NTR - New Test Runner, a Test Regression Runner and Analyzer

Nitin Rastogi
Staff Program Analyst
Cadence Design Systems, Inc

Mona Jain
Sr Staff Program Analyst
Cadence Design Systems, Inc

Saran Prasad
IT Director
Cadence Design Systems, Inc

ABSTRACT
In this paper, we describe the New Test Runner, a test regression and analyzer for Improving Testing capabilities and its implementation in form of a working system across various BU's (Business Units) in Cadence.

1. INTRODUCTION
In the prevailing competitive environment, companies are facing tremendous market pressures to launch defect-free products in a timely manner. Cadence has an objective of reducing the number of CCRs/Defects filled by customers to 50% by year 2010. One of the ways to achieve this is by testing the products in an effective and efficient manner and catch/fix the bugs before they are reported by customers.

Experience has shown that as software is developed, reemergence of faults is quite common. Sometimes it occurs because a fix gets lost through poor revision control practices (or simple human error in revision control), but often a fix for a problem will be "fragile" i.e. it fixes the problem in the narrow case where it was first observed but not in more general cases which may arise over the lifetime of the software. Finally, it has often been the case that when some feature is redesigned, the same mistakes will be made in the redesign that were made in the original implementation of the feature.

Therefore, in most software development situations it is considered a good practice that when a bug is located and fixed, a test that exposes the bug is recorded and regularly retested after subsequent changes to the program. Although this may be done through manual testing procedures or using programming techniques, it is often done using automated testing tools.

Regression tools were introduced in order to provide a solution for maintaining version quality along the life cycle of a product. However, in most cases they suffer from insufficient ability to configure tests and reuse them, huge maintenance overhead, failure to supply sufficient debug aids, and above all limited results analysis capabilities. NTR has Failure Analysis capabilities and thus is an innovative environment for regression tools results analysis.

NTR is based on the Incisive Enterprise Manager platform. This paper focuses on presenting NTR major capabilities: Advanced Failure Analysis, Reports Generation and Debug Support.

For the past few years different products have been tested using NTR regression tool which has optimized their development process and spared the tedious work of regression tests results analysis.

2. PROBLEM STATEMENT
During the life cycle of a product, the regression tests and harness tools used to validate the product evolve from fairly simple components to complex and hard-to-manage ones. As the validation of a product generally consumes many resources, any improvement in the regression testing infrastructure tends to pay off quickly.

Until now, solutions were confined to the level of implementation and execution. These solutions were usually based on predefined templates or scripts that implement new tests. In most cases, they were successful, because 99% of the tests in a regression suite share very few execution flows. No attempt was made to address challenges beyond execution. In most cases, failure analysis ended with a summary of failed tests per group. Configurability typically focused solely on the environment setup. Test reuse was barely considered. Maintainability and extensibility of tests was somehow addressed by having common templates and scripts, but there was no way to track or enforce correct usage.

The current systems/tools that are being used internally lack exhaustive failure analysis capabilities of regression results. This creates a situation for the engineers where they end up fixing the same issue in the multiple runs. There is no way to categorize the failures which will help the engineers to identify the cause in good time and apply the fixes quickly. In the current environment there is no automatic way of comparing the results of the multiple sessions which are very important when benchmarking a regression or checking what has changed talking of test results.

A good regression suite should also have some way to help the engineers to debug the failures and fix them with ease. The difficult part in debugging the failures is to generate the execution environment and validating all the stages of the run.

Different groups follow a different approach of executing their testcases and their verification mechanisms. There is no standard procedure of writing a flow of events in the regression process.

Other than all the important features provided by any regression tool, standard/customized reports should be generated such that it helps different audience. Sometimes the need of different kind of reports differs from person to person. For example, managers, validation engineers, developers might require different data from the result of regression.

In a company like Cadence where code is protected under different version control systems, it is highly desirable to have systems which can supports and integrates smoothly with version control systems such as Clearcase and CVS. The volume

of testcases running under regressions are sometimes so large that it burdens the resource on which they are executing. In such scenarios distributed computing is a way to make maximum and effective utilization of the existing resources. This will also reduce the total execution time of the regression.

In Cadence environment wide variety of verification products and tools are used to test a product under testing, hence the regression environment should be such that it supports the needed tools. It is always good to have regression tools which require less effort in managing the testcases and their setup. There is a need to find a quick way of running testcase regressions and it can be achieved by reusing the old test flows.

Another issue with most of the standard testing tools used in Cadence is that once the testing is done and failures are known there is no way to record information about the failures or assign ownerships.

In our later sections we will talk about how to deal with the problems stated above.

3. OVERVIEW

New Test Runner (NTR) has been developed internally, and is based on the Incisive Enterprise Manager platform. NTR addresses the challenge of making regression testing simple, repeatable, automated and most important easy to manage. NTR encapsulates hierarchical test case management, multi-user support, automated test execution and results analysis to facilitate easy testing.

NTR categorizes regression results in groups. These groups classify failures according to their cause instead of providing each failure out of context. The major advantage of this approach is eliminating redundant work on common failures. Most regression tools supply only a pass / fail result for each test. However, NTR generates a detailed report on each test failure cause. NTR FC approach supplies a framework for clustering multiple test results with a common failure cause. A cluster can be viewed as a group of tests with the same failure cause. NTR may filter some of the information about the failure cause so that the common aspects can be easily identified. The importance of clustering different tests with the same failure cause is to identify a single problem, which most likely can be fixed with a single bug fix.

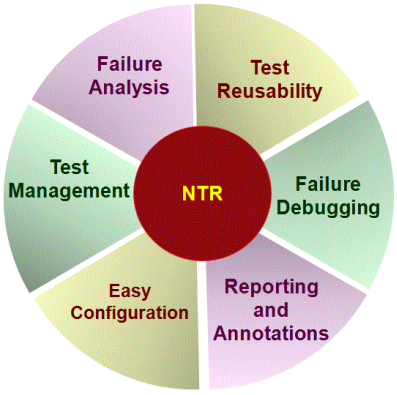

Figure 1. Features of NTR.

The architecture of NTR is such that it supports the existing LSF deployment matrix of Cadence. It can be easily integrated with different LSF configurations for different clusters. This setup enables regressions to run on distributed environment and also takes care of job scheduling overheads.

NTR also supplies traditional tests management features. The most important one is version quality comparison mechanism for identifying degradations. As a regression tool NTR provides a way to analyze if a problem is new or exists in previous versions. In order to supply this functionality, NTR has a compare mechanism that produces a compare status for each problem.

When NTR identifies a new problem, the regression manager needs to classify it. Classification can be done by attaching information for the problem. As part of classification, the regression manager identifies the owner of the failure and assigns the problem to him. In order to execute this task, NTR provides two solutions.

1 The notes mechanism contains the ability to easily attach information to problems after they occur.

2 The failure category mechanism lets user define a problem, and each test that matches the failure category, is attached with all the available information for this category. The failure categories are transferred from regression to regression. This feature optimizes the time spent by regression manager in analyzing failures. A failure, which is attached to an existing category, is considered as a known failure.

NTR also helps the user to isolate the simplest test from the problem cluster in order to debug the problem. In addition, the NTR supplies tools for running the failed tests under the user environment with the test debugger utility.

After finishing a regression run, the NTR produces a report specifying all the problems encountered in this run, in addition to available information and compare status of each identified problem. The report mechanism supplied wide functionality in order to achieve maximum flexibility and meet the needs of various users.

4. IMPLEMENTATION DETAILS

NTR is a layer over vManager and uses it for execution of the testcases. The input to NTR is the VSIF file that contains the attributes and the test definitions. Once the vSif file is read, NTR creates a directory for each test in its own workspace and copies necessary files to it. LSF is used for distributing the jobs to the server farm machines. Results from the testcase execution are sent back to the workspace and parsed to generate a Verification Session Output File (vSof) file.

These vSofs are available as web reports.

4.1 Regression Definition

A regression definition consists of three major parts:

• Basic test definitions

• Configuration and environment setup

• Test selection

4.1.1 Basic Test Definitions

A pure test definition looks as follows:

```
//Test definition file: socfv.vsif

test socfv_flow
        {
                home: /grid/ilab/vol07/NTR/vsifs;
                 simulate_script:"run.csh";
        };
```

In this case the test definition contains the path of the testcase directory and the simulate script which executes the actual testcase. In a single verification session input file (vSif file), multiple tests with different properties can be defined. No configuration specification takes place at this level. Instantiating the test definitions in the vSif and completing the required configuration is done at a higher level of hierarchy.

4.1.2 Configuration and Environment Setup

Configuration covers a wide range of things. NTR gives flexibility to configure different properties of test environment.

Example:

```
//Configuration file: socv_flow.vsif

group SOCFV_KIT
   {
      home_root:/grid/ilab/vol07/NTR/vsifs;
      display: ANY_DISPLAY;
      scan_script: specman.flt;
      precious_files: *;
          test socfv_flow
         {
                 home: socvflow;
                  simulate_script:"run.csh";
         };
              };
```

In the above example, the socv_flow.vsif file contains environment setup that would be applicable for the entire group of testcases. The testcase execution will be based on these definitions. There can be several levels of configurations and the configurations related to tool version are likely to be specified higher in the hierarchy, as versions might change frequently.

4.1.3 Test Selection

A common requirement of regression testing environments is to enable flexible selection of tests from a test suite. The two main motivations for that are:

• Product restrictions – A typical example is restrictions on the support matrix of the products being validated. Typical matrices are OS and cross-product version support. Maintaining the support matrix might become demanding if it is not done in a central location and in a top-down manner.

• Feature-test correlation – Accurate correlation reduces the required computing resources. Shortening the turnaround time from launch of regression session to final results provides a significant productivity increase.

It lets us define the requested subset of tests in a way that governs the regression definition while avoiding specifications per tests or group of tests. The knowledge of test definitions enables accurate and straightforward test selection by evaluating conditions in terms of properties. The following example reflects the lack of support for the 64 bit platform:

```
session platforms {

refine: not ($attribute(bits) == 64);

};
```

NOTE: Test selection is done at the regression session level, which is orthogonal to the test and configuration definitions. The NTR ability to support a fine granularity of test selection enables three layers of test selection, each layer adding its own restrictions:

1. Project

2. Feature

3. Developer

At the project level, selections are mainly driven by the support matrix. At the feature level, selections focus on tests that exercise a specific feature. At the developer level, selection depends on current development requirements.

4.2 Failure Analysis

Failure analysis is an iterative effort, consuming significant human resources. Good and well-defined procedures are required to perform failure analysis efficiently.

Fundamentally, failure analysis should focus on problems and not on failed tests. This is true from both management and development points of view. The number of problems and their severity uncovered by a regression suite is much more significant than the number of failed tests. To facilitate better analysis, runtime information (like the OS version used to execute the test and failure descriptions) should be gathered and added to the set of test properties. Having a comprehensive set of runtime and behavior properties leverages the analysis capabilities of tools such as vManager.

4.2.1 Classing Failures

Classing failures supports a focus on problems rather than failed tests. A failure class definition is a problem definition that serves the need to characterize a failure and apply it to the entire regression suite. Defining a failure class involves:

• Defining the signature of a problem, possibly using the failure description, and even the conditions that cause it.

• Assigning relevant information to the problem, like owner, status, etc.

A failure class is like an empty bucket waiting to be filled with failed tests that meet the bucket terms. As such, failure class definition is useful for random driven simulations and sporadic failures, where the original test definitions cannot be recognized upfront as failures.

With NTR, developers can define meaningful and precise failure classes. For example, you could have a failure class for a specific simulator version running in a specific mode as follows:

```
fc warm_restore {
condition:
    ($attribute(simulator) == ncsim) and
    ($attribute(link_mode) == static) and
    ($attribute(os) == solaris) and
    ($attribute(bits) == 32) and
    ($attribute(failure_description) == "INTERNAL ERROR");
failure_comment: Occurs in IUS5.8 earlier versions;
failure_defect: 910976; //PCR number
failure_status: defect;
};
```

NTR uses its knowledge of test properties to help developers define the failure class condition. NTR automatically detects the set of attributes that share the same value in a group of failed tests.

The benefits of classing failures are many and varied. Classing failures can be used to control regression execution, for example, by rerunning particular failure classes or by performing automatic problem definition before analysis takes place. Those and other aspects should to be covered by other papers.

4.2.2 Annotating Failures

In some cases, defining a problem by classing failures is not needed. This could happen when a problem is very unique or temporary. Nevertheless, it might still be helpful to annotate such a failure, as done using NTR. NTR not only allows annotation of tests but also the propagation of test annotations (as most annotations must be propagated from regression to regression in order to maintain the annotations). The question is how to identify corresponding tests in the results of two different regressions. In other words, what defines a test instance? The answer is not trivial because the same test with the same name can be used by different configurations (see Section 4.4 "Top-Down Configurability and Test Reusability"). To accommodate that, the properties that define a test are identified, and finding corresponding tests comes down to identifying tests with the same values for properties.

4.2.3 Comparing Regression Results

The ability to find corresponding tests in different regressions also enables comparison of two regressions and identification of new failures, new passed tests, and so on. In a regression suite with failures, this is a powerful tool. It can serve as a Go-NoGo gauge. Having no new failures in a regression generally indicates high quality of the product under development. The same technique may also be used to compare performance results, CPU, or memory.

4.3 Test Banks

It is a fairly common practice to split a regression suite into distinct groups, or "banks", of tests, all sharing something in common. Usually, test banks validate different aspects of a product. This practice is useful as it captures the essence of the tests. NTR supports this methodology in two ways:

• Hierarchical structure of tests with meaningful group names – NTR also enables viewing of tests in an expandable tree structure, with zooming in and out, rather than a flat listing of groups.

• Specifying the scope of tests and group of tests – For example, the following group of tests has a verilog scope:

```
group ports {
scope: verilog;
#include verilog_ports.vsif
};
```

One test can have multiple scopes, which can be useful for various purposes, like test selection. Despite the advantage of having test banks, there are two fundamental problems with it:

• Test banks are black boxes that do not provide knowledge about internal tests. Hence using them negates most of what we presented in this paper.

• Test banks are defined statically by the regression suite moderator. However, in most cases, tests have multiple aspects and should be used in various contexts.

4.4 Top-Down Configurability & Reusability

The hierarchical structure of regression definition is used to implement top-down configurability and test reusability. To deliver a comprehensive solution, three features are required:

• Selecting Tests at the Group Level

• Forward-Referencing to Properties

• Conditional Value Assignment

4.4.1 Selecting Tests at the Group Level

The idea is to apply a configuration to a subset of tests. Those tests are typically instantiated by multiple configurations. Each configuration might apply to different set of tests within a group.

4.4.2 *Forward-Referencing to Properties*

A top-down configuration might require the ability to govern property values from the top level. However, in some cases a property value depends on a lower-level test definition. Forward referencing addresses that. The following example demonstrates a forward-reference to a test_name property used to define the log_file property of IFV.

```
group IFV {

product: ifv;

log_file: $attribute(test_name)_ifv.log;

#include top.vsif;

};
```

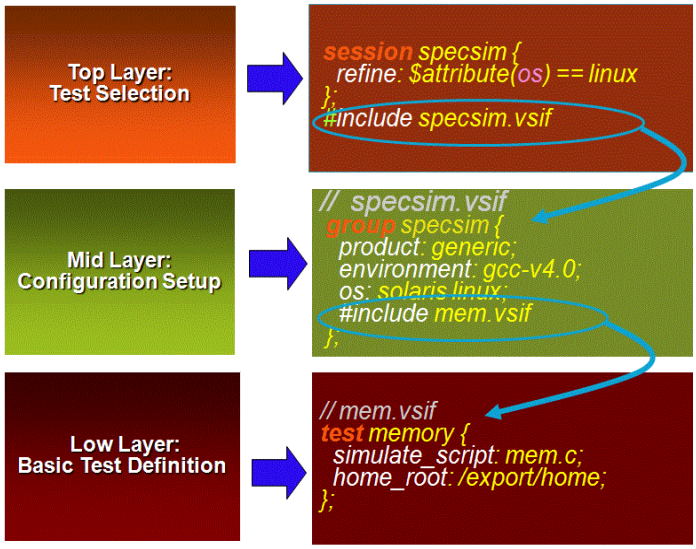

Figure 2. Top-Down configurability and reusability.

4.4.3 *Conditional Value Assignment*

Conditional value assignment is the opposite requirement to forward-referencing, which is, defining a test property according to a top configuration. The following example shows how the hdl_compile_args property is conditionally assigned according to

the value of the simulator property. The simulator is defined at a higher level of hierarchy.

```
test memory {

hdl_compile_args: $condition($attribute(simulator)==ncsim);

hdl_files: mem.v;

};
```

4.5 Test Suite Management

Managing a regression suite can be difficult, especially when no documentation is available and the size of the regression suite is large. Using an analysis tool that reads the regression definition (vSif), anyone can analyze the content of the regression suite without prior knowledge about the structure of the regression suite. Once the regression definition is read, queries can be used to understand the content of the regression suite.

4.6 Test Maintainability and Extensibility

The fact that a test definition is kept solely at the declarative level dramatically reduces the required maintenance effort. Changing an execution flow or adding a new property or property value barely influences existing declarations. In addition, the logic that translates property values into execution is centralized and can be adjusted to make changes that affect all tests. Moreover, products from the same family might share common execution logic implementation. Therefore, a single change can apply to all tests for multiple products.

4.7 Debug Abilities

NTR as a concept can be used to implement various debugging aids. So far we have identified the following possibilities:

• Identification of the simplest test that represents a problem – The simplest test is chosen according to its properties. A limited number of files and tools, short execution duration, and other properties contribute to test simplicity.

• Rerunning failed tests

• Interactive or verbose reruns – The fact that the lower level execution logic is decoupled from the test definition makes it easy to support debug flows that are general enough to address all product tests.

• Using various versions of tools and comparing previous results with current results

4.8 Easy Definition of Tests

The process of adding new tests to regression suites also benefits from NTR. Developers focus on test properties. They do not need to know much about test execution. Developers also do not need to take into consideration reuse of tests (with the exception of any need for conditional value assignment) Moreover, developers have a compiler that constantly checks the test definition, producing informative errors and warnings and thereby making the NTR a user-friendly environment for test definition.

5. CONCLUSION

NTR tool provides an edge to software validation teams through its capability of failure analysis and classification. It also gives testcase comparison in multiple regressions and provides debug utility to analyze, fix and rerun the failures.

6. DEFINITIONS, ABBREVIATION AND ACRONYMS

Acronym	Description
NTR	New Test Runner
LSF	Load Sharing Facility
FC	Failure Classification
VSIF	Verification Session Input File
VSOF	Verification Session Output File

7. ACKNOWLEDGEMENTS

Name	Description
Yair Nergaon	Concept Owner

8. REFERENCES

[1] Automated verification management with extended language and simulation support. Available link: http://www.cadence.com/products/fv/enterprise_manager/pages/default.aspx

[2] gridMatrix
JRL (Job Request Language) available link: http://www.isi.edu/~deelman/wfm-rg/pulsipher_06_04.ppt

9. BIOGRAPHY OF THE AUTHORS

Saran Prasad

Saran is working as IT Director at Cadence, since 2002. He is postgraduate in Electronics an MBA and alumni of IIMC. He has industry experience of over 19 years. Saran is currently perusing his PhD in software testing domain.

Nitin Rastogi

Nitin is working as Staff Program Analyst in Cadence Design Systems, Inc. since 2001. He is graduate in computer science and a member of IIMB alumni and leading the test automation team in the present company. Nitin has submitted papers in the Quality conferences inside Cadence as well as externally and is helping the Product Validation team for their test automation needs.

Mona Jain

Mona has around 13 years of industry experience and is currently playing the role of a Technical Manager in the Engineering Tools team in Cadence. Previous to Cadence she was working in the role of J2EE Technical Architect and Development manager. Mona has also worked in United States for 6 years with various large corporate in both Microsoft as well as J2EE technologies. Mona Jain completed her MCA from Pune University in 1995