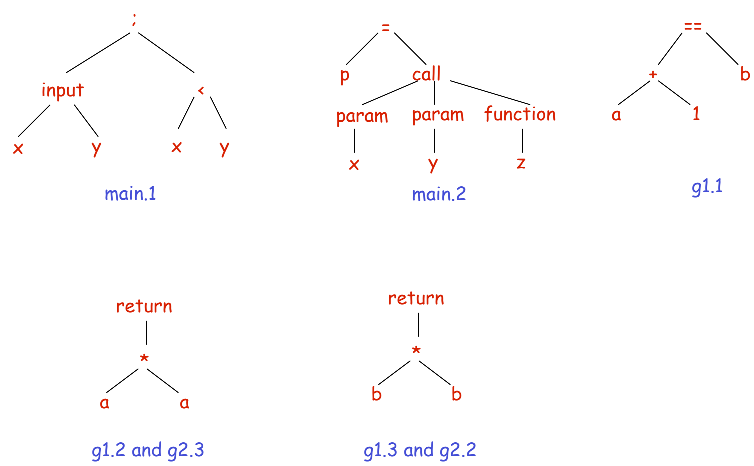


UTA

## 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






# 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



UTA

## 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?

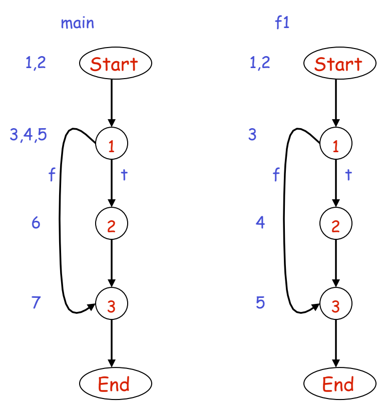
# Example (1)

```
1. main () {
2.   int x, y, z;
3.   input (x, y);
4.   z = f1(x);
5.   if (z > 0)
6.      z = f2(x);
7.   output (z);
8. end
9. }
```

```
1. int f1(int x) {
2.   int p;
3.   if (x > 0)
4.      p = f3(x, y);
5.   return (p);
6. }
```



# 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

UTA

## The Set-Cover Problem

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



UTA

## 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, f1.2, 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 Algorithm

- 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 $minCov = \phi$, $yetToCover = m$.

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.

UTA

## 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 |



UTA

## 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 elements 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.

UTA

# 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) ∩ 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.



UTA

# 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 | 1,2,4,5,6,7,8,10,11,12,13,15,16 | 13 |

## Regression Testing

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





## Summary

- Regression testing is about ensuring new changes do not adversely affect existing functionalities.
- Three techniques can be used to reduce the number of regression tests: modification-traversing selection, minimization, and prioritization.
- Modification-traversing selection and minimization do not reduce coverage, but prioritization does. The latter is however a practical choice when resources are limited.