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

<table>
<thead>
<tr>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Develop P</td>
<td>1. Modify P to P'</td>
</tr>
<tr>
<td>2. Test P</td>
<td>2. Test P' for new functionality</td>
</tr>
<tr>
<td>3. Release P</td>
<td>3. Perform regression testing on P' to ensure that the code carried over from P behaves correctly</td>
</tr>
<tr>
<td></td>
<td>4. Release P'</td>
</tr>
</tbody>
</table>

Regression-Test Process

1. Test revalidation/selection/minimization/prioritization
2. Test setup
3. Test execution
4. Output comparison
5. Fault Mitigation
**A Simple Approach**

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

**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:
- \( t1: \langle \text{service} = ZtoC, \text{zip} = 47906 \rangle \)
- \( t2: \langle \text{service} = ZtoA, \text{zip} = 47906 \rangle \)

Can the above two tests be applied to the new version?
The RTS Problem (1)

The RTS problem is to find a minimal subset $T_r$ of non-obsolete tests from $T$ such that if $P'$ passes tests in $T_r$ then it will also pass tests in $T_u$.

Formally, $T_r$ shall satisfy the following property:

$\forall t \in T_r$ and $\forall t' \in T_u \cup T_r$, $P(t) = P'(t) \Rightarrow P(t') = P'(t')$.

The RTS Problem (2)

- The RTS problem is to find a minimal subset $T_r$ of non-obsolete tests from $T$ such that if $P'$ passes tests in $T_r$ then it will also pass tests in $T_u$.

- Formally, $T_r$ shall satisfy the following property:

$\forall t \in T_r$ and $\forall t' \in T_u \cup T_r$, $P(t) = P'(t) \Rightarrow P(t') = P'(t')$. 
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 $T_{no} = T_u \cup T_r$.
- Phase 2: $T_r$ is isolated from $T_{no}$ 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
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

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. **}**

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);
1. **}**

Consider the following test set:
- t1: <x=1, y=3>
- t2: <x=2, y=1>
- t3: <x=1, y=2>

**CFG**

Software Testing and Maintenance

13
**Execution Trace**

<table>
<thead>
<tr>
<th>Test (t)</th>
<th>Execution Trace (trace(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>main.Start, main.1, main.2, g1.Start, g1.1, g1.3, g1.End, main.2, main.4, main.End</td>
</tr>
<tr>
<td>t2</td>
<td>main.Start, main.1, main.3, g2.Start, g2.1, g2.2, g2.End, main.3, main.4, main.End</td>
</tr>
<tr>
<td>t3</td>
<td>main.Start, main.1, main.2, g1.Start, g1.1, g1.2, g1.End, main.2, main.4, main.End</td>
</tr>
</tbody>
</table>

**Test Vector**

<table>
<thead>
<tr>
<th>Test vector (test(n)) for node n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>main</td>
</tr>
<tr>
<td>g1</td>
</tr>
<tr>
<td>g2</td>
</tr>
</tbody>
</table>
Syntax Tree

Input:
- main.1:
  - input: x, y
  - main: x, y

- main.2:
  - call: param, param
  - function: x, y, z
  - a, 1

Output:
- g1.1:
  - return
  - a

- g1.2 and g2.3:
  - return
  - b

- g1.3 and g2.2:
  - return
  - a

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 SelectTests

**Input**: (1) \( G \) and \( G' \), including syntax trees; (2) Test vector \( \text{test}(n) \) for each node \( n \) in \( G \) and \( G' \); and (3) Set \( T \) of non-obsolete tests

**Output**: A subset \( T' \) of \( T \)

**Procedure SelectTestsMain**

1. Set \( T' = \emptyset \). Unmark all nodes in \( G \) and in its child CFGs.
2. Call procedure SelectTests (\( G.\text{Start} \), \( G'.\text{Start} \))
3. Return \( T' \) as the desired test set

**Procedure SelectTests (N, N')**

1. Mark node \( N \)
2. If \( N \) and \( N' \) are not equivalent, \( T' = T' \cup \text{test}(N) \) and return, otherwise go to the next step.
3. Let \( S \) be the set of successor nodes of \( N \)
4. Repeat the next step for each \( n \in S \).
   1. If \( n \) is marked then return else repeat the following steps:
      1.1 Let \( l = \text{label}(N, n) \). The value of \( l \) could be \( t \), \( f \) or \( \epsilon \)
      1.2 \( n' = \text{getNode}(l, N) \)
      1.3 SelectTests(\( n, n' \))
4. Return from SelectTests

**Example**

Consider the previous example. Suppose that function \( g1 \) is modified as follows:

```c
1. int g1(int a, b) {
2.     int a, b;
3.     if (a - 1 == b) \{ \text{Predicate modified} \}
4.         return (a*a);
5.     else
6.         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)

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. }`

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

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 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 $C$.

Example

- Consider the previous example:
  - $E = \{\text{main.1, main.2, main.3, f1.1, f1.2, f1.3}\}$
  - $TE = \{\{\text{main.1, main.2, main.3, f1.1, f1.3}\}, \{\text{main.1, main.3, f1.1, f1.2, f1.3}\}, \{\text{main.1, main.3, f1.1, f1.2, f1.3}\}\}$
- The solution to the set cover problem is:
  - $C = \{\{\text{main.1, main.2, main.3, f1.1, f1.3}\}, \{\text{main.1, main.3, f1.1, f1.2, f1.3}\}\}$
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 $\text{minCov} = \{i_1, i_2, \ldots, i_k\}$ such that for each column in $C$, there is at least one nonzero entry in at least one row with index in $\text{minCov}$.

Step 1: Set $\text{minCov} = \emptyset$, $\text{yetToCover} = m$.
Step 2: Unmark each of the $n$ tests and $m$ entities.
Step 3: Repeat the following steps while $\text{yetToCover} > 0$.

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

3.2. Among all the unmarked tests (rows) in $C$ that also cover entities in $\text{LC}$, 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 $\text{minCov}$. Mark all entities covered by test $s$. Reduce $\text{yetToCover}$ by the number of entities covered by $s$. 
**Example**

- Consider the previous example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>
  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 |

**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.
Procedure PrTest

Input: (1) $T$: a regression test set to be prioritized; (2) $\text{entitiesCov}$: set of entities covered by tests in $T$; (2) $\text{cov}$: Coverage vector such that for each test $t \in T$, $\text{cov}(t)$ is the set of entities covered by $t$.

Output: $\text{PrT}$: A prioritized sequence of tests in $T$

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

Step 2: $\text{PrT} = \langle t \rangle$, $X = X' \setminus \{t\}$, $\text{entitiesCov} = \text{entitiesCov} \setminus \text{cov}(t)$

Step 3: Repeat the following steps while $X \neq \emptyset$ and $\text{entitiesCov} \neq \emptyset$.

3.1. $\text{resCov}(t) = |\text{entitiesCov} \setminus (\text{cov}(t) \cap \text{entitiesCov})|

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

3.3. Append $t$ to $\text{PrT}$, $X = X \setminus \{t\}$, and $\text{entitiesCov} = \text{entitiesCov} \setminus \text{cov}(t)$

Step 4: Append to $\text{PrT}$ any remaining tests in $X'$ in an arbitrary order.

Example

Consider a program $P$ consisting of four classes $C_1$, $C_2$, $C_3$, and $C_4$. Each of these classes has one or more methods as follows: $C_1 = \{m_1, m_{12}, m_{16}\}$, $C_2 = \{m_2, m_3, m_4\}$, $C_3 = \{m_5, m_6, m_{10}, m_{11}\}$, and $C_4 = \{m_7, m_8, m_9, m_{13}, m_{14}, m_{15}\}$.

| 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.