Regression Testing

Dr. Mark C. Paulk

SE 4367 – Software Testing, Verification, Validation, and Quality Assurance

Phases of Testing Topics

Part IV. Phases of Testing

9. Test Selection, Minimization, and Prioritization for Regression Testing

10. Unit Testing

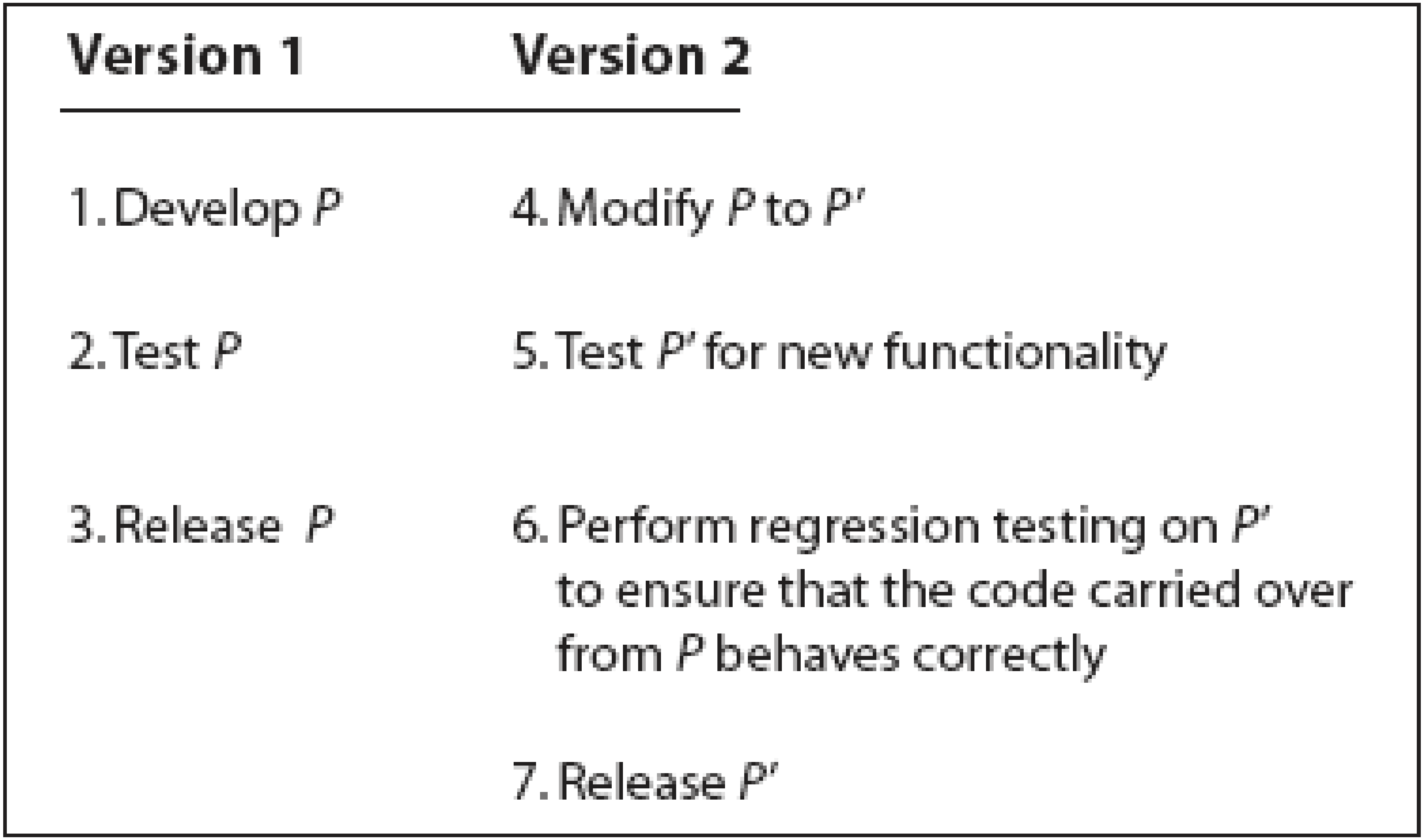
11. Integration Testing

Regression Testing

Selective re-testing of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

(IEEE 610)

The Regression Testing Process



Version 1	Version 2
1. Develop *P*	4. Modify *P* to *P′*
2. Test *P*	5. Test *P′* for new functionality
3. Release *P*	6. Perform regression testing on *P′* to ensure that the code carried over from *P* behaves correctly
	7. Release *P′*

Which Tests to Use?

Idea 1

All valid tests from the previous version and new tests created to test any added functionality.

- **the TEST-ALL approach**

What are the strengths and shortcomings of this approach?

The Test-All Approach

The test-all approach is best when you want to be certain that the new version works on all tests developed for the previous version and any new tests.

But what if you have limited resources to run tests and have to meet a deadline?

What if running all tests, as well as meeting the deadline, is simply not possible?

- **10-minute target in Extreme Programming 2004**

Test Selection

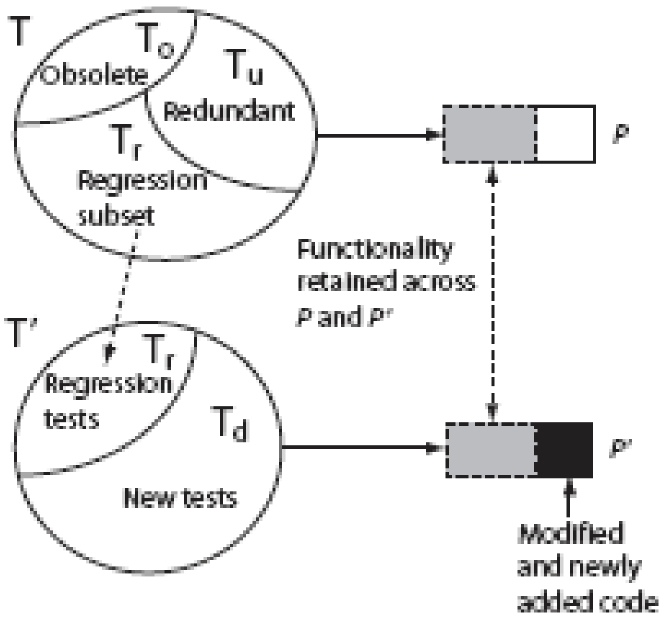
Idea 2

Select a subset T_r of the original test set T such that successful execution of the modified code P' against T_r implies that all the functionality carried over from the original code P to P' is intact.

Finding T_r can be done using several methods.

- **test minimization**
- **test prioritization**

Regression Test Selection Problem



Given test set T, our goal is to determine T_r such that successful execution of P' against T_r implies that modified or newly added code in P' has not broken the code carried over from P.

Note that some tests might become obsolete when P is modified to P'.

- **such tests are not included in the regression subset T_r**
- **the task of identifying such obsolete tests is known as <u>test revalidation</u>**

Execution Trace and Slice Method

Step 1: Given P and test set T, find the execution trace of P for each test in T.

Step 2: Extract test vectors from the execution traces for each node in the CFG of P.

- **a test vector for node n, denoted by test(n), is the set of tests that traverse node n in the CFG**

Step 3: Construct syntax trees for each node in the CFGs of P and P’.

Step 4: Traverse the CFGs and determine the a subset of T appropriate for regression testing of P’ .

- **traverse the two CFGs from their respective START nodes using a recursive descent procedure**
- **if nodes N in CFG(P) and N’ in CFG(P’) are found to be syntactically different, all tests in test (N) are added to T’**

Dynamic Slice

Let L be a location in program P and v a variable used at L.

Let trace(t) be the execution trace of P when executed against test t.

The <u>dynamic slice</u> of P with respect to t and v, denoted as DS(t, v, L), is the set of statements in P that

- **lie in trace(t) and**
- **affected the value of v at L**

Test Selection Using Dynamic Slice

Let T be the test set used to test P.

- **P' is the modified program**
- **n_1, n_2, … n_k are the nodes in the CFG of P modified to obtain P'**

Which tests from T should be used to obtain a regression test T' for P'?

Find DS(t) for P. If any of the modified nodes is in DS(t), then add t to T'.

Test Minimization

Suppose that P contains two functions, main and f.

Suppose that P is tested using test cases t_1 and t_2.

During testing it was observed that

- **t_1 causes the execution of main but not of f**
- **t_2 does cause the execution of both main and f**

Now suppose that P’ is obtained from P by making some modification to f.

Which of the two test cases should be included in the regression test suite?

Obviously there is no need to execute P’ against t_1 as it does not cause the execution of f.

Thus the regression test suite consists of only t_2.

We used function coverage to minimize a test suite $\{t_1, t_2\}$ to obtain the regression test suite $\{t_2\}$.

Test minimization is based on the coverage of testable entities in P.

Testable entities include

- **program statements**
- **decisions**
- **def-use chains**
- **mutants**

Test Prioritization

Note that test minimization will likely discard test cases.

- **There is a small chance that if P' were executed against a discarded test case it would reveal an error in the modification made.**

When very high quality software is desired, it might not be wise to discard test cases as in test minimization.

- **in such cases one uses test prioritization**

Tests are prioritized based on some criteria.

- **For example, tests that cover the maximum number of a selected testable entity could be given the highest priority, the one with the next highest coverage m the next higher priority and so on.**

Summary

Regression testing is an essential phase of software product development.

In a situation where test resources are limited and deadlines are to be met, execution of all tests might not be feasible.

In such situations, one can make use of sophisticated techniques for selecting a subset of all tests and hence reduce the time for regression testing.

Test selection for regression testing can be done using any of the following methods:

- **only the modification traversing tests**
  - **based on CFGs**
- **tests using execution slices**
  - **based on execution traces**
- **tests using dynamic slices**
  - **based on execution traces and dynamic slices**
- **tests using code coverage**
  - **based on the coverage of testable entities**
- **tests using a combination of code coverage and human judgment**
  - **based on amount of the coverage of testable entities**

Questions and Answers