Presentation
Paper
Bio
Return to Main Menu

P R E S E N T A T I O N

F6

Friday, May 5, 2000
11:15AM

TRIMMING THE TEST SUITE: USING COVERAGE ANALYSIS TO MINIMIZE RE-TESTING

Jim Boone
SAS Institute, Inc.

International Conference On
Software Testing Analysis & Review
May 1-5, 2000
Orlando, FL, USA

Trimming the Test Suite: Using Coverage Analysis to Minimize Retesting

Jim Boone
Senior Quality Assurance Analyst
SAS Institute

Contents

- Why and when to trim the test suite.
- How to trim the test suite with Coverage Analysis System (CAS) tools.
- Who is using CAS tools to select tests.
- How we create CAS databases.
- When CAS methods are not useful.

Overview of SAS Software

- Who we are
  - SAS Institute is the world's largest privately held software company.
  - We develop information delivery and decision support software.
- Size and complexity of system
  - 15 million lines of C-code
  - 99,200 automated tests
  - 15 to 18 vendor platforms per release

Why the need to minimize retesting?

- Verifying that a fix does not cause new defect.
- Need to run on multiple hosts or with multiple configurations.
- Bring timely and reliable closure to production system.
- Fewer tests to run saves time resolving runs.
- Quicker feedback to development.

When to trim test suite?

- Regression cycle.
- Maintenance cycle.
- When code changes are small and timely retesting is needed.

How can you select tests to run?

- Run entire test suite.
  - useful when few tests or time is not an issue.
- Run selected tests based on information from development and educated guess.
- Run selected tests based on Coverage Analysis tools and data.

Coverage Analysis System (CAS)

- Our CAS databases contain information for each source file on which lines are executed and by which test.
- This allows us to be able to select tests to run based on which source file changed.
- We can also further refine our selection based on changes within a source file.

Three Different CAS Selection Methods

- AllTests
  - All tests that execute the modified source file.
- MaxMin
  - The minimum number of tests that will execute the maximum number of lines of source file.
- ByLine
  - All tests that execute lines in the vicinity where the source file was changed.

Advantages and Disadvantages: AllTests

- Advantages:
  - All tests are returned that execute the modified source file, so maximum coverage is provided.
- Disadvantages:
  - Too many tests are often selected.
  - For large source files with small changes, many selected tests do not test the changed code.

Advantages and Disadvantages: MaxMin

- Advantages:
  - Consistently provides minimum number of tests that execute changed source file.
- Disadvantages:
  - Tests are often excluded that would find a problem -- least reliable method.
  - Slower to determine this set of tests than AllTests.

Advantages and Disadvantages: ByLine

- Advantages:
  - Only targeted tests are selected that execute code around the changed lines.
  - Can be significantly fewer tests than AllTests.
- Disadvantages:
  - Test suite may be too large to run in time available.
  - Test suite may not be reduced from AllTests.
  - May return no tests (for example, for new code).
  - More time is needed to generate list from CAS.

More about ByLine Method

- We compare the original source file to the changed file to determine the line numbers of the changed boundaries.
- All tests that execute the code within the changed boundaries are selected.
- If ByLine does not return any tests, automated testing will not test code changed for the defect.

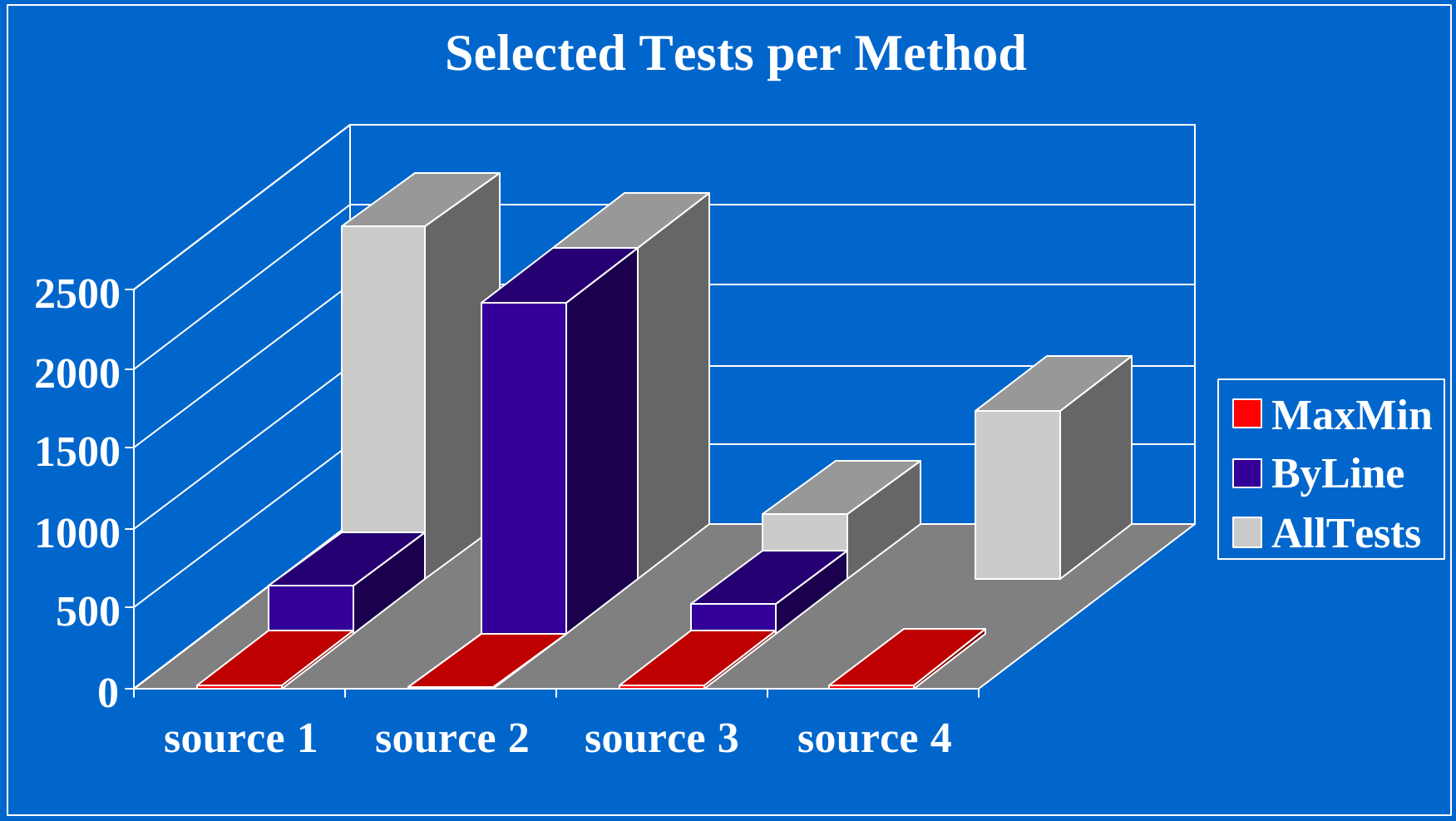
CAS Query Tool

- Queries the CAS databases and provides a set of tests in a format used by our automated test submission tool.
- Has options for AllTests, MaxMin, and ByLine selection.

When do we use each method?

1. AllTests -- if number of tests selected is small, then run these selected tests.
2. ByLine -- when AllTests is too large, then run ByLine-selected tests if not too large or null.
3. MaxMin -- when AllTests and ByLine are both too large, or when AllTests is too large and ByLine is null.

Chart of the three CAS methods

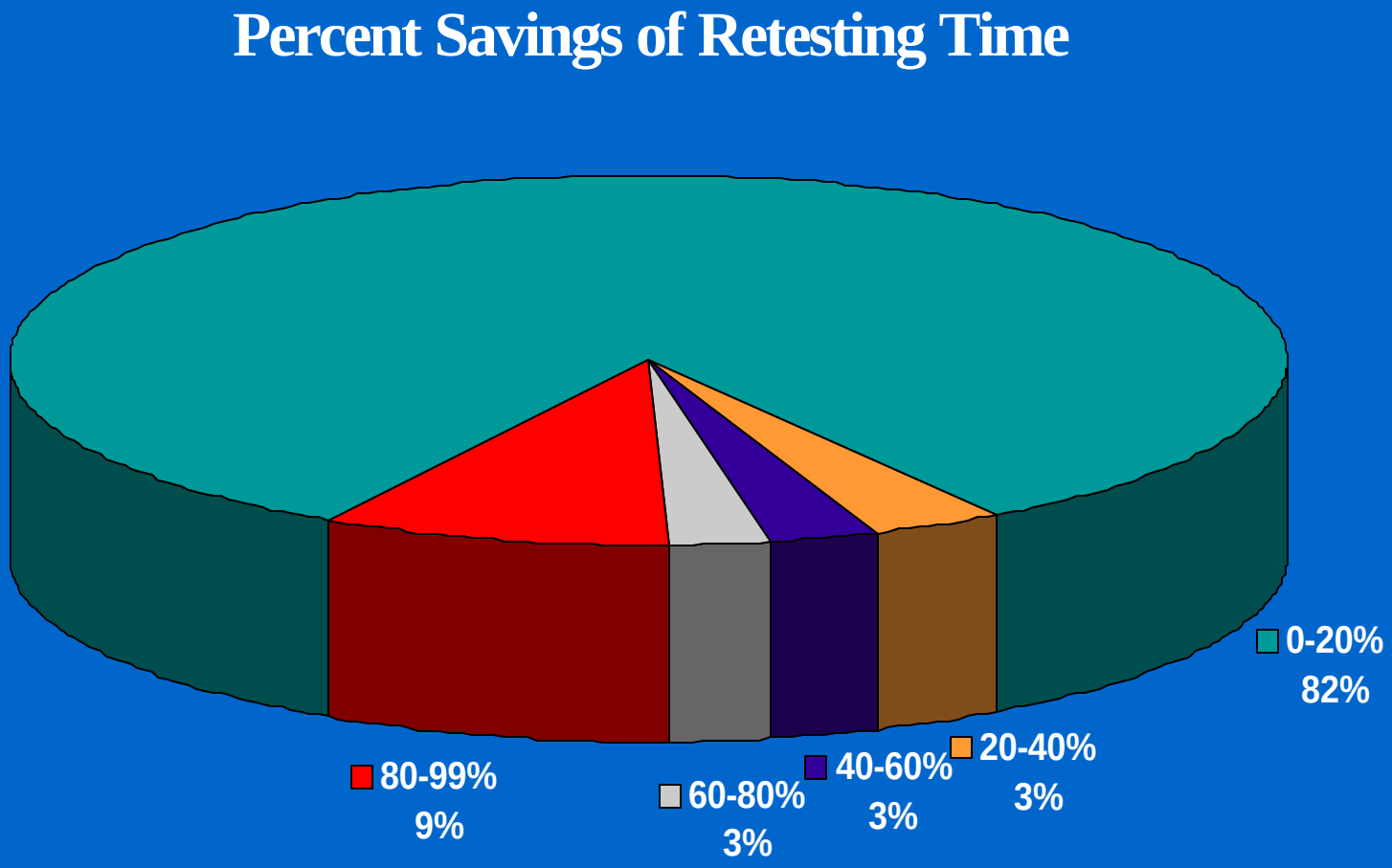


AllTests vs. ByLine (% source files)

- ByLine generated no tests 24.4%
- ByLine generated less tests than AllTests 50.4%
- ByLine and AllTests have same numbers of tests 25.2%

Note: Sample size is 1129 source files

Percent Savings using ByLine vs. AllTests



Who is using our CAS tools?

- Developers
  - Before checking-in corrected source file, run selected test suite in their playpen.
- Testers and Quality Assurance
  - During regression cycle for each new build image on all hosts (15-18 platforms).
  - To test selected defects.
  - To analyze test coverage and write new tests.

Issues with building a CAS for a large system

- Size of files -- CAS files are very large and can cause disk space problems.
- Length of time to run tests and build coverage database can be an issue (we build over each weekend).
- Time to query -- need for quick queries is imperative.

How we create our CAS databases

- Run all tests to generate coverage data files
- Coverage data files are then compressed.
- Selectively copy test's coverage data files based on mapping for each source directory.
- Uncompress and filter in place to reduce size and subset information.
- Filtered data files are written into the CAS database.

How and why we map and filter

- Why
  - Reduce size of files and CAS database.
  - Quicker builds.
  - Quicker queries.
- How
  - Test selection mapping is customized for testing area.
  - Filter is based on source directory.

Mapping of tests to source directories

- There is a many-to-one relationship between tests and source directories and between databases and source directories.
- An example for Source Directory 1 is:
  - TestDir1 TestDir2 TestTable1 > SourceDir1
- Tests chosen to go into SourceDir1 database are usually selected because they were written to test that code area.
- Mappings are customized by testers in their area.

When is CAS not useful for test selection?

- New development and major rewrites to existing code -- cannot use existing coverage to generate test selection.
- Change is to non-source files -- headers, configuration files, etc.
- When test suite is small enough to run all tests with each new image.

Conclusion

- We use our CAS to minimize retesting during the regression cycle.
- Ability to subset test suite allows for quick and reliable retesting to take place on multiple hosts in timely manner.
- Using CAS to trim the test suite during regression allows us to validate our production system faster and more reliably.

Trimming the Test Suite:
Using Coverage Analysis to Minimize Retesting

Abstract

When you have 15 million lines of code to test, and you need to get a quality product to market as soon as possible, you need ways to streamline testing. This paper explores how to use Coverage Analysis System (CAS) data to determine the optimum set of automated tests to execute for a corrected defect. We have derived different methods using CAS data:

- Selecting all of the tests that execute the modified source file
- Selecting the minimum number of tests that provide coverage for as many lines in the source file as have been executed
- Selecting all of the existing tests that execute lines in the vicinity of where the source file was changed.

Strengths, weaknesses, and the best stage for using each method are given.

Written by

Jim Boone
Jim.Boone@sas.com

SAS Institute Inc.
1 SAS Circle
Cary, NC 27512
(919) 677-8000 x6871

and

Pat Berryman
Pat.Berryman@sas.com

SAS Institute Inc.
1 SAS Circle
Cary, NC 27512
(919) 677-8000 x7137

Introduction

There are several global approaches to select tests:

- Select all the tests. Running all of the tests works for small system, but not for large ones, especially when the changes are small and you need to quickly determine if they introduced a new defect.

- Select tests based on information provided by the software developer. Selecting tests based on information from software developers depends on excellent communication between the software developer and tester, and the tester's knowledge of the testware. This approach highlights the value of keeping experienced staff, but this approach can fail because of human fallibility.

- Select tests based on Coverage Analysis System (CAS) data and tools.

This paper discusses the third approach, selecting tests based on CAS data and tools.

Traditionally, CAS data is useful in determining that enough tests have been written, and in identifying source lines that have no test coverage. However, we are using Coverage Analysis to analytically select which tests to run based on source file changes. We have implemented three different methods, each with advantages and disadvantages, of using the CAS information to select tests.

Why trim the test suite?

At SAS Institute Inc., the world's largest privately held software company, we develop and test information delivery and decision support software that contains over 15 million lines of C-code. We run 99,200 tests and release production software on 15 to 18 different platforms at a time. The task of fixing those last defects and verifying that each release is production quality is very time-consuming.

Trimming the test suite is effective if you need to:

- test a large software system with millions of lines of code
- run thousands of tests
- test various configurations
- test on several operating systems.

Running and reviewing only pertinent tests can significantly reduce the time-to-market.

CAS data and tools help us trim the number of tests to those that are most likely to find a problem when verifying each fix. Selectively running and resolving fewer tests allows us to determine new problems more quickly, which in turn helps us move the software to production. Time-to-market concerns make it important to achieve production quality as soon as possible, so we use CAS methods to reliably speed up the process.

When to trim the test suite

We have found that the Regression and Maintenance phases are the most useful phases in the software life cycle in which to use CAS tools to trim tests. During these phases, new source code is not added and only small changes should be made. Since only small changes are made to the source files, CAS data will be applicable for the source file both before and after the changes.

We find the end of the Regression phase the best time to use CAS data. During this time the code changes are generally small which is necessary for these methods to have validity.

CAS methods to trim test suite

We have implemented three different methods:

- AllTests
- MaxMin
- ByLine.

Each method has its advantages and disadvantages.

AllTests

The AllTests method selects all of the tests that execute the modified source file. The advantage of this method is maximum coverage while providing pertinent tests. Because all of the tests returned execute the source file, this method is much better than random selection. In fact, the tests that are selected provide a complete, relevant set of automated tests.

However, there are disadvantages using tests provided by this method. First, the AllTests method often returns more tests than can be run and quickly reviewed. Secondly, for large source files, many of the selected tests do not execute the changed code and do not need to be run.

MaxMin

The method, MaxMin, provides the minimum number of tests that provide maximum coverage in the source file that has been executed. This means the set of tests is the smallest number of tests that executes all source lines that are executed by the entire test suite. Its advantage is that it consistently provides the minimum number of tests that test the source file. The disadvantages are that tests are often excluded that would find a problem, and it is somewhat slower to determine this set of tests than for AllTests. The reason it is slower is that MaxMin first determines the list of tests for AllTests and then minimizes this list.

Below is an example of Max/Min coverage summary.

```
/sas/r800/af/src/ztaf.c Total coverage: 89
Total lines: 191   Executed lines: 67   Executable lines: 75
Coverage:  75( 75)      af:test:abap     abapab01        New Hits: 56
Coverage:  81(  8)      af:test:affrx2   afcngz01        New Hits: 6
Coverage:  81(  3)      af:test:afoop2   aznmem01        New Hits: 2
Coverage:  59(  1)      af:test:abmisc   abdpgb03        New Hits: 1
Coverage:  77(  1)      af:test:abhl     abhlbgl3        New Hits: 1
Coverage:  52(  1)      af:test:Slide7   frdmgv02        New Hits: 1
```

In this example, the first line is the name of the source file followed by the total percent executed lines of executable lines of the file. The second line shows the total number of lines in file, the number of executed lines and number of executable lines. The rest of the lines are specific tests and have the format:

Coverage: % coverage for test (additional % (with this test) SrcDir:Test:Table Name Test Name # of new lines hit

A test table contains a list of tests that can be run as a group.

ByLine

The ByLine method selects all of the tests that execute lines of code in the vicinity where the source file was changed. The advantage of this method is that it selects tests that execute lines where code changed. A test executes the source file but doesn't execute lines of code around where code was changed, that test does not get included in the list. So significantly fewer tests can be returned than using AllTests method.

ByLine has the following disadvantages.

- Sometimes, the number of tests is not smaller than AllTests set of tests.
- There are instances when ByLine does not return any tests.
- ByLine is the slowest of the methods to determine the list of tests.
- The ByLine method is the most sensitive to out-of-date CAS data. (See the section "Building our CAS databases" for more information.)

In order to find the line numbers of the changed parts of the file, we compare the version of the source file before the change to the version after the change. We use two *differencing* techniques to find the line number boundaries of changes in the source. These techniques handle instances when code is added, removed, and changed. We combine the results of both techniques when querying the CAS databases.

The ByLine method is most useful and needed when a large number of tests is returned by AllTests method. At SAS Institute, we have host-layer and core-layer source code that is executed for most every test, so the number of tests that would get returned for changes to these files would very high. So targeted selection of tests that execute lines that changed can reduce the number of tests, but not always.

If ByLine method does not find any test cases, then automated tests from any method are not likely to be relevant. Notification is sent to the tester with the name of the source file and the defect identifier so that tests or interactive scripts will be written. This process provides feedback to testers about where extra testing is needed as related to a specific defect.

Comparisons of the AllTests and ByLine methods

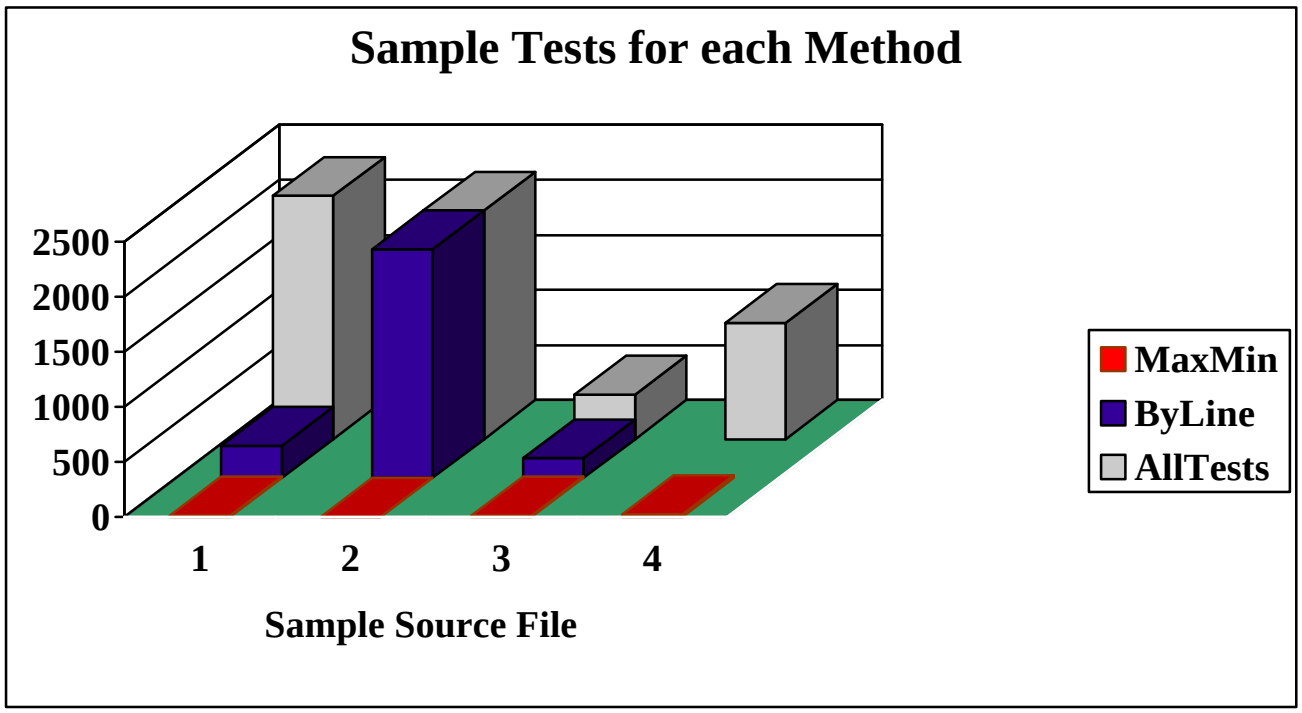
Generally, AllTests is run first and is used if the number of tests is sufficiently small: otherwise, the ByLine method is run. As stated above, the number of tests may not be reduced compared to AllTests, or ByLine may not return any tests.

Comparing the number of tests returned by the ByLine method against the number of tests returned by AllTests method for changed source files over several weeks prior to regression, we get the following results.

No tests are generated using ByLine method	24.4%
The number of tests using ByLine were less than AllTests	50.4%
The number of tests using ByLine and AllTests were the same	25.2%

These results are based on a sample of 1129 source files. During this time, ByLine was helpful for about half the source files by reducing the number of test to run. It was helpful another quarter of the time by indicating that automated tests would not provide coverage for the source change.

Chart of The Three CAS Methods

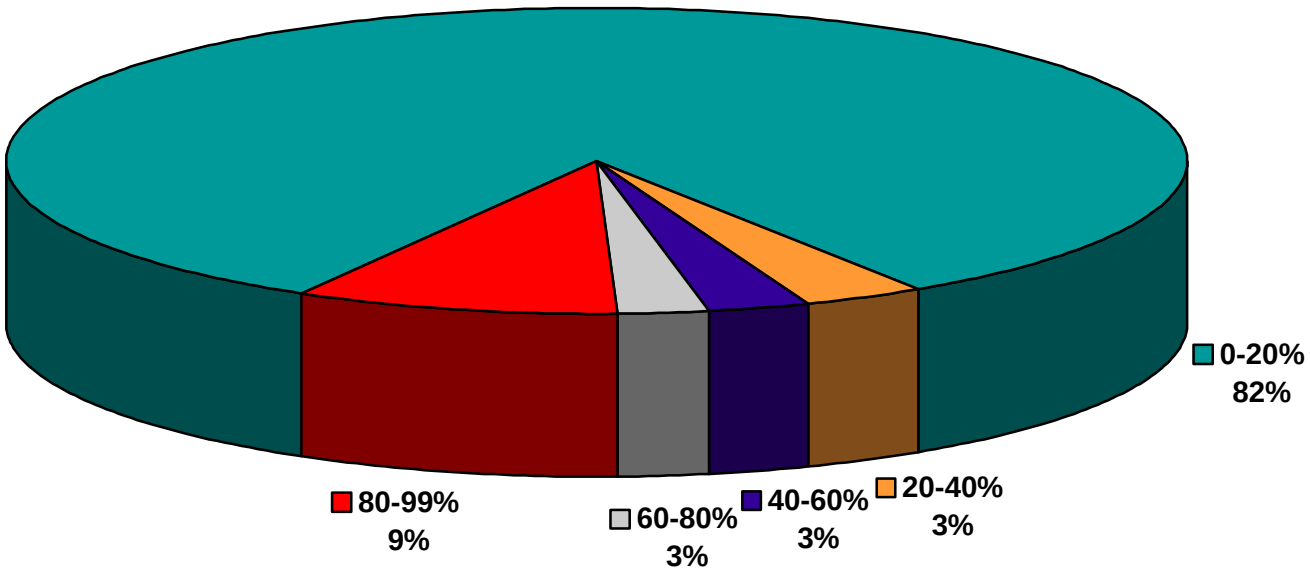


Number of tests Returned by Each Method for Sample Source Files

	Source 1	Source 2	Source 3	Source 4
MaxMin	14	2	13	20
ByLine	295	2075	178	
AllTests	2207	2075	403	1052

In the diagram above, for source 1, the number of tests returned is significantly reduced using ByLine compared to AllTests. For source 2, there isn't any reduction using Byline. For source 3, there is moderate reduction. For source 4, ByLine does not find any tests. MaxMin found a small number of tests for all of the source files, yet its use is risky because relevant tests may be missed.

Percent Savings using ByLine vs. AllTests



Where the Percent Savings in Retesting Time is calculated using the formula:

$$\% \text{ Savings} = \frac{\#\text{ of Tests from AllTests} - \#\text{ of Tests from ByLine}}{\#\text{ of Tests from AllTests}}$$

The chart above illustrates the percentage of savings by using ByLine versus AllTests. This chart is based a sampling of 854 source files where ByLine found test coverage. There are two clusters of significance:

- Most source files did not achieve much savings (0-20%).
- The significant statistical cluster achieved between 80 to nearly 100% savings. This group is where ByLine is useful, especially if the number of tests from AllTests is very large.

When are CAS methods not useful

New development and large rewrites to existing code are the primary times when CAS methods are not useful for selecting tests to run. Before tests are run and coverage data is collected for the new code, the CAS databases can't know which tests execute this new code. Likewise, when code is moved from one

source file to another, the CAS data is not useful due to the dependence on source file location.

When a change is made to code other than to executable source files, such as header or configuration files, CAS tools are not useful. Our system mainly contains line numbers for executable code. CAS databases do not retain information about files such as C-code header files because that code is not executable.

When the test suite is small enough to run all of the tests with each new image in the time available, it is best to run the whole test suite without trying to trim it.

Software communities using the tools

We have two users groups that select tests to run using CAS tools and data: developers and testers. Developers are using CAS during software development to select automated tests to run and review before pushing code changes to source library. They tend to use MaxMin to provide a representative sample of tests to run before finalizing the change unless it is late in Regression cycle. Late in the Regression cycle, we ask them to use the tests from AllTests and ByLine methods to achieve a more thorough regression check before pushing.

Testers use AllTests and ByLine methods during the final stages of regression. We use tests from AllTests if the number is manageable; otherwise, we run ByLine. When we run the CAS tool, we can specify a maximum number of tests for AllTests. If the number specified is exceeded, then the ByLine test set is determined. If ByLine does not return any, the AllTests test set is kept.

Building our CAS databases

Building CAS Databases

We begin building CAS databases on late Friday afternoons. First, we run all of the tests using an option to generate coverage data for each test. Next, these data files are compressed and remain organized according to our testware directory structure.

The next step is to selectively copy coverage data files from the testware directory structure to a source directory organization. We have a many-to-one mapping of testware to the source directory structure that indicates which tests go into each CAS database. An example for Source Directory 1 is:

TestDir1 TestDir2 TestDir3/TestTable1 $\Rightarrow$ SourceDir1

TestDir1 and TestDir2 include all coverage data files from all tests in TestDir1 and TestDir2. TestDir3/TestTable1 includes coverage data files for a test table from a different test directory.

The tests selected to go into the SourceDir1 database are chosen because these tests were written to test the source code in SourceDir1. Refining these mappings requires expert knowledge about the testware and source code.

After the coverage data files are copied to the source directory structure, they are uncompressed and filtered so that only the information for files in the source directory is kept in the data file. Finally, the filtered data files are written into the CAS database.

Issues and Concerns Building CAS databases

There are three primary concerns that we have to address when we are building our databases.

- Size. One of the issues about building CAS databases for a large system is the potentially large size of files and databases. The larger the CAS databases are, the longer they take to build. Secondly, larger CAS databases take longer to query.

- Reliability. We build all CAS databases during the weekend so that they can be used Mondays. If the process fails, we are left with out-of-date CAS data. The ByLine method is the most sensitive to out-of-date CAS data. One of the versions of the source file, before or after its changed, must match the content of the CAS database for the database to be reliable.

- Performance. Performance during building and accessing databases is slower for larger databases. To reduce their size, we selectively map tests to the source and are filtering the data files so that a database has the only the information for a single source directory.

Conclusions

We are using three methods with Coverage Analysis tools and data to trim our test suite. These methods provide reliable and quick empirical approaches to selecting tests to verify corrected defects. These techniques work best during late regression and maintenance cycles when source changes are small and new code is not added. They are especially helpful in speeding up the regression cycle by bringing closure to a production system in a reliable manner.

Jim Boone

Jim Boone is a Senior Quality Assurance Analyst at SAS Institute, Inc. Mr. Boone designed, wrote and supported configuration, source code and test management tools for 7 years at SAS Institute before moving to Quality Assurance.

In the last five years in QA Mr. Boone has been investigating enhancements to our Coverage Analysis System (CAS) with a focus on testing and using SAS' in-house developed CAS tool set and later finding ways to minimize re-testing using CAS.

Mr. Boone is also the lead test analyst for the SAS Component Language (SCL).