Continuous Change-Driven Build Verification

Marc Hornbeek, Spirent Communications
Manager, SCM Systems
marc.hornbeek@spirent.com

ABSTRACT

This white paper describes the "change-driven" method, associated tools and experiences for automated, intelligent software build verification. The method and tools involve integrating code change management such as Perforce with software build systems, and automated build verification systems. A weighted Code-to-Test (CODTEF) attribute table correlates code changes to tests. Code changes automatically determine test selection and test results are automatically targeted back to code faults. The resulting closed-loop continuous build/test environment is vastly more efficient than traditional regression approaches which bottleneck many continuous change and Agile environments. It is shown how the change driven method enables higher test coverage during continuously varying build schedules and priorities.

INTRODUCTION

At Spirent we extract source code change information from our Perforce software change management database for each software build and use a Change-Driven Test methodology, abbreviated ChDT for quick reference, to improve the effectiveness of the software code-build-test-fix cycle. This Change-Driven methodology helps get maximum value out of a given pool of automated tests cases for a given amount of time and available test resources. It solves the test scaling problem with Agile development for large scale software systems.

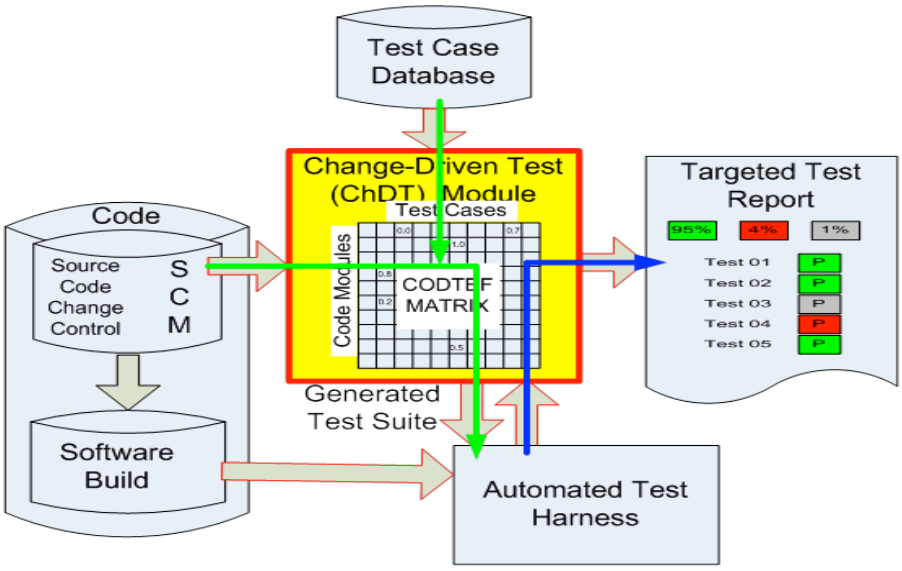
PERFORCE ENVIRONMENT AT SPIRENT

Spirent has a very large 1.6 TB Perforce database serving 288 users, 5 development sites, 36Million Lines of code, 72 code branches, 132 code groups. It operates at a 99% reliability level. Code changes through Perforce are used to generate 52 development builds per day at 95% success rate. Builds are verified using Nine Build Verification (BV) systems with 493 functional test cases executed per build, executed in parallel in 4 hours.

CHANGE-DRIVEN SYSTEM CONCEPT

As show in the following diagram, the Change-Driven Testing methodology introduces a ChDT system between the different components of the Code/Test Loop. At the heart of the ChDT system is a matrix of Code-to-Test correlation factors called the CODTEF matrix. The CODTEF values dynamically adjust according to prior test results. The CODTEF values are used by the ChDT system to select test cases that correlate most strongly the specific code changes that

are derived from Perforce for each software build since the last test run. The resulting generated test suite is fed to the Automated Test Harness. The Test Harness reports the test results back the ChDT system which uses a reverse traversal of the CODTEF matrix to determine which test results are most relevant and generates user-specific reports. The ChDT system updates the CODTEF matrix using the most recent test results, thereby completing the loop.



HOW TO APPLY CHANGE DRIVEN TESTING

Implementing ChDT requires careful, meticulous work and is worth it. When completed it will realize the following benefits:

- Resource utilization more adaptable: time, and equipment variables
- Improved Quality earlier in the cycle: direct correlation to code, measurable coverage
- Fewer change/test/fix cycles: fewer low-yield tests, results to correct owners
- Faster test analysis: improved, more targeted measurements
- Supports advanced metrics: generates data for MTBF models
- Improved Maintenance: identifies error-prone code, high and low yield tests

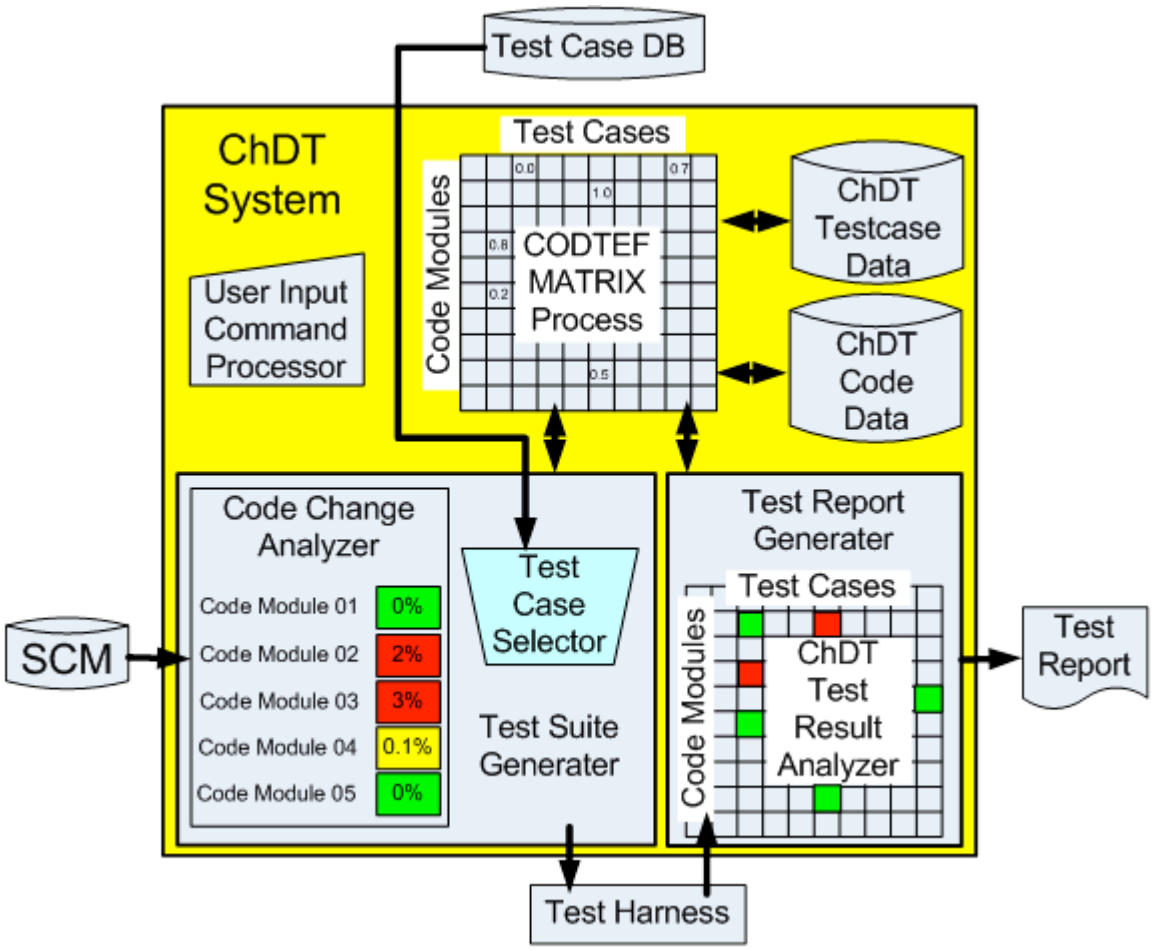
Change-Driven Test Automation may be applied to every test phase that would benefit from test efficiency gains including continuous build verification, feature testing, regression testing, milestone builds preparation and release builds preparation. However it should be apparent that, for ultimate safety, all tests should be selected for actual milestone builds and customer release builds.

PRE-REQUISITIES

To apply Change-Driven Testing, an organization needs to have in place some prerequisites as follows:

1. A source code management system such as Perforce is required before you can add any tools to track code changes, one of the key parameters of the CODTEF matrix.
2. An automated test framework is needed to manage the execution of tests.
3. It is necessary that the test creation and test management process for running test scripts will support tests that can be run independently of each other. If this is not available then the scripts will need to be restructured to realize this.
   Note: Organizations that do not have a strong test script standard in place to ensure scripts are created to run independently of each other are referred to the TSCRIPT methodology listed in the references section. TSCRIPT is good test script methodology to follow with or without Change-Driven Testing. After all test automation systems are only as good as the test scripts that run on them.
4. Tools developers are necessary to make special software tools and changes to existing tools to implement the different components required for Change-Driven Testing.

CHDT ARCHITECTURE



This above diagram illustrates a generic ChDT architecture. The big yellow ChDT box in the middle may suggest a huge monolithic module is necessary, but the ChDT system can be implemented as a set of loosely coupled smaller applications implemented over several project phases.

The key elements of the architecture are:

1. A user interface command processor which processes system level input parameters and generates responses to the user.

2. A database for storing and retrieving the CODTEF matrix values, Testcase meta data, Code metadata, and resource data.

3. A code change analyzer tool which is a plug-in for the source code control system that generates code change data for each target software build cycle or range of builds.

4. A test selection tool that processes the user inputs and change information to create a specific test case list according to the test selection algorithms.

5. And a Test Report Generator tool which generates a variety of user specific reports and metrics.

User Interface (ChDT UI)

The form of the user interface is up to the implementer. It can be a web interface, command line or batch system for example.

ChDT UI shall accept the following input parameters which are required to control the ChDT system:

R : Resources available for test
Code Range: identifies the code changes to be tested.
T : Time available for running all of the tests
Q : Quality level = minimum **CODTEF** value used in selection
M : Must-do tests = {tx, ..}
TE : Test Exceeded for each selection level = **TE**[TE1,TE2,TE3]

Database (ChDT DB)

ChDT DB is database for storing and retrieving the **CODTEF** values, Testcase meta data, Code metadata, and resource data. In addition to acting as a repository for the above data values, the **ChDT DB** system maintains the **CODTEF** values according to the algorithm below:

After each test run ChDT adjusts the **CODTEF** values according the to the test result Inconclusive, FAIL or PASS as follows:

Inconclusive: Do not adjust **CODTEF**

FAIL: **New CODTEF = Prior CODTEF + dF(1- CODTEF), Maximum=1**
where dF = average test failure rate

PASS: **New CODTEF = Prior CODTEF - dP(1- CODTEF), Minimum=0**
where dP is the change rate of **CODTEF** for PASS that will match the change rate of FAIL if the system performs at the average level over time.
dP= dF(dF/(1- dF))

Gradual **CODTEF** adjustments prevents instabilities in test selections and reduces the possibility of eliminating important tests when FAIL/PASS trends temporarily stray significantly from average values. If a test Fails frequently, **CODTEF** will tend towards 1, which causes the test to be selected most of the time. This is a "high yield test". Either the code or the test is a candidate to be re-factored. If a test Passes for a long time, **CODTEF** will eventually tend towards 0, which causes the test to never be selected. This is a "low yield test" that can be pruned from the database. **CODTEF** = 1 can only be set manually, and once set, will not adjust unless changed manually or until the adjustments round up to 1.000 depending on the precision of **CODTEF** values.

Code Analyzer (ChDT CA)

The code analyzer tool is a plug-in for the source code control system that generates code change data for the **Code Range**. This will typically be a counter tool that counts the number of changes that occurred for each code module in the user-specified **Code Range**. For example if the ChDT system is applied to specific directories of the code version management system then **ChDT CA** will count the number of file changes which occurred for the specified **Code Range**. The number of changes per code module will be used by the **ChDT Test Selector** to determine which **CODTEF** values to use for a particular **Code Range.** It is up to the ChDT system implementer to decide which changes to count. Refer to section 4.0 **Data Model**.

Test Selector (ChDT TS)

The test selector tool processes the user inputs and change information to create a specific test case list according to the test selection algorithm I below:

Test Selection = TS1 +TS2 +TS3 +TS4 =
= Test Selection Level 1 – **M** / Must-do tests
- Always select, provided T is big enough and resources are available. If not then select the tests which have the highest failure rate, provided TE1 is not exceeded, otherwise abort the selection.

+ Test Selection Level 2 – Recently Failed tests
- Select tests which failed last run, provided **T** is big enough and resources are available. If not then select from this group tests which have the highest failure rate, provided TE2 is not exceeded, otherwise abort the selection.

+ Test Selection Level 3 – **Q** / Best **CODTEF** Value tests
- Select tests which have **CODTEF=Q** or higher, provided **T** is big enough and resources are available. If not then select from this group the tests which have the highest failure rate, provided TE3 is not exceeded, otherwise abort the selection.

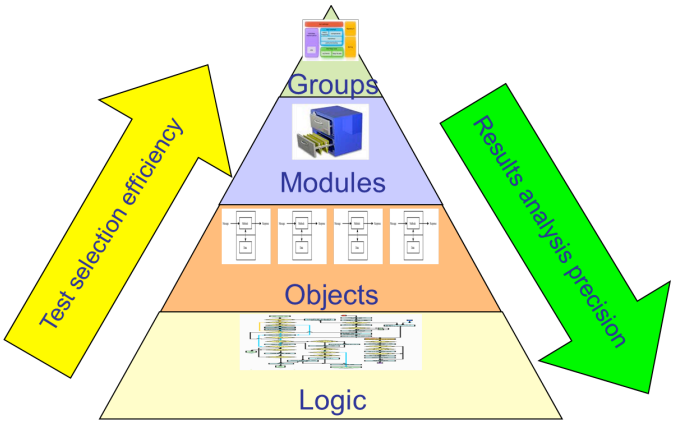
+ Test Selection Level 4 – Additional tests if time allows
  - If time remains select tests which have the highest **CODTEF**<**Q**, until the total time to test = **T**.

Report Generator (ChDT RG)

The Report Generator **ChDT RG** tool generates user specific reports and metrics for specific classes of users. This eliminates wasteful redundant reports of test results to irrelevant owners:
Code Owners Report: Test results for code modules that they own, where the **CODTEF** value is **Q** or higher
QA Managers Report: Summary of failed test results and Excluded tests for risk assessment
Testers Report: Detailed test results for failure analysis

DATA MODEL (CHDT MODEL)

The diagram below shows there are trade-offs to consider when choosing the data framework for a given Change-Driven system implementation. Very fine-grained Change-Driven system designs that use code-logic-to-test relationships will have the advantage of resolving to a more precise view of results during the analysis phase, however the test selection process will be more time consuming because code logic analysis will naturally take longer than analysis at a higher level, and the system would be language dependent. On the other end, coarse Change-Driven system designs that use coarser code-groups-to-test relationships will have the advantage of rapid test selection, but the precision of results will be limited to the code group. A middle layer, such as using modules or small groups of modules, provides a better balance for a many organizations.



SUGGESTED DEVELOPMENT PHASES

The following project development phases are suggested for implementing ChDT. Start with applying ChDT to large groups of code/tests and then progress to smaller sub-groups and finally advanced reports and tools.

Phase 1 – Group Level

Analysis and restructure: Break up current fixed test suite into a small set of groups. Identify code group – test group relationships.
Tools: Automatically identify group code changes, Launch specific test groups, Report results according to group owners

Phase 2 – Sub-Group Level

Design: Identify code subgroups and initial test correlations. Identify and design specific tools changes.
Tools: Implement remaining tools not implemented in Phase 1, except the advanced metrics

Phase 3 – Advanced Reports and Metrics

Analysis and design: Determine which advanced metrics will be used by the organization such as Reliability Growth Modeling.
Tools: Create metric tools using the data collected during the Change-Driven loops.

MAINTENANCE

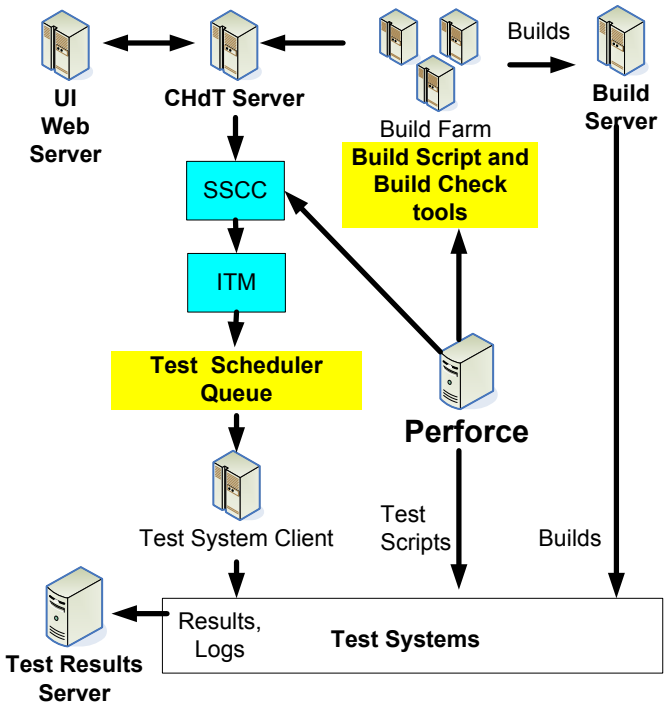
After the first phase of ChDT system is in place:

- Set initial **CODTEF** factors for new code and tests (or leave blank)
- Adjust the default **CODTEF** values for Blanks
- Adjust **dF** and **dP** factors according to the long term average of test failures.
- Prune or refactor low yield tests
- Re-factor high yield tests

SPIRENT EXPERIENCE WITH CHANGE DRIVEN TEST

The diagram below illustrates how the Change-Driven system has been implemented within Spirent. In the Spirent implementation a Smart Source Code Count module (SSCC) processes two inputs files. One is the SSCC Branch Range file which indicates the begin and end labels for source analysis. The other is the SSCC Source Code Filter file which indicates which files in Perforce are to be analyzed. Upon completion of a build, the build script automatically launches the SSCC module on the CHdT server using the specific input files, scans all the source code changes in Perforce within the given branch range and produces the SSCC Results file that includes a count of all file changes that match the Filter file. This process is down for each of the nine Build Verification systems to determine if the build ranges contains changes specific to each BV system. An Intelligent Test Module (ITM) then used the SSCC result to determine whether or not to schedule each BV system for a particular build or not. In this way BV systems

are only scheduled when there are relevant code changes, resulting in a major savings in BV utilization and improved coverage of builds with less redundant BV results.



EXAMPLE SSCC INPUT FILE – BRANCH RANGE

```
<?xml version="1.0" ?>
- <Branch name="mainline">
  <StartBuild number="3.60.7740" />
  <EndBuild number="3.60.7850" />
</Branch>
```

EXAMPLE SSCC CODE FILTER FILE

```
<?xml version="1.0" ?>
- <FileGroups>
- <FileGroup name="IPS" owner="Marc Hornbeek">
  <FilePath name="PortConfigs"
    path="//TestCenter/mainline/content/traffictests/benchmarking/bll/src/PortConfigs" />
  <FilePath name="Rfc2544Back2Back"
    path="//TestCenter/mainline/content/traffictests/benchmarking/bll/src/Rfc2544Back2Back" />
  <FilePath name="Rfc2544Common"
    path="//TestCenter/mainline/content/traffictests/benchmarking/bll/src/Rfc2544Common" />
  <FilePath name="Rfc2544FrameLoss"
    path="//TestCenter/mainline/content/traffictests/benchmarking/bll/src/Rfc2544FrameLoss" />
  <FilePath name="Rfc2544Latency"
    path="//TestCenter/mainline/content/traffictests/benchmarking/bll/src/Rfc2544Latency" />
  <FilePath name="Rfc2544Throughput"
    path="//TestCenter/mainline/content/traffictests/benchmarking/bll/src/Rfc2544Throughput" />
  </FileGroup>

- <FileGroup name="Routing" owner="Owner Name">
  <FilePath name="bfd" path="//TestCenter/mainline/content/routing/bfd" />
  <FilePath name="bfd" path="//TestCenter/mainline/content/routing/bgp" />
  <FilePath name="eoam" path="//TestCenter/mainline/content/routing/eoam" />
  </FileGroup>
  </FileGroups>
```

EXAMPLE SSCC RESULT FILE

```
<?xml version="1.0" ?>
- <Report branch="mainline" startbuild="3.50.4330" endbuild="3.50.4810">
- <FileGroup name="IPS" owner="Marc Hornbeek">
  <FilePath name="PortConfigs" changes="0" />
  <FilePath name="Rfc2544Back2Back" changes="0" />
  <FilePath name="Rfc2544Common" changes="0" />
  <FilePath name="Rfc2544FrameLoss" changes="0" />
  <FilePath name="Rfc2544Latency" changes="0" />
  <FilePath name="Rfc2544Throughput" changes="0" />
  </FileGroup>

- <FileGroup name="Routing" owner="Owner Name">
  <FilePath name="bfd" changes="0" />
  <FilePath name="bfd" changes="8" />
  <FilePath name="eoam" changes="0" />
  </FileGroup>
  </Report>
```

CONCLUSION

Code changes automatically determine test selection and test results are automatically targeted back to code faults. The resulting closed-loop continuous build/test environment is more efficient than traditional regression approaches which bottleneck many continuous change and Agile environments. The change driven method enables higher test coverage during continuously varying build schedules and priorities.

REFERENCES

- **A Systematic Review on regression test selection techniques;** Emelie Engström, Per Runeson, Mats Skoglund, Department of Computer Science, Lund University, SE-221 00 Lund, Sweden
- **An Optimized Change-Driven Regression Testing Selection Strategy for Binary JAVA Applications;** Sheng Huang, Yang Chen, Jun Zhu, Zhong Jie Li, Hua Fang Tan; IBM China Research Laboratories, and Department of Computer Science, Tsinghua University Beijing
- **An Empirical Study of Regression Test Selection Techniques;** Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, Gregg Rothermel; Ohio State University, Oregon State University, and University of Maryland
- **Market Survey of Device Software Testing Trends and Quality Concerns in the Embedded Industry**, Wind River, June 2010
- **TCASE: A Practical System for Creating Successful Test Automation Business Cases**, Marc Hornbeek, 2010
- **TCHANGE : A Practical System For Implementing Change-Driven Test Methodology,** Marc Hornbeek, 2010
- **TSCRIPT: A Practical System for Realizing Exemplary Automated Test Scripts,** Marc Hornbeek, 2010

Author Bio

Marc Hornbeek manages automation of the software build and test infrastructure at Spirent Communications. He has been the primary architect of test automation tools and the champion of test automation for multiple firms ranging from start-ups to large multi-national companies. Marc has published more than 30 articles in the field of test automation and business systems and has been the speaker, speaker session organizer, rapporteur for many public conference events including IEEE, ISO, ITU, ISS, NCF, NEC, Interop, Comnet, Infocom, Globecom, IWPTS and User Forums. [1]

DEFINITIONS

ChDT	Change-Driven Test Methodology. A trademark of ManageCycle.com
Code Group	A group of code modules
Code Group Name	The name of a group of code modules
Code Group Owner	The name of the person who is responsible for the code group
Code Module meta data	The code module information used by ChDT calculations
Code Owner	The name of a person responsible for a code module
Code Range	The from/to range of software build numbers or software build dates for which the ChDT code analyzer operates
CODTEF	Code to Test correlation factors used in ChDT Matrix. A trademark of ManageCycle.com
CODTEF matrix	Matrix relating all the code modules to all the test cases using CODTEF values
CRH	Code Result History is the PASS/FAIL results of thecode module for the most recent test runs that resulted in PASS or FAIL result
cx	code module id "x"
dF	The incremental adjustment to CODTEF values after a test FAILs
dP	The incremental adjustment to CODTEF values after a test PASSs
FAILS	The test completed and was not successful.
LCRT	Last Code Result Time is the date/time of the end of the last test run for which the test for a code module resulted in a PASS or FAIL result
LCTR	Last Code Test Result is the last PASS/FAIL test result for the code module during the most recent test run that resulted in PASS or FAIL result
LTR	Last Test Result is the PASS/FAIL results of the most recent test run that resulted in PASS or FAIL result
LTRT	Last Test Result Time is the date/time of the end of the last test run for which the test resulted in a PASS or FAIL result
M	Must-do tests.. Specified by the user input. This is a list of tests that must be performed regardless of CODTEF factors.
NA	Test result not available. The test was attempted but for any one of a number of reasons was not able to conclude PASS or FAIL verdict.
PASS	The test completed and was successful.
Q	Quality level requested byt the ChDT user. This Q system input specifies the minimum CODTEF value to be used by the Test Selection Algorithm.
Rx	Resource "x" is the id of a resource needed to run a test case.
SSCC	SmartTest Source Code Counting tool
T	Time system input that the ChDT user specifies as the maximum amount of time allowed for the requested test run. This is used in the Test Selection algorithm.
TE	Test Exceeded is a system input parameter set by the user. TE is set of guard values to make sure at least a minimum number of tests are selected for each of the Test Selection algorithm terms. This is expressed as a %.
Testcase meta data	The test case information used by ChDT calculations
TRH	Test Results History is the record of last "n" PASS/FAIL test results
TS	Test Selection. This is an algorithm with multiple terms that governs which tests are selected for a particular test run, in accordance with the system inputs.
TT	Time to Test is the duration how long a test takes to run.
tx	test case id "x"
Verdict	The final result of a test may be PASS, FAIL, or NA