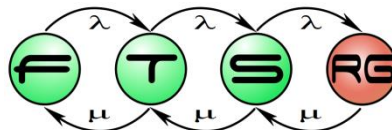


Regression testing

Zoltan Micskei

**Budapest University of Technology and Economics
Fault Tolerant Systems Research Group**



Main topics of the course

- Overview (1.5)
 - Introduction, V&V techniques
- Static techniques (1.5)
 - Specification, Verifying source code
- Dynamic techniques: Testing (7)
 - Testing overview, Test design techniques
 - Test generation, **Automation**
- System-level verification (3)
 - Verifying architecture, Dependability analysis
 - Runtime verification

Learning outcomes

- Recall approaches for regression testing (K1)
- Perform regression test selection on simple examples (K3)

Regression testing

“selective **retesting** 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”

Source: IEEE Standard 24765

Challenge with regression testing

- Default strategy: **retest-all**
- **Too many tests**
 - Does not fit into testing window (e.g. nightly build)
 - Retesting all after small change?
 - Slow feedback
- **Tradeoff**: precision vs. resources

Regression testing approaches

- **Test Suite Minimization**
 - Reduce test suite by eliminating redundant tests
- **Test Case Selection / Regression Test Selection**
 - For a given modification, find relevant tests
- **Test Case Prioritization**
 - Reorder tests to maximize some property (e.g. rate of fault detection, speed)

S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," STVR, 22:2, pp. 67–120, 2012.

Main idea of regression testing approaches

- Elements:

- P, P': program versions
- T: test suite
- C: coverage items (requirements, code structure...)

- Mapping:

- $T \rightarrow C$

- Use this mapping to calculate new test suite

Why is this hard?

- Test Suite Minimization
 - minimal hitting set problem (NP-complete)
- Test Case Selection
 - minimal set cover problem (NP-complete)
- Use heuristics in practice

REGRESSION TEST SELECTION

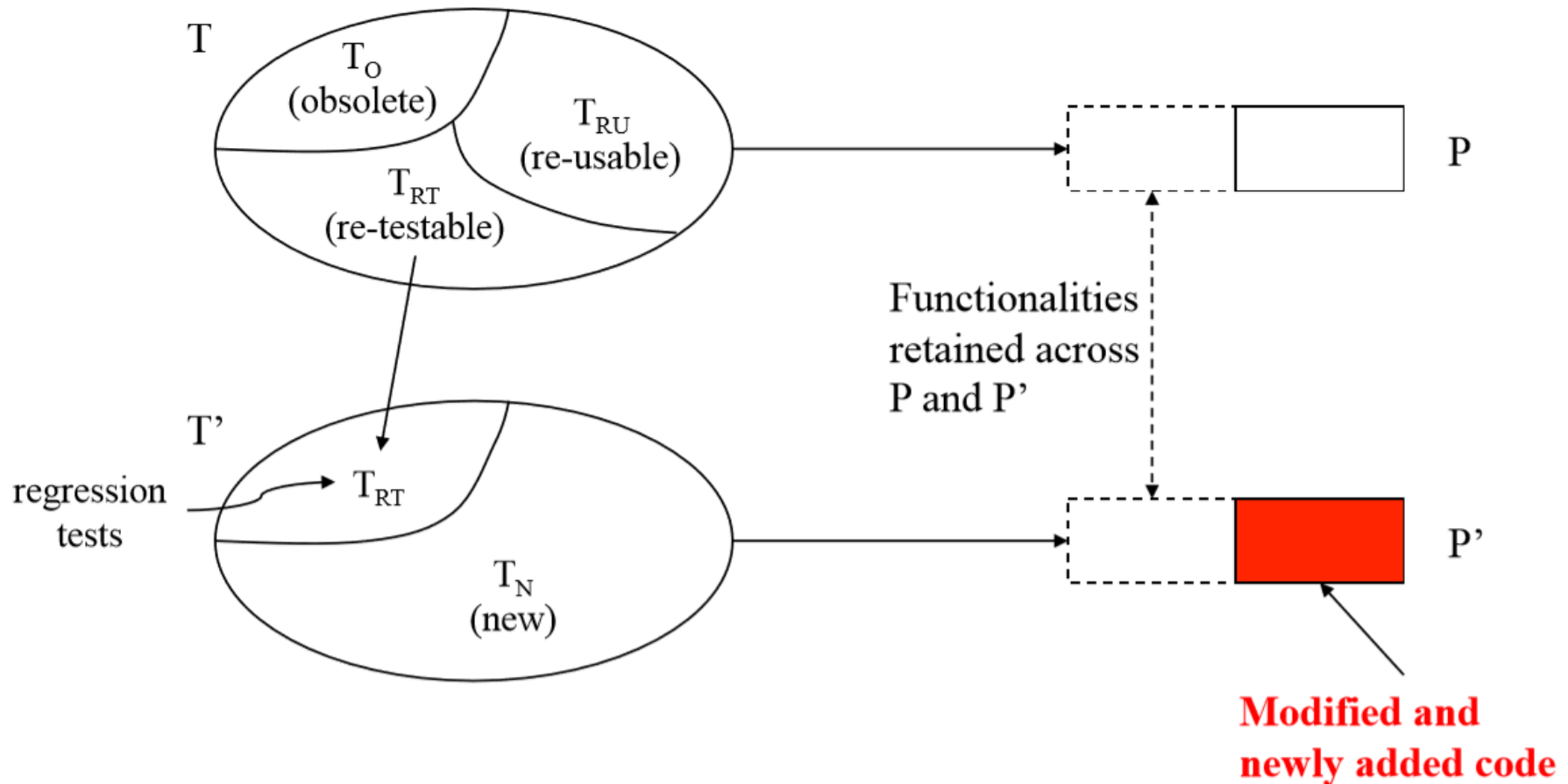
Example

- Tests: t1, t2, t3, t4
- Requirements: r1, r2, r3
- Mapping:
 - t1 -> r1
 - t2 -> r1, r2
 - t3 -> r2
 - t4 -> r3
- Change: r2 and r3
- Which tests to execute?

Classification of tests

- **Re-usable**: tests that exercise unmodified parts of the system.
- **Re-testable**: tests that are changed or are able to cover changed parts in the system.
- **Obsolete**: tests that cannot be used anymore due to changed specification or system structure.
- **New structure** tests that contribute to the overall coverage of the current, new system structure.
- **New specification** tests that verify new elements in the current specification.

Classification of tests



Source: Lionel Briand. An Introduction to Regression Testing, Simula Research Lab. 2011

Scope and granularity of RTS

What coverage items to use?

- Requirements (traceability is needed)
- Structural: code coverage
 - Component / File / Class / Method / Line ...
 - Precision vs. speed of RTS

Approaches for RTS

- Integer programming
- Data-flow analysis
- Dynamic slicing
- Textual difference (e.g. diff)
- ...

Simple greedy algorithm

- **Essential test case:** if a requirement is tested by only one test case
- **Algorithm:**
 - Select first the essential test cases
 - While there are uncovered requirements
 - Select the test case that satisfies the maximum number of unsatisfied test requirements

Exercise

We have a web shop application with a JavaScript front-end, a cloud-hosted back-end consisting of the inventory management and product recommendation modules, and a database. Unit tests test independently the inventory (UT1) and recommendation modules (UT2). Integration tests use the API of the back-end. The first integration test (IT1) checks a basic purchase, the second one (IT2) checks buying a recommended product. The system test (ST1) is a test driving the web UI.

- Our team performed a modification on the recommendation module.
- Give the mappings from tests to tested components.
- Classify each test with respect to the given modification.

Tool support: Visual Studio

Test Assemblies `**\*test*.dll;-:**\obj\**`  Version `2.* (preview)` 

Test selection

Select tests using `Test assemblies` 

Test assemblies `**\*test*.dll`
`!**\obj\**` 

Search folder `$(System.DefaultWorkingDirectory)` 

Test filter criteria 

Run only impacted tests ☒  

Number of builds after which all tests should be run `50` 

Test mix contains UI tests ☐ 

Execution options

Select test platform using ☒ Version ☐ Specific location

Test platform version `Visual Studio 2017` 

Settings file  

Microsoft DevOps Blog: Accelerated Continuous Testing with Test Impact Analysis

<https://blogs.msdn.microsoft.com/devops/2017/03/02/accelerated-continuous-testing-with-test-impact-analysis-part-1/>

Case study: Cisco Systems

- Videoconferencing software
 - Features: video call, multi-site call...
- Test suite: more than 5000 tests
- Test case: 30 min. manual preparation for each
 - → retest-all 100 days vs. couple of days testing budget



A. Gotlieb et al. Automated Regression Testing Using Constraint Programming, AAAI, pp. 4010-4015, 2016.

Summary

- Retest-all is not practical in many situations
- Regression testing approaches
 - Test Suite Minimization
 - Test Case Selection
 - Test Case Priorization
- Many heuristics and tools