SYSC 4101 / 5105

Regression Testing

Main resources:

- Mathur, A.P., "*Foundations of Software Testing*", Pearson, 2008
- Leung, K.N., White, L., "Insights into Regression Testing", *Proc. IEEE International Conference on Software Maintenance (ICSM)*, pp. 60-69, 1989.
- Rothermel, G, Harrold, M.J., "A Safe, Efficient Regression Test Selection Technique," ACM Transactions on Software Engineering and Methodology, 6(2), pp.173-210, 1997

Terminology

- Regress = returning to a previous, usually worse, state
- Regression testing
  - Portion of the testing life cycle
  - During which program P has been modified into program P' (maintenance)
  - P' needs specific testing attention to ensure that
    - newly added or modified code behaves correctly
    - code carried over unchanged, continues to behave correctly
- Important problem:
  - Regression testing constitute the vast majority of testing effort in commercial software development.

Terminology (cont.)

- Software Maintenance
  - The process of modifying a software system or component
- Why?
  - Corrective maintenance (repairing faults),
  - Perfective maintenance (changes demanded by client)
  - Adaptive maintenance (adapt to new hardware/software environment),
  - Preventive maintenance (foreseeable client requests, hardware/software environment change)
- Consequences:
  - Added code, Modified code, Deleted code, Unchanged code.
    - Most of regression testing is currently white-box.
  - Added, modified, deleted, unchanged (UML) model elements.
    - There is a trend to do black-box (or grey-box) regression testing.

Regression Testing vs. Testing

Regression Testing is not a simple extension of testing.
The main differences are:

1. Availability of test plan
   - Testing starts with a specification, an implementation of the specification and a test plan (black-box and/or white-box test cases). Everything is new.
   - Regression testing starts with a (possibly modified) specification, a modified program, and an old test plan (which requires updating).
2. Scope of test
   - Testing aims to check the correctness of the whole program.
   - Regression testing aims to check (modified) parts of the program.
3. Time allocation
   - Testing time is normally budgeted before the development of a product (part of development costs).
   - Regression testing time is not normally included in the total product cost (problem!).

Regression Testing vs. Testing (cont.)

4. Development information
   - During testing, knowledge about the development process is available (it is part of it)
   - During regression testing, the testers may not be the ones who developed/test the previous version!
5. Completion time
   - Completion time for regression testing should normally be less than that for testing (only parts are tested)
6. Frequency
   - Testing occurs frequently during code production
   - Regression testing occurs many times throughout the life of a product, once after every modification is made to it

Two Phases of Product Development

Version 1 of software product

1. Develop P with specification S
2. Test P with test set T
3. Release P

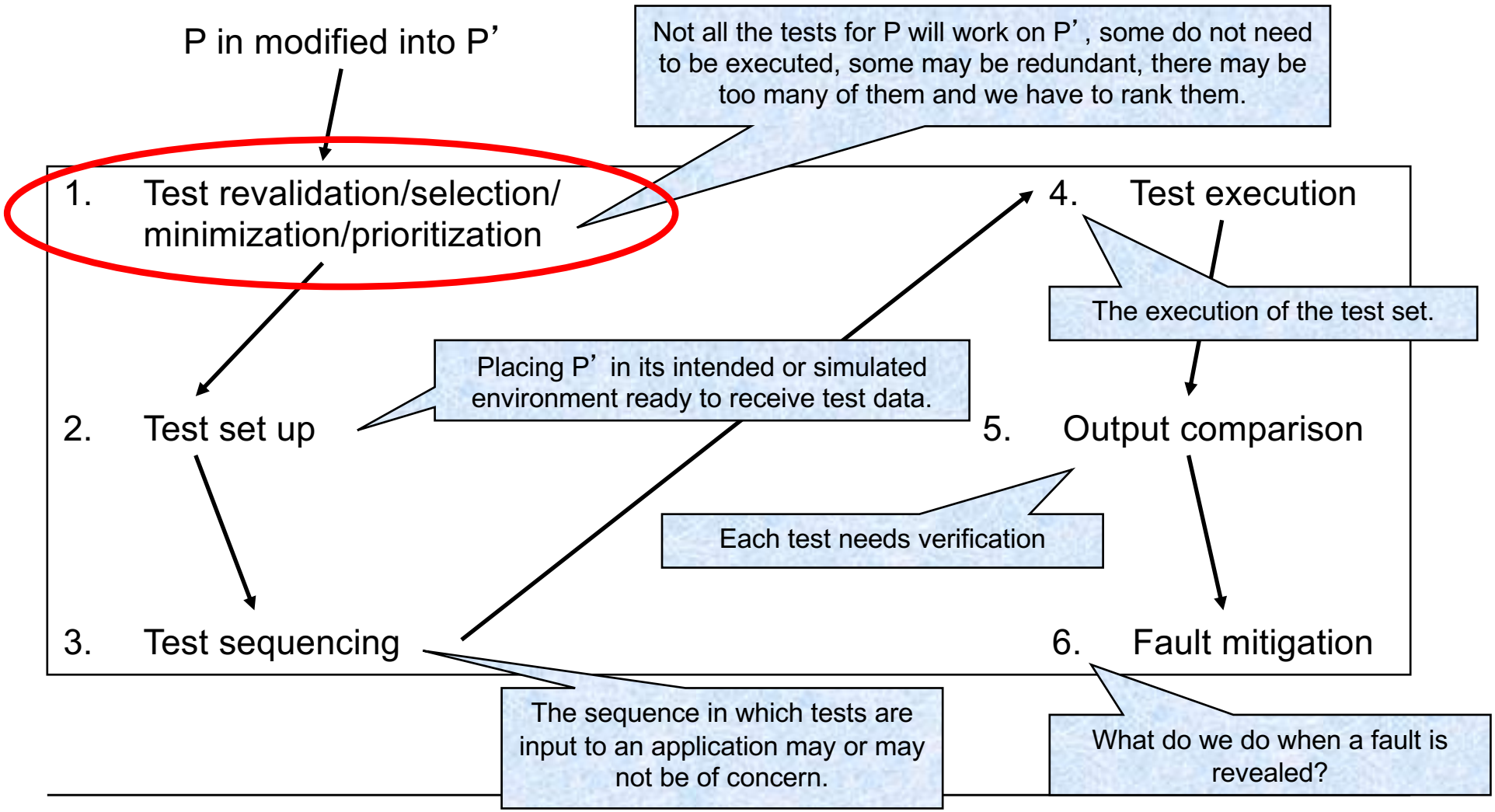
Version 2 of software product

1. Modify P into P'(specification S')
2. Test P' for new functionalities
   - Black-box + white box testing or …
   - Build T"
3. Perform regression testing on P' to ensure that the code carried over from P behaves correctly
   - Main principle: reuse!
     - Reusing tests derived for P
     - Reusing test scaffolding built for P
   - Identifying T', a subset of T

$\Rightarrow$ P' is tested with T'$\cup$T"

4. Release P'

Regression Testing Process



Regression Testing Process

- Test Revalidation
- (Regression) Test Selection
- Test Minimization
- Test Prioritization

Test Revalidation

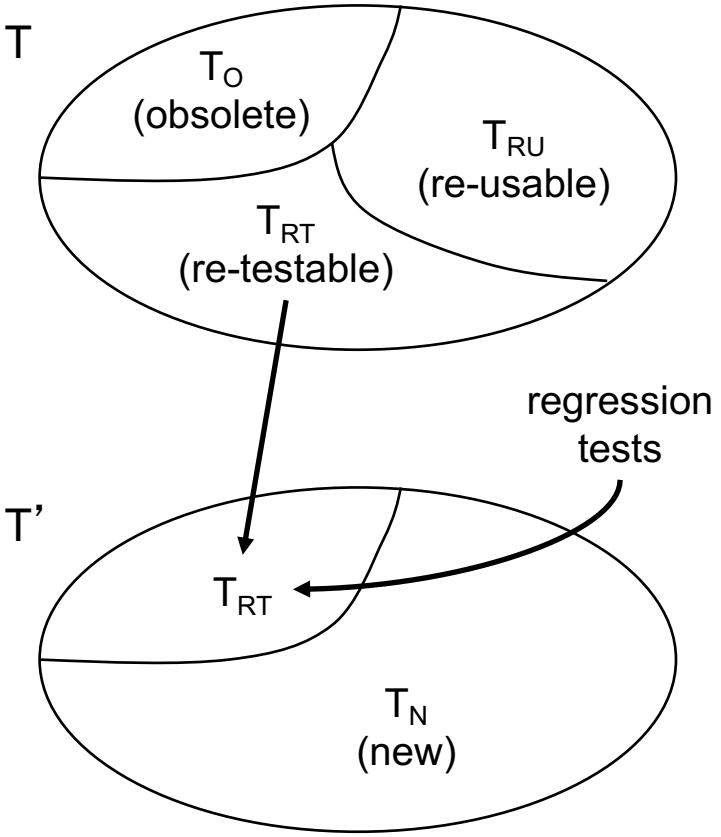
- Changes may be so drastic that tests in T won't run on P'.
- Obsolete Test Cases:
  - Black-box or white-box test cases in T that can no longer be used on P'
- Three ways that a test case may become obsolete:
  - Program modification leads to (now) incorrect input/output relation
  - If P's component has been modified, some test cases may correctly specify the input/output relation, but may not be testing the same construct.
    - E.g., the modifications from P to P' change some equivalence classes. $t \in T$ used to exercise a boundary, which is no longer a boundary. $t \in T$ is obsolete as it does no longer test a construct (boundary) of interest.
  - A structural test case $t \in T$ may no longer contribute to the structural coverage of the program.

Regression Testing Process

- Test Revalidation
- (Regression) Test Selection
- Test Minimization
- Test Prioritization

Regression Test Selection

- Regression Test Selection
  - Should we select and execute tests in T that do not traverse modified parts of P'?
  - Classification of tests:
    - Re-testable: should be re-run
    - Reusable: may not need to be re-run.
- Test Selection Problem
  - Find a test set T_{RT} on which P' is to be tested to ensure that code that implements functionalities carried over from P work correctly.
- In addition P' must be tested to ensure that newly added code behaves correctly.



Regression Test Selection Techniques

- Test All
  - $T' = T - T_O$
  - Often not a practical solution as $T - T_O$ is often too large (too many test cases to run in the amount of time allocated)
- Random Selection
  - Select randomly tests from $T - T_O$
  - The tester decides how many tests to select
  - May turn out to be better than no regression testing at all
  - But may not even select tests that exercise modified parts of the code!
- Selecting Modification Traversing Tests
  - Selecting a subset of $T - T_O$ such that only tests that guarantee the execution of modified code and code that might be impacted by the modified code in P' are selected (T_{RT}). (The remaining ones are T_{RU}.)
  - A technique that does not discard any test that will traverse a modified statement is known as a "**safe**" regression test selection technique.

Test Selection Using Execution Traces

1. P is (has been) executed and execution traces are (have been) recorded (traces must have been recorded otherwise, no selection possible)
   - Execution trace = sequence of statements hit during one execution
   - trace(t) = execution trace of test case t.
   - test(n) = set of tests that hit node n at least once.

   We consider execution traces for $T - T_O$.
2. P' is compared with P
   - Build the control flow graphs G and G' of P and P', respectively
     - G and G' account for control flow, as well as function calls and variable declarations
   - Identify nodes (edges) in P and P' that are equivalent or not
     - Different techniques exist
3. Selection
   - For each node $n \in G$ that does not have an equivalent node in G'
   - $T' = T' \cup test(n)$

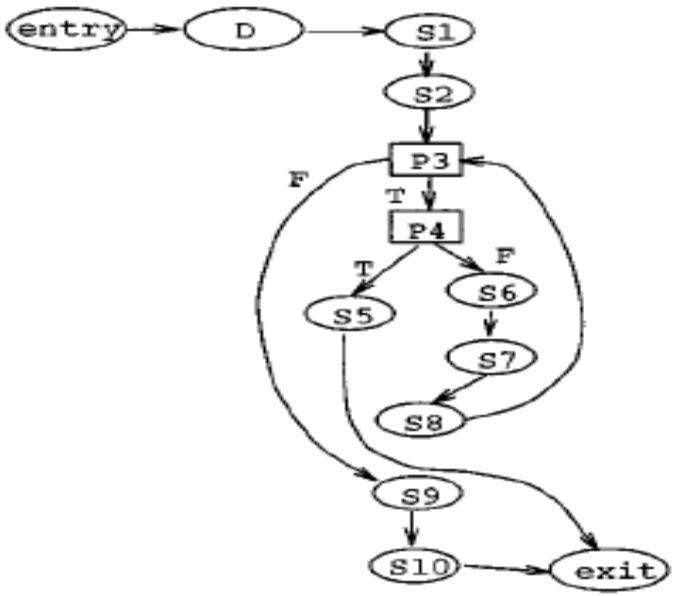
Selection Algorithm

```
algorithm   SelectTests(P, P', T):T'
input       P, P': base and modified versions of a procedure
            T: a test set used to test P
output      T': the subset of T selected for use in regression testing P'
1.  begin
2.    T' = φ
3.    construct G and G', CFGs for P and P', with entry nodes E and E'
4.    Compare(E, E')
5.    return T'
6.  end

procedure   Compare(N, N')
input       N and N': nodes in G and G'
7.  begin
8.    mark N "N'-visited"
9.    for each successor C of N in G do
10.     L = the label on edge (N, C) or ε if (N, C) is unlabeled
11.     C' = the node in G' such that N', C') has label L
12.     if C is not marked "C'-visited"
13.       if !Equivalent (C, C')
14.         T' = T' ∪ TestsOnEdge ((N, C))
15.       else
16.         Compare(C, C')
17.       endif
18.     endif
19.   endfor
20. end
```

Determines whether the statements S and S' associated with N and N' are lexicographically equivalent.

Testing Program P




```
      Procedure avg
S1.   count = 0
S2.   fread(fileptr,n)
P3.   while (not EOF) do
P4.      if (n<0)
S5.         return(error)
         else
S6.         numarray[count] = n
S7.         count++
         endif
S8.      fread(fileptr,n)
      endwhile
S9.   avg = calcavg(numarray,count)
S10.  return(avg)
```

Computes the average of positive values read from a file.

Testing Program P (cont.)

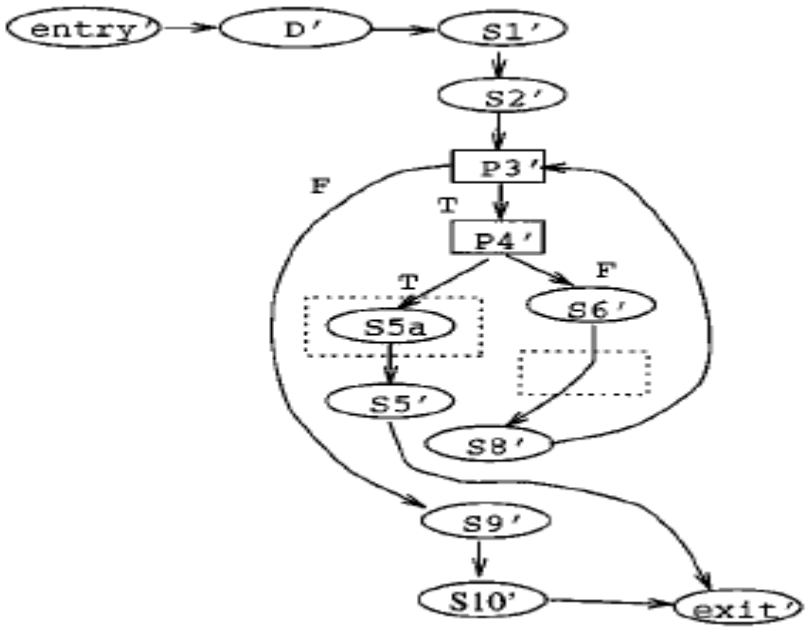
Test Information			
Test	Type	Output	Edges Traversed
t1	Empty File	0	(entry, D), (D, S1), (S1, S2) (S2, P3) (P3, S9), (S9, S10), (S10, exit)
t2	−1	Error	(entry, D) (D, S1), (S1, S2), (S2, P3), (P3, P4), (P4, S5), (S5, exit)
t3	1 2 3	2	(entry, D) (D, S1), (S1, S2), (S2, P3), (P3, P4), (P4, S6), (S6, S7), (S7, S8), (S8, P3), (P3, S9), (S9, S10), (S10, exit)

3 test cases

Coverage of edges:
011 for (P3,P4) means that only the 2nd and 3rd test cases cover this edge

Test History	
Edge	TestsOnEdge(edge)
(entry, D)	111
(D, S1)	111
(S1, S2)	111
(S2, P3)	111
(P3, P4)	011
(P3, S9)	101
(P4, S5)	010
(P4, S6)	001
(S5, exit)	010
(S6, S7)	001
(S7, S8)	001
(S8, P3)	001
(S9, S10)	101
(S10, exit)	101

Modified Program P'




```
Procedure avg2

S1'.  count = 0
S2'.  fread(fileptr,n)
P3'.  while (not EOF) do
P4'.     if (n<0)
S5a.        print("bad input")
S5'.        return(error)
         else
S6'.        numarray[count] = n
         endif
S8'.     fread(fileptr,n)
      endwhile
S9'.  avg = calcavg(numarray,count)
S10'.return(avg)
```

Selection Algorithm

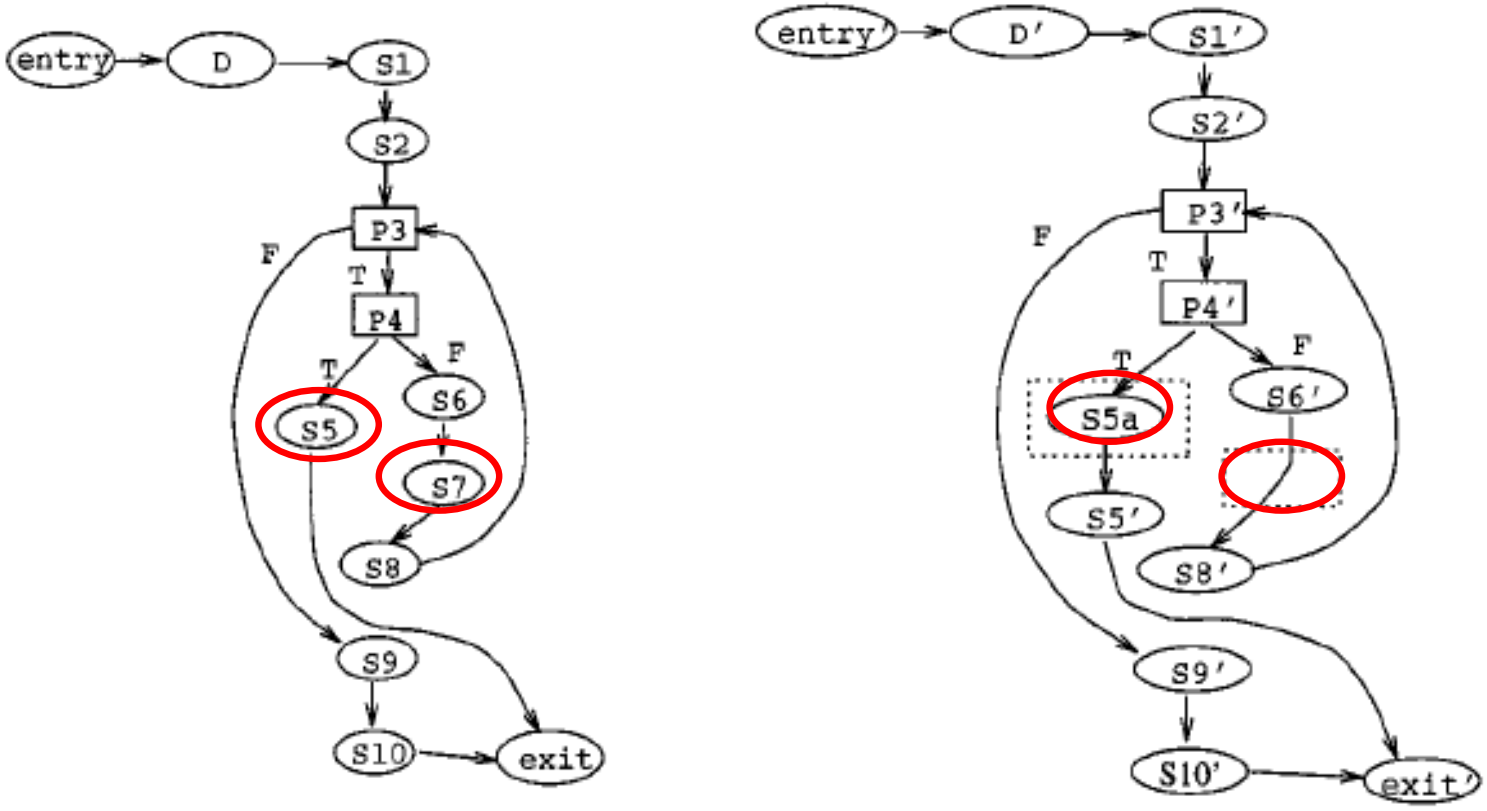
```
     Procedure avg
S1.  count = 0
S2.  fread(fileptr,n)
P3.  while (not EOF) do
P4.     if (n<0)
S5.        return(error)
        else
S6.        numarray[count] = n
S7.        count++
        endif
S8.     fread(fileptr,n)
     endwhile
S9.  avg = calcavg(numarray,count)
S10. return(avg)
```

Not lexicographically equivalent: Add test case that traverses (P4,S5), i.e.,t2

Not lexicographically equivalent: Add test case that traverses (S6,S7), i.e.,t3

```
     Procedure avg2
S1'. count = 0
S2'. fread(fileptr,n)
P3'. while (not EOF) do
P4'.    if (n<0)
S5a.       print("bad input")
S5'.       return(error)
        else
S6'.       numarray[count] = n
        endif
S8'.    fread(fileptr,n)
     endwhile
S9'. avg = calcavg(numarray,count)
S10'.return(avg)
```

$$T' = \{t2, t3\}$$


entry
D
S1
S2
P3
F
T
P4
T
F
S5
S6
S7
S8
S9
S10
exit
entry'
D'
S1'
S2'
P3'
F
T
P4'
T
F
S5a
S6'
S5'
S8'
S9'
S10'
exit'

Regression Testing Process

- Test Revalidation
- (Regression) Test Selection
- Test Minimization
- Test Prioritization

Test Minimization

- Regression test selection finds a subset T' of T.
- Suppose p contains n testable entities
  - Functions, basic blocks, conditions, DU-pairs, …
- Suppose that tests in T' cover m<n of the testable entities
  - There is likely an overlap amongst the entities covered by two tests in T'
- Is it possible (and beneficial) to reduce T' to T_{min}
  - Such that $T_{min} \subset T'$
  - Each of the m entities covered by tests in T' are also covered by tests in T_{min}

- Question
  - Will T_{min} have the same fault detection effectiveness as T'?
- Answer
  - It depends on the modifications (from P to P'), the faults, the entities used.

Algorithms for Test Minimization

- Naïve algorithm
  - Compute all the subsets of T' of size 1, size 2, size 3, … and stop when we have found one that covers all the entities covered by T'.
- Greedy algorithm

  (A greedy algorithm follows the heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum.)

  1. Find $t \in T'$ that covers the maximum number of entities
  2. $T_{min} = T_{min} \cup \{t\}$
  3. Remove t from T', remove the covered entities from consideration
  4. Repeat from step 1
- Other algorithms (e.g., genetic algorithms)

Regression Testing Process

- Test Revalidation
- (Regression) Test Selection
- Test Minimization
- Test Prioritization

Test Prioritization

- After regression test selection, T' might be overly large for testing P'(even after minimization, if any)
  - Not enough budget to execute all those tests.
- Test prioritization
  1. Ranking tests (1st, 2nd, …)
  2. Deciding to stop execution of tests after the nth ranked test
- Test prioritization requires a criterion, or criteria for ranking
- Single criterion prioritization
  - Criteria 1: cost (e.g., execution time)
    - Tests with lower costs are ranked first while test with higher costs are ranked last
  - Criteria 2: risk (expected risk of not executing a test)
    - Tests with higher risks are ranked first while test with lower risks are ranked last
- Multi-criteria prioritization
  - Finding a trade-off between cost and risk

Prioritization with AHP

- Analytical Hierarchy Process (AHP)
  - Originally designed to prioritize requirements
  - To use the expert knowledge
  - Comparing pairs and using information on pairs rather than ranking everything at once
    - Easier to compare pairs than to rank everything

1. Pair-wise comparison of tests, assigning a value to each pair
   - Different dimensions to compare pair (i,j): business value, risk of fault-proneness, cost, frequency of use (by users)

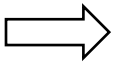
Numerical value	Explanation
1	Two test cases have equal importance.
3	Test case i has a slightly higher importance value than test case j.
5	Test case i has a strongly higher importance value than test case j.
7	Test case i has a very strongly higher importance value than test case j.
9	Test case i has an absolute higher importance value than test case j.
Reciprocals	If test case i has one of the numerical values when it compares with test case j, then test case j has the reciprocal value when compared with i.

Prioritization with AHP (cont.)

2. Build a table
   - Rows and columns are test cases
   - Cells show the evaluation of pairs.
3. Modify table
   i. Sum columns
   ii. Divide cells by column sum
   iii. Compute sum for row
   iv. Divide row sum by number of test cases and obtain a priority

	TC1	TC2	TC3	TC4
TC1	1	1/3	5	7
TC2	3	1	7	9
TC3	1/5	1/7	1	1/3
TC4	1/7	1/9	3	1

1	TC1	TC2	TC3	TC4	Sum	Priority
TC1	0.230	0.208	0.3125	0.404	1.1545	0. 29
TC2	0.690	0.629	0.4375	0.519	2.2755	0.57
TC3	0.046	0.088	0.0625	0.019	0.2155	0.05
TC4	0.033	0.069	0.1875	0.058	0.3475	0.09
Sum	4.342	1.587	16	17.33	N/A	N/A



Prioritization:
TC2, TC1, TC4, TC3

Scalability issue:
Instead of comparing test case pairs, compare the corresponding use cases.

Regression Testing Process

Revalidation / Test Selection / Minimization / Prioritization

- Comments:
  - Minimization and prioritization can be considered selection techniques.
  - Minimization and prioritization techniques can be used during testing P (not necessarily during regression testing).
- Test minimization is risky
  - Tests removed from T' might be important for finding faults in P'
  - Minimization techniques are not necessarily safe
    - One could discard a test that hit a modified part of the code.

Discussion

- Regression testing requires tools (automation)
  - Instrumentation tool to compute traces
  - Tools to build control flow graphs, build traces and slices
  - Tools implementing selection, minimization, prioritization algorithms
- "Un-automated regression testing is equivalent to no regression testing."
- Existing tools:
  - Capture/replay for GUIs.
  - DejaVOO (Java): research
  - Telcordia Software Visualization and Analysis Toolsuite (xSuds)
  - …
- Issue: up to date coverage information
- Most research and tools perform white-box regression testing (analysis of source code)
  - There is a trend to build tools for black/grey-box (UML) regression testing