November 30, 2016

Accurate Regression Test Cases Generation in Live Build Routines

Urs Ganesh Raj
Hewlett Packard Enterprise

Kalapriya Kannan
Hewlett Packard Enterprise

Accurate Regression Test Cases Generation in Live Build Routines

Abstract

Regression testing is typically performed after changing the code of a software program or system. Identifying the relevant regression tests to be performed to test all of the code changes is difficult, and usually a manual process. A technique is disclosed that automatically identifies and selects an optimal set of test cases to cover the impact of changes in the code base.

Description

This disclosure relates to the field of software testing.

Regression test optimization includes the two important related problems of selecting a subset of test cases that give maximum cost-value benefit, and ordering test cases such that early attainment of this cost-value tradeoff is achieved. However, due to the growth of the code base and the growing number of test cases, the intelligence is largely driven by human knowledge. Selection of the test cases is a difficult problem which is currently best decided by human knowledge. Typically many test cases are executed based on prior knowledge of the expected impact of the code changes. This results in side effects, such as a less than optimal selection of test cases to demonstrate the correctness of the code changes, or expending more testing time than the actual required time. Improvement requires that changes in the builds dictate the test cases which get executed, rather than manually associating test cases to changes.

According to the present disclosure, and as understood with reference to the Figure, identification and selection of the best set of test cases to cover the impact of changes in the code base is improved through a methodology referred to as SenSei. SenSei automatically associates "changes observed between different versions" of the code to the necessary test plans. These associations provide SenSei the ability to identify and list the optimal set of test cases required to perform regression. SenSei compares two builds and captures the changed modules between these builds. The changes are further used to mine the repository of test cases, which are tagged with module names. These module names associated with the test cases specify the modules in the code that the test cases touch upon. Sensei compares the change modules in the code to the modules in the test cases, and identifies all test cases that are absolutely required to be executed.

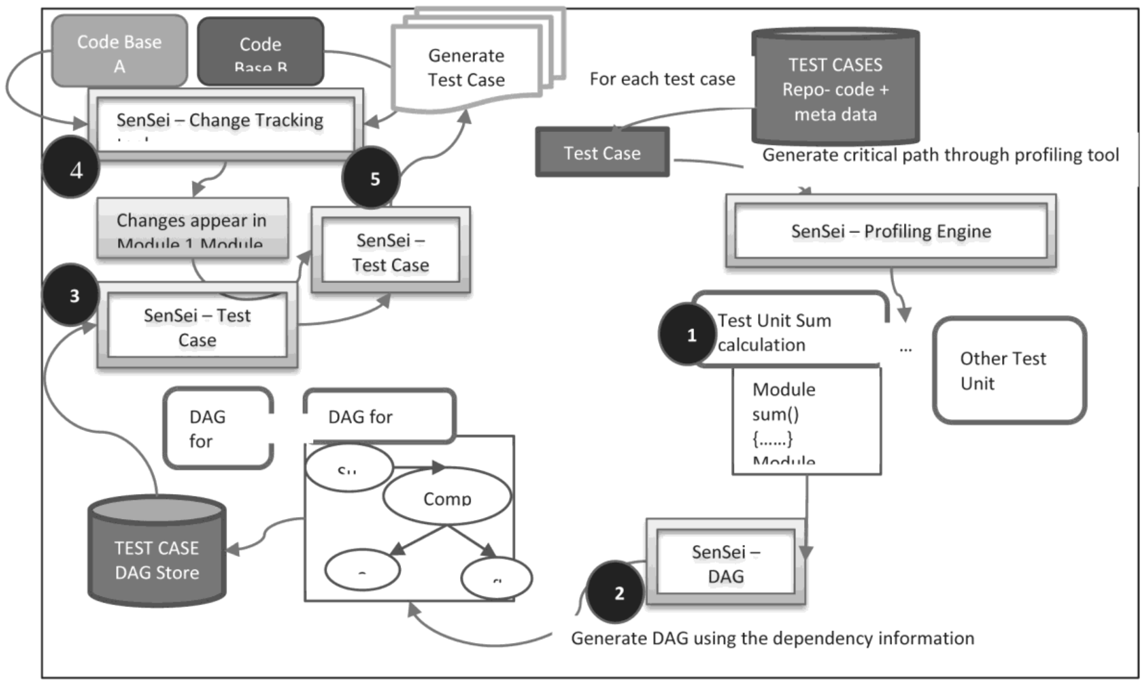
SenSei has two components. A first component analyzes two builds (sets of code files) and generates the changes between the two builds in terms of the modules (module name) which are changed between builds. A second component performs the trace/profiling of the test cases during execution, and builds the directed acyclic graph (DAG) of the critical path taken by the test cases.

SenSei stores the test cases tagged along with the DAG. SenSei provides an interface that allows searching the DAGs for the module names. With the names of modules that are changed, a lookup is performed into the DAG Store. All matching DAGs nodes are identified (that are associated with the module name in the DAG to the string given in the search). All test cases to which the DAGs are associated where the matching is observed is listed as regression candidates.

SenSei further optimizes the DAG store to perform a merged view of all common nodes of different DAGs of test cases. This results in listing the set of those test cases which are common for the same path, and picks up test cases that have different paths resulting complete coverage.

SenSei employs a multi-stage process. The SenSei Profiling Engine 1 considers each test case (code + meta data describing the test case) stored in repositories and uses a profiling tool (such as dtrace) to generate the critical path of the test case. The trace is typically a text file with several details of information like the input and the execution path within the modules. We limit our granularity to the level of modules. The SenSei DAG Generation Engine 2 consumes the trace file and provides a DAG which is the critical path of the test case and is stored a graph store in the Test Case DAG Store. The SenSei Test Case Optimization Engine 3 performs sub-graph isomorphism to identify overlapping portions of the graph and picks up the test cases that cover unique paths. The SenSei Change Tracking module 4 takes the sources and versions that are to be compared, and automates obtaining the information from the change versioning, and identifies the module names that are changed. Thus, a set of test cases along with the DAG (set of nodes representing modules and dependencies) are captured.

By identifying regression test cases according to the above process, the disclosed technique advantageously identifies the most accurate set of regression tests due to the changes in the code base. The time needed to identify the correct set of regression test is essentially reduced to the time it takes to execute the code. It provides 100% coverage and accuracy for quality assurance purposes. It can directly interface with different build repositories and is automated, requiring no human intervention to derive the set of regression test cases by considering all dependencies. It is simple to configure, and requires no maintenance once configured.


Code Base A
Code Base B
Generate Test Case
For each test case
TEST CASES Repo- code + meta data
SenSei – Change Tracking
4
5
Test Case
Generate critical path through profiling tool
Changes appear in Module 1 Module
SenSei – Test Case
SenSei – Profiling Engine
3
SenSei – Test Case
1
Test Unit Sum calculation
...
Other Test Unit
Module sum() {......} Module
DAG for
DAG for
Comp
TEST CASE DAG Store
SenSei – DAG
2
Generate DAG using the dependency information

Disclosed by Urs Ganesh Raj and Kalapriya Kannan, Hewlett Packard Enterprise