GERT: An Empirical Reliability Estimation and Testing Feedback Tool

Martin Davidsson, Jiang Zheng, Nachiappan Nagappan, Laurie Williams, Mladen Vouk

Department of Computer Science
North Carolina State University, Raleigh, NC 27695
{mhdavids, jzheng4, nnagapp, lawilli3, vouk}@ncsu.edu

Abstract

Software testing is an integral part of the software development process. Some software developers, particularly those who use the Extreme Programming test-driven development practice, continuously write automated tests to verify their code. We present a tool to complement the feedback loops created by continuous testing. The tool combines static source code metrics with dynamic test coverage for use throughout the development phase to predict a reliability estimate based on a linear combination of these values. Implemented as an open source plug-in to the Eclipse IDE, the tool facilitates the rapid transition between unit test case completions and testing feedback. The color-coded results highlight inadequate testing efforts as well as weaknesses in overall program structure. To illustrate the tool's efficacy, we share the results of its use on university software engineering course projects.

1. Introduction

With the increasing recognition that software reliability is essential to the success of a software project, the demand for tools to estimate this measure is apparent. Additionally, it is desirable for software reliability estimation to be iteratively calculated throughout the development process, rather than after the project has reached the system test stage or after completion when changes are more costly. As such, we are developing an open source tool that is integrated tightly with the developer's existing development environment. The prototype tool is called the "Good Enough" Reliability Tool (GERT). The "good enough" in the name acknowledges that the reliability estimates provided by the tool may have larger confidence bounds than those based on operational profile associated testing, such as those discussed in Musa [6], but may be "good enough" to guide the testing process, especially at the unit level. GERT[1] is available as an open source plug-in under the Common Public License (CPL[2]) for the open source Eclipse[3] development environment

GERT provides a means of calculating software reliability estimates and of quantifying the uncertainty in the estimate (a.k.a. the confidence interval). The estimate and the confidence interval is built using the Software Testing and Reliability Early Warning (STREW) in-process metric suite [7, 8]. This suite is discussed in Section 3. Additionally, the tool provides color-coded feedback on the thoroughness of the testing effort relative to prior successful projects.

The tool is designed for ease of use. GERT is intended for development teams that write extensive automated test cases, as is done with the test-driven development (TDD) practice [2] which has been recently popularized by the Extreme Programming [1] software development methodology. GERT complements the feedback loops normally created by controlled continuous testing. The tool combines static source code metrics with dynamic test coverage information.

The objective of this work is to create and provide an evaluation of the GERT. We describe the structure and the capabilities of GERT and report on the use of the tool to analyze the reliability and testing of four student projects in a junior/senior software engineering class at North Carolina State University (NCSU). Our analysis focuses on whether the information provided by the tool provides the developer with the appropriate direction for improving the testing process and, thus, the reliability of the product.

The remainder of this paper is organized as follows. Section 2 provides background information on existing reliability tools. Section 3 discusses the metrics that are collected by the tool. Sections 4 and 5 explain the reliability estimation model integrated into GERT and the test quality feedback provided by the tool. The GERT architecture is discussed in Section 6. Section 7 provides a sample illustration of use. Class project results are discussed in Section 8. Finally, we conclude and discuss

[1] GERT can be obtained from http://gert.sourceforge.net. GERT was a winner in the International Challenge for Eclipse competition in the student project category

[2] http://www-124.ibm.com/developerworks/oss/CPLv1.0.htm

[3] Eclipse is an open source integrated development environment. For more information see http://www.eclipse.org

future work in Section 9. Screen shots and comments on four student projects are provided in the Appendix.

2. Some Existing Tools

The Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) [11] is a menu-driven tool that estimates reliability of software systems using a black-box approach. It provides a range of reliability models, but it assumes that software testing is done using an operational profile. SMERFS Cubed [4], the latest evolution of the tool, is available for the Windows operating system. This latest version extends the functionality to support both hardware and systems and has a graphical user interface.

The Automated Test Analysis for C, or ATAC[5], is a white-box tool that evaluates the efficacy of software tests using code coverage metrics. The tool, run from the command line, comes with a specialized compiler (`atacCC`) that creates instrumented binaries. The run-time trace file records block coverage, decision coverage, c-use, p-use, etc. The tool, through coverage, assesses test completeness and visually displays lines of code not exercised by tests, and reduces test set sizes by avoiding overlap among tests.

Computer-Aided Software Reliability Estimation (CASRE[6]) is another tool for black-box software reliability estimation. The models incorporated in CASRE are to a large extent the same as those in SMERFS. Neither tool has models that incorporate test coverage. CASRE is Windows-based and it has the ability to aggregate results from several different models into a single reliability estimate. By taking linear combinations of several component models and applying specific weights to each, one can construct more complex models which may improve predictive reliability estimates. The tool takes advantage of windowing environments. However, similar to SMERFS CASRE lacks an automated method for collecting data. As a result, its use may not scale well in larger software projects. Furthermore, none of the models incorporated in CASRE consider test coverage data in their analysis.

Software Reliability and Estimation Prediction Tool (SREPT[7]) can be used to assess the reliability of software across several stages of its life-cycle [10]. By combining static complexity metrics for early prediction and failure data obtained later in the project, SREPT can be used to estimate reliability as soon as the software's architecture is in place. From its results, SREPT can also project release times based on certain criteria.

Finally, the Reliability of Basic and Ultra-reliable Software systems (ROBUST[8]) [5] is a tool that supports five different software reliability growth (SRG) models, one of which involves test coverage. Two of the other four models can be adapted for use with static metrics for estimation in the early stages of development. The tool increases accuracy of the SRG models by supporting any combination of recalibration, stabilization, and data smoothing. ROBUST operates on data sets of failure times, intervals, or coverage. As with most of the other tools, the data may be displayed in text form to view individual data points or in graph form based on the active model.

A limiting feature of these popular reliability tools is the dependency on external data sources, i.e. the tools tend to be a separate part of the software process and thus may require a considerable amount of extra work on the part of end users before they can play a useful part in process steering. GERT attempts to ameliorate this extra work.

3. Metrics Collected

The GERT estimates are based, in part, on the Software Testing and Reliability Early Warning (STREW) [7, 8] metric suite of internal, in-process software metrics. These metrics are intended for a) early, in-process use and b) to cross-check each other. GERT currently collects the following metrics of the STREW Version 1.4 suite (SLOC = source lines of code):

1. *number of test cases / SLOC (R1);*
2. *number of test cases / number of requirements (R2);*
3. *test lines of code / SLOC (R3);*
4. *number of assertions / SLOC (R4);*
5. *number of test classes / number of source classes (R5);*
6. *number of conditionals / SLOC (R6);*
7. *SLOC / number of source classes (R7);*
8. *statement coverage (R8); and*
9. *branch coverage (R9).*

Note that for all metrics above, except R6 and R7, large values of the metric is potentially good (and may indicate better quality), while low values are a potential indicator of poor quality. Initial experiments with the use of STREW metrics (on student and industrial) programs have been encouraging. However, not all metrics have consistently demonstrated a correlation with software reliability. Work on identification of an optimal set of metrics is still in progress.

4. Reliability Estimation Model

[4] http://www.slingcode.com/smerfs

[5] http://dickey.his.com/atac/atac.html

[6] http://www.openchannelsoftware.com/projects/CASRE_3.0/

[7] http://www.ee.duke.edu/~kst/software_packages.html

[8] http://www.cs.colostate.edu/testing/robust/

GERT utilizes a multivariate regression model to predict reliability. The regression equation is constructed with the reliability (or failure rate) as the dependent variable and the empirical values of STREW metrics as predictors. Based on these coefficients the current values of the metrics are calculated to provide an empirical estimate of the reliability. Again, for this model to have practical validity, one has to assume the existence of an extensive and representative suite of test cases early in the software development process.

The current version of GERT recognizes test cases and assertions that are written for use with the JUnit[9] testing framework. For example, Extreme Programmers that practice TDD may create such a test suite. With TDD, software engineers develop production code through rapid iterations of the following steps: (1) writing a small number of representative automated test cases; (2) running these unit test cases to ensure they fail (since there is no code to run yet); (3) implementing code which should allow the unit test cases to pass; (4) re-running the unit test cases to ensure they now pass with the new code; (5) refactoring of the implementation and test code, as necessary; and (6) periodically re-running all the test cases in the code base to ensure the new code does not break any previously-running test cases. Note that in these iterations, test cases will almost always fail initially because test code is written prior to implementation code. They are all meant to ultimately pass. These automated test cases also serve as regression tests. It is important to note that all discussion of assertions in this paper are with regards to JUnit assertions only, not the recently-added support for assertions in the Java language itself.

The regression equation is built using a set of historical values of the STREW measures. Ideally, this set of historical values is collected for each individual engineer or team to compensate for stylistic differences among programmers. For example, one developer might write fewer test cases, each with multiple assertions checking various conditions. Another developer might test the same conditions by writing many more test cases, each with only one assertion. We intend for our tool to provide useful guidance to each of these developers without prescribing the style of writing test cases. Through our research, we also intend to calibrate a default regression equation that can be used in the absence of historical values.

The confidence interval around the point estimates from the model is calculated using Equation 1 [9], where $Z_{\alpha/2}$ is the upper $\alpha/2$ quartile of the standard normal distribution, R is the reliability point estimate, and n is the number of test cases included in the project.

$$\text{Confidence interval} = R \pm z_{\alpha/2}\sqrt{\frac{R(1-R)}{n}} \qquad (1)$$

[9] http://junit.org/

While Equation 1 is a relatively simplistic approach to the computation of the confidence bounds, we believe that it is "good enough" given the context in which it is used and provides for valuable early information.

5. Test Quality Feedback

In addition to providing a reliability estimate, the tool also provides color-coded feedback on the thoroughness of the testing effort relative to the historical data from comparable projects. Color coding alerts developers as to whether a metric value is within acceptable limits. The acceptable limit boundary of the metric values for R1-R5 and R8-R9 is calculated using Equation 2 with a negative sign (i.e. subtraction). The mean of the historical values for each metric serves as the reference limit. The color-coded feedback works in reverse for ratios R6 and R7 because the higher the value the higher the possibility of errors [3]. The color coding scale employed is shown below in Table 1.

$$\text{Bound}(Rx) = \mu_{x+/-}z_{\alpha/2} * \frac{\text{Standard deviation of metric Rx}}{\sqrt{n}} \qquad (2)$$

Table 1: Color-coded feedback scale (except R6 and R7)

Color	Range of metric values
RED	Rx < Bound(*Metric*)
ORANGE	Bound(*Metric*) ≤ Rx ≤ μ_x
GREEN	Rx > μ_x

where μ_x = Mean of the metric x; Rx is the metric ID. For example, if the ratio R3 is below the bound, the value will display in red, indicating that the ratio of test lines of code to the source lines of code is less than in previously successful projects and more test cases should be written to bring the value of the ratio up.

6. GERT Architecture

GERT is currently available as a plug-in to the Eclipse IDE. The tool is designed so that users can easily migrate between the GERT perspective and their preferred development perspective of the Eclipse workbench. Figure 1 illustrates the default GERT architecture. GERT collects metrics shown in Figure 1 also.

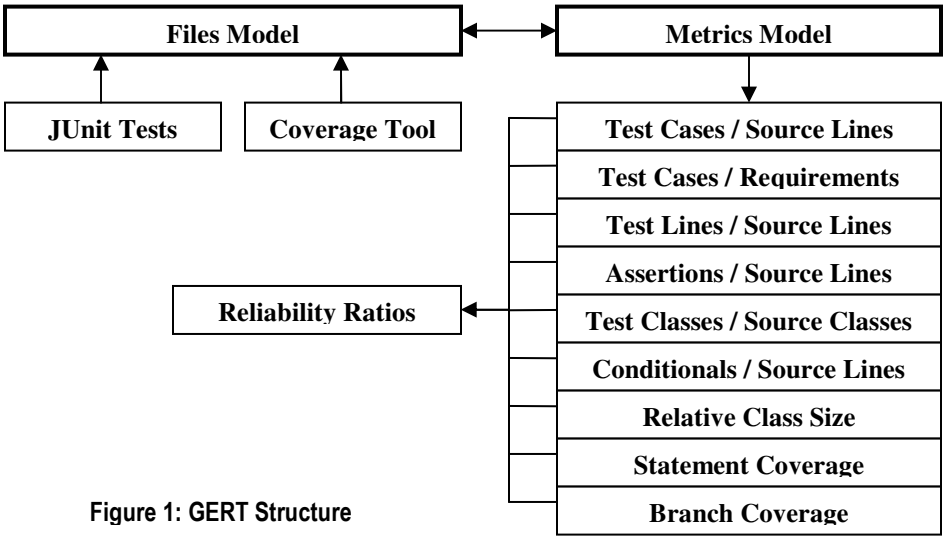

Figure 1: GERT Structure

6.1 Operational Modalities

As shown in Figure 2, a software project to be analyzed is contained in the "Package Explorer" in Eclipse. Any subset of a software project can be chosen from this view to take part in the reliability calculations and test feedback, enabling the tool to isolate reliability issues down to a specific package or even file. These subsets of files are selected through a context menu item as shown on the left side of Figure 2. The portion of the project chosen to be included in the analysis is shown in a tree structure on the right hand panel labeled "Reliability Files." Modifications to this selection are easily performed by repeating the process of adding files and packages or removing files in the "Reliability Files" panel. Each project maintains a separate list of files that is persistently preserved between user sessions. Activating a different project will repopulate the view with files from the new project that were previously chosen for reliability testing, if any. Changes to the file selection triggers the tool to include the metrics of the added source code into its analysis.

Once the analysis has completed, a results summary appears in tabular format. Aggregated metrics for the chosen set of files are displayed foremost in the top table producing the nine metrics (R1-R9 in Section 3). Primitive metrics that are collected for the source files include number of conditionals and statement and branch coverage. Similarly, the number of test methods and number of assertions is collected from the test files. Lines of code and number of classes are summed for both source and test files. Furthermore, all files are partitioned into source and test files. Individual metrics for each file are presented based on the type of the file. GERT enables the user to export all gathered data to a standard comma separated values (CSV) file. From this format, creating custom tables and charts in an external application becomes trivial.

GERT collects static software metrics efficiently and reliably by utilizing the features offered by the Java Development Tools (JDT) framework that comes packaged with Eclipse. Specifically, the Java Abstract Syntax Tree (AST) public class of the JDT is employed to provide syntactic analysis of Java source code. The AST nodes form parent-child relationships ranging from the overall class file down to individual tokens in a line of code. As such, the AST greatly simplifies the task of counting the number of occurrences of any given Java structure or element. GERT implements the Visitor design pattern to traverse the AST of every Java file in the software project being analyzed. The AST Visitor recursively traverses the entire tree while keeping track of how many elements of interest it comes across. For instance, the AST enables GERT to examine the super class of each class file it finds. If the super class is defined by `junit.framework.TestCase`, then the child is taken to be a test file and is filed as such. Additionally, all information is cached as it is processed. After a successful traversal, the relatively expensive procedure is only repeated once the file's last modified time stamp is updated.

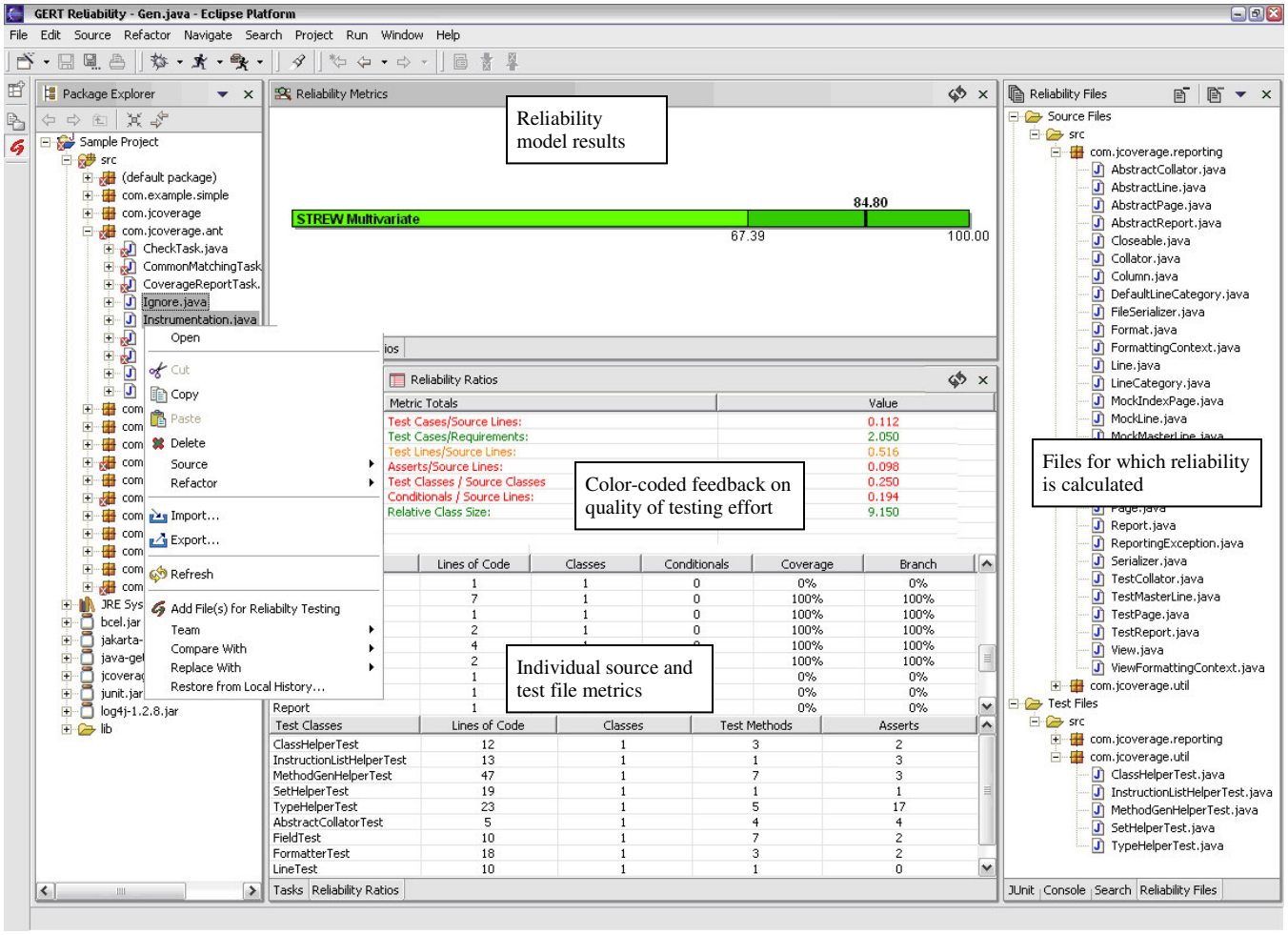

Figure 2: GERT Screen shot

GERT incorporates the open source JCoverage[10] tool for test coverage calculation. The tool determines where class files are built to instrument these classes with coverage probes. Instrumentation occurs in the Java bytecode and tracks execution of individual statements and which conditional branches are taken during successive runs. Instrumentation may be removed by recompiling the project. In turn, GERT scans the project for test cases written for the JUnit testing framework and calls each in order. This artificial usage provides the basis on which the coverage report is generated. The percentage of statements executed and branches taken in the code are displayed along side each source file.

The tool also offers the ability to scan each source file to determine precisely which lines of code were omitted by calls from the JUnit tests. Once the report is complete, double clicking on any file in the "Reliability Files" view will bring the file up in the active editor. If the file did not attain 100% statement coverage, the lines which were not executed by the JUnit tests will appear with a marker (see Figure 6 in the Appendix). We intend to enhance GERT in the future with an indication of the branches that do not get executed. Additionally, the row that corresponds to the active file in the "Reliability Metrics" table will appear highlighted. The line markers are associated with files across the entire project scope. The coverage markers will appear inside the editor for every perspective available in Eclipse.

6.2 Reliability Results Display

GERT currently supports one reliability estimation model: the STREW regression. The results of the model are displayed by a color-coded bar at the top of the screen. In the bar, the reliability point estimate is shown as a bold line. The calculated confidence interval around the point estimate is also displayed. As with all analysis data, the values that compose the graph are easily exported to a comma separated values (CSV) file. Moreover, the graph can be oriented vertically if desired, as shown in Figure 4. Once the environment is set up, the user can return to the

[10] http://jcoverage.org

tool's perspective at any point in the development cycle and retrieve updated values for the reliability metrics.

6.3 Configuration

All preferences for the tool are editable through a sub-page of the Eclipse IDE preference pages. These settings are retained throughout subsequent sessions as is the layout of the tool's different graphical views. The configurations available consist of defining the parameters for the STREW model. If the user wishes to employ a different reliability model, the expression of metrics that determine the point estimate can be entered among these preferences. Optionally, the colors used to highlight the "Reliability Ratios" view and their associated ranges are defined from the same menu.

6.4 Third Party Software

GERT provides an easy-to-use tool for empirical reliability estimation and test feedback. Some of GERT's functionality is handled by other open source tools that have been incorporated in GERT's source code. Currently, the task of performing coverage analysis and administrating unit testing is handled by JCoverage and JUnit, respectively.

- JCoverage, licensed under the GNU General Public License (GPL[11]) is an extension to the Apache Ant build tool.
- JUnit, licensed in similar fashion under the CPL, provides a framework for running unit testing. GERT calculates coverage based upon JUnit test cases.

7. Illustration of Use

We have created a sample project which consists of the JCoverage source code to illustrate the operation of the tool, as shown in Figure 2. The values displayed are not a true reflection of the reliability of JCoverage because we do have any prior data from a comparable tool and are presented for illustrative purposes only. For this example, we analyze two packages consisting of both source files and test files. This is accomplished by selecting both packages in the "Package Explorer" and bringing up the context menu for these items and selecting the "Add Package(s) for Reliability Testing" option. Once the files contained in these packages appear in the "Reliability Files" tree, we execute the analysis action and, upon completion, we refresh the results views. The resulting output appears automatically, based on the parameters established in the preference menu. Point estimates, confidence intervals, and individual file metrics are all viewable from a single screen. The user may immediately resume development by selecting the appropriate perspective shortcut, to improve upon the areas whose estimated reliability was found to be inadequate. For instance, if the user prefers to develop in the default Java Perspective, lines inside a source file which is a member of "Reliability Files" will now be marked as a line which was not executed by the available JUnit tests.

8. Student Projects

Student projects were analyzed using GERT. The students were junior/senior computer science students taking a software engineering class at NCSU. The students developed an open source Eclipse plug-in in Java that automated the collection of project metrics. Each project was developed by a group of four or five students during a six-week final class project. A total of 22 projects were submitted; all were used in the analysis. Table 2 below shows the size of the projects developed.

Table 2: Eclipse project size

Metric	Mean	Std Dev	Max	Min
SLOC	1996.9	835.9	3631	617
TLOC	688.7	464.4	2115	156

The plug-ins were tested using a set of 31 black-box test cases. Twenty six of these were acceptance tests and were given to the students during development. The actual reliability of the student programs was approximated by inputting the results of these black box test cases into the Nelson model. Using a randomly-chosen set of 18 programs, multiple regression Equation 4 was built.

Reliability Estimate = 0.859 + 0.09459*R1 + 0.01333*R2 - 0.0404*R3 + 1.674*R4 + 0.01242*R5 - 1.222*R6 + 0.000867*R7 **(4)**

[Note that R8 and R9 (statement and branch coverage) are not included in this model. When the programs were analyzed, coverage had not been implemented in GERT.]

The coefficients of this model were input into GERT to estimate the four remaining projects (Groups A, N, Q, and X). The data from these projects is presented in Table 3. Screen shots of running GERT on these projects can be found in the appendix. A limitation of this approach is that we built the regression equation using one set of students and use this equation for a different set of students. As discussed above, it is best when the coefficients of the regression equation are built using an individual's or a team's own historical data.

Table 3 shows the overall comparison between the four student projects. The foremost eight rows represent the basic measures, followed by the seven STREW reliability metrics calculated by GERT. The rearmost six rows show the lower limit, point estimate, and upper limit calculated by STREW Reliability Model.

[11] http://www.gnu.org/copyleft/gpl.html

Table 3: Overall data comparison between the four student projects

	Group A	Group N	Group Q	Group X
source lines of code	969	1191	1236	1385
number of source classes	22	15	21	19
number of conditionals	129	140	143	117
number of requirements	20	20	20	20
test lines of code	350	38	497	1332
number of test classes	10	4	8	8
number of test cases	79	16	45	207
number of assertions	77	25	126	667
number of test cases/source lines of code (R1)	0.082 (R)	0.013 (R)	0.036 (R)	0.149 (R)
number of test cases/number of requirements (R2)	3.950 (G)	0.800 (G)	2.250 (G)	10.350 (G)
test lines of code/source lines of code (R3)	0.361 (Y)	0.032 (R)	0.402 (Y)	0.962 (G)
number of assertions/source lines of code (R4)	0.079 (R)	0.021 (R)	0.102 (R)	0.482 (Y)
number of test classes/number of source classes (R5)	0.455 (Y)	0.267 (R)	0.381 (Y)	0.421 (Y)
number of conditionals/number of source lines of code (R6)	0.133 (R)	0.118 (R)	0.116 (R)	0.084 (R)
number of lines of code/number of classes (R7)	44.045 (G)	79.400 (G)	58.857 (G)	72.895 (G)
STREW Reliability (lower limit)	85.615	63.473	90.237	100.000
STREW Reliability (point estimate)	91.895	83.337	96.122	100.000
STREW Reliability (upper limit)	98.174	100.000	100.000	100.000
Nelson Model (Estimate of Actual Reliability)	70.968	77.419	96.774	93.548

Color-coded feedback is shown in the table as well. In the seven rows in which STREW reliability metrics are listed, (R) means that he corresponding ratios displayed in red in the "Reliability Ratios" view, indicating that the ratios are less than those in previously successful projects. Similarly, (G) means the corresponding ratios displayed in green, indicating that the ratios are more than the mean of the corresponding metric of previously successful projects. (Y) displayed in yellow, indicating that the ratios are between the lower limit and the mean of the corresponding metric.

For the process of predicting reliability to become a regular practice of the TDD, tool use must add minimal overhead. We feel the simplicity of use and clarity of results are positive attributes of GERT. Developers can quickly generate new reliability estimates for updated code to see which metric ratios need improvement. Furthermore, the comprehensiveness of unit test cases is revealed in the process.

For the four student projects, the meaning of the actual results was generally sound. However, in one case, the STREW model estimated reliability at 100%. In another case, the estimate was quite different from the actual reliability. We believe that the limitation may lie in the experimental structure rather than in the model. The model was built using data from one set of programmers and used with a different set of programmers. Continued usage of the tool by the same individual or group over time will determine whether or not programming style significantly influences results.

9. Summary and Future Work

In this paper, we presented the "Good Enough" Reliability Tool, an open source tool which provides reliability estimation and test feedback information. The current version of the tool is appropriate for use by software engineers who create suites of automated test cases using the JUnit testing framework. The tool enables reliability estimates to become part of frequent feedback loops. Developers, in addition to working towards passing all tests, can establish goals that lead to source code containing desirable metric ratios where all color-coded ratios show up in green.

In future releases of the tool we intend to incorporate the following features:

- Maintain the collection of historic reliability and metrics values throughout the software project's development life-cycle.
- Build an automatic regression equation calculation system to (1) estimate the defects/SLOC using multiple linear regression and (2) estimate the reliability using a logistic regression equation.
- Automate the operation of the tool so that the tool runs nightly builds automatically in the background without using user intervention
- Define extension-points that follow the Eclipse plug-in framework to allow for additional modular views to integrate with the existing model and controller.
- Include p-use and c-use of source code as part of the suite of metrics and incorporate the values in overall reliability estimates.

The tool is currently being deployed in several industrial locations to collect these metrics to empirically estimate the reliability of the software and to validate the STREW metric suite.

Acknowledgements

We thank Ryan Sturmer, Michael Fogleman, Bob Hopkins for working on G-Unit, a precursor to GERT. We acknowledge the open source GNU license of JCoverage that we adapted and included in GERT. Finally, we thank IBM for the Eclipse Innovation Award that funded the initial research.

References

[1] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.

[2] K. Beck, *Test Driven Development -- by Example*. Boston: Addison Wesley, 2003.

[3] K. El Emam, Benlarbi, S., Goel, N., Rai, S.N., "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Transactions on Software Engineering*, vol. Vol. 27, no. pp. 630 - 650, 2001.

[4] D. Hamlet, Voas J., "Faults on Its Sleeve: Amplifying Software Reliability Testing," International Symposium on Software Testing and Analysis, pp.89-98, 1993.

[5] N. Li and Y. Malaiya, "ROBUST: A Next Generation Software Reliability Engineering Tool," IEEE International Symposium on Software Reliability Engineering, pp.375-380, 1995.

[6] J. D. Musa, *Software Reliability Engineering*. New York: McGraw-Hill, 1999.

[7] N. Nagappan, Williams, L., Vouk, M.A., "Towards a Metric Suite for Early Software Reliability Assessment," International Symposium on Software Reliability Engineering, FastAbstract, Denver,CO, pp.238-239, 2003.

[8] N. Nagappan, Williams, L., Vouk, M.A., Osborne, J., "Initial Results of Using In-Process Testing Metrics to Estimate Software Reliability," North Carolina State University, Raleigh TR-2004-5.

[9] NIST/SEMATECH, *e-Handbook of Statistical Methods*: http://www.itl.nist.gov/div898/handbook/.

[10] S. Ramani, S. Gokhale, and K. S. Trivedi, "SREPT: Software Reliability Estimation and Prediction Tool," 10th Intl. Conference on Modeling Techniques and Tools (Tools '98), Lecture Notes in Computer Science 1469, Palma de Mallorca, Spain, pp.27-36, September 1998.

[11] G. E. Stark, "A survey of software reliability measurement tools," IEEE International Symposium on Software Reliability, pp.90-97, May 1991.

[12] R. Thayer, Lipow, M., Nelson, E., *Software Reliability*. Amsterdam: North-Holland, 1978.

APPENDIX: Example Output from Student Projects

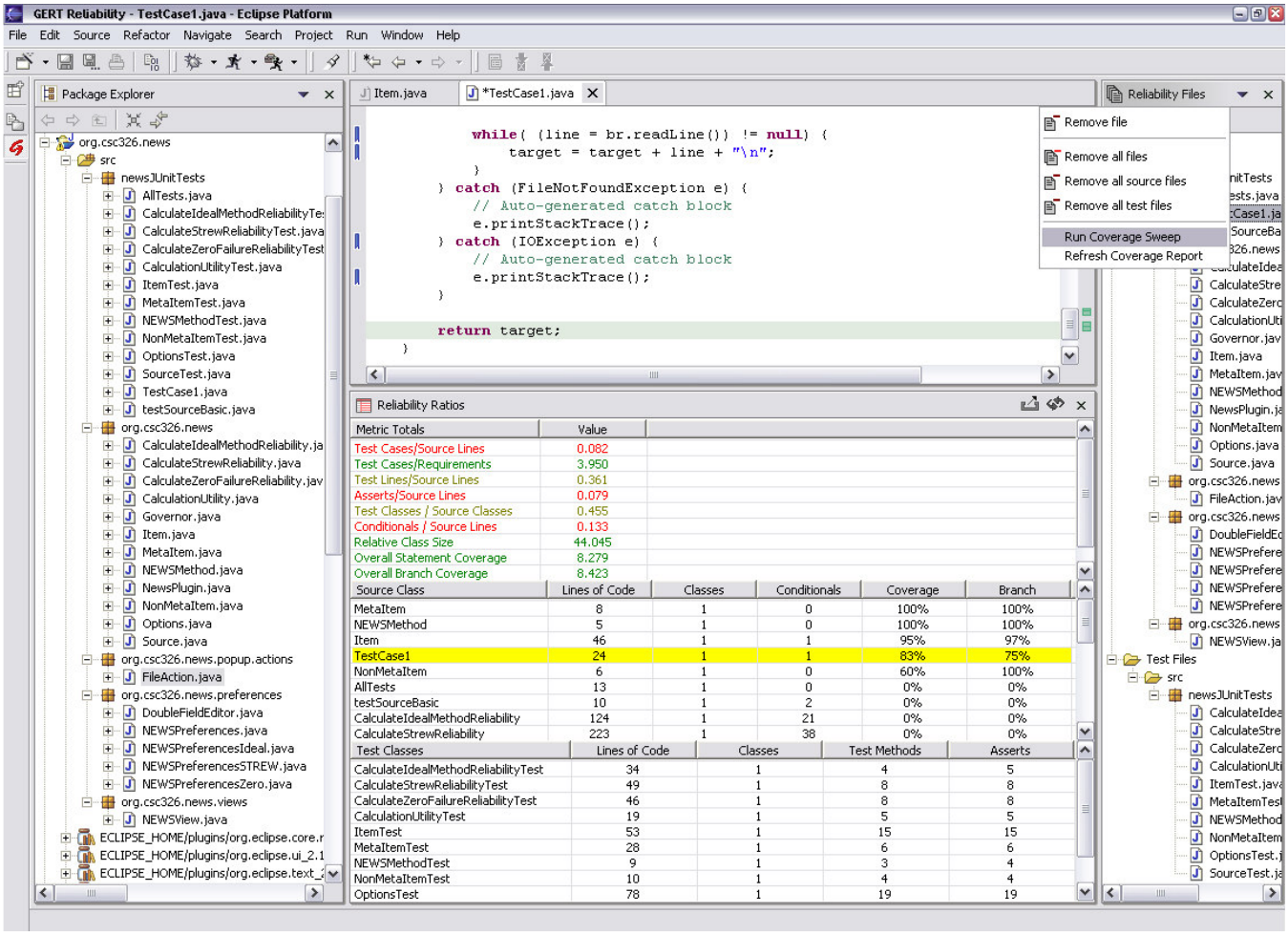

Figure 3: GERT screen shot for Group A

Table 4: Metric Values for Group A (SLOC = Source Lines of Code)

Qty Test/ SLOC	Qty Test/ Qty Req	Tst LOC/ SLOC	Qty Assert/ SLOC	Qty Tst Class/ Qty Src Class	Qty Cond/ SLOC	SLOC/ Qty Classes
0.082	3.950	0.361	0.079	0.455	0.133	44.045

The point estimate calculated by STREW Reliability Model is 91.89%. However, the Nelson model based upon running the black box functional test cases approximated actual reliability at 71%. We believe this is an anomaly due to the fact that we build the STREW model with one data from 18 student projects and used the model to predict reliability of a different set of programmers. Historical values of STREW metrics are dependant upon programming and testing style. The project of group A is generally well organized with clear package hierarchy. Essentially, the test files contained one assertion for every test method in the test files. However, three out of seven metrics, R1, R4, and R6 display in red which means more test cases and assertions should be written to bring the values of the ratios more in line with successful projects.

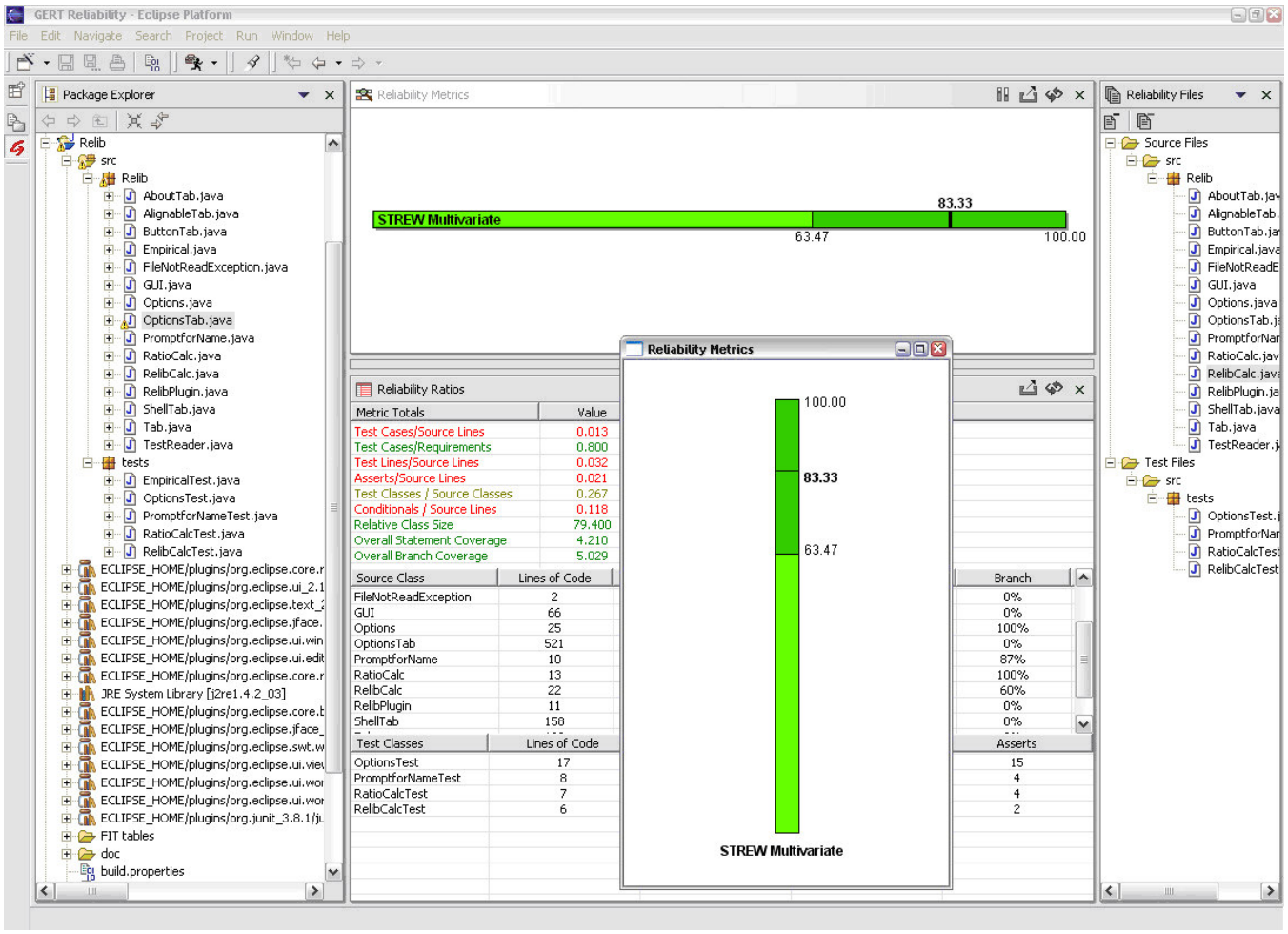

Figure 4: GERT screen shot for group N

Table 5: Metric Values for Group N (SLOC = Source Lines of Code)

Qty Tst/ SLOC	Qty Tst/ Qty Req	Test LOC/ SLOC	Qty Assert/ SLOC	Qty Tst Class/ Qty Src Class	Qty Cond/ SLOC	SLOC/ Qty Classes
0.013	0.800	0.032	0.021	0.267	0.118	79.400

Prior to examining the programs written by group N, we felt the reliability estimate would suffer because the test files in this project were not well written. There are only four test files and each of them has a very limited number of test methods. For example, the test file with the most test methods, which is OptionsTest.java, has only six test methods. Also, the numbers of assertions is quite small in the test files. The results given by the tool are consistent with what we intuitively felt they should be. Only the number of test cases/number of requirements (R2) and the number of lines of code/number of classes (R7) display in green. All other metrics display in red. The project has too few test cases and assertions to do well in the STREW model. As a result, the point estimate calculated by STREW Reliability Model, 82.19 is much lower than that of the other student projects. Additionally, the confidence interval of the model is much larger compared with the other projects.

This screen shot also displays the optional vertical orientation of the Reliability Metrics View. This view can be copied to reports on the reliability estimates of the project.

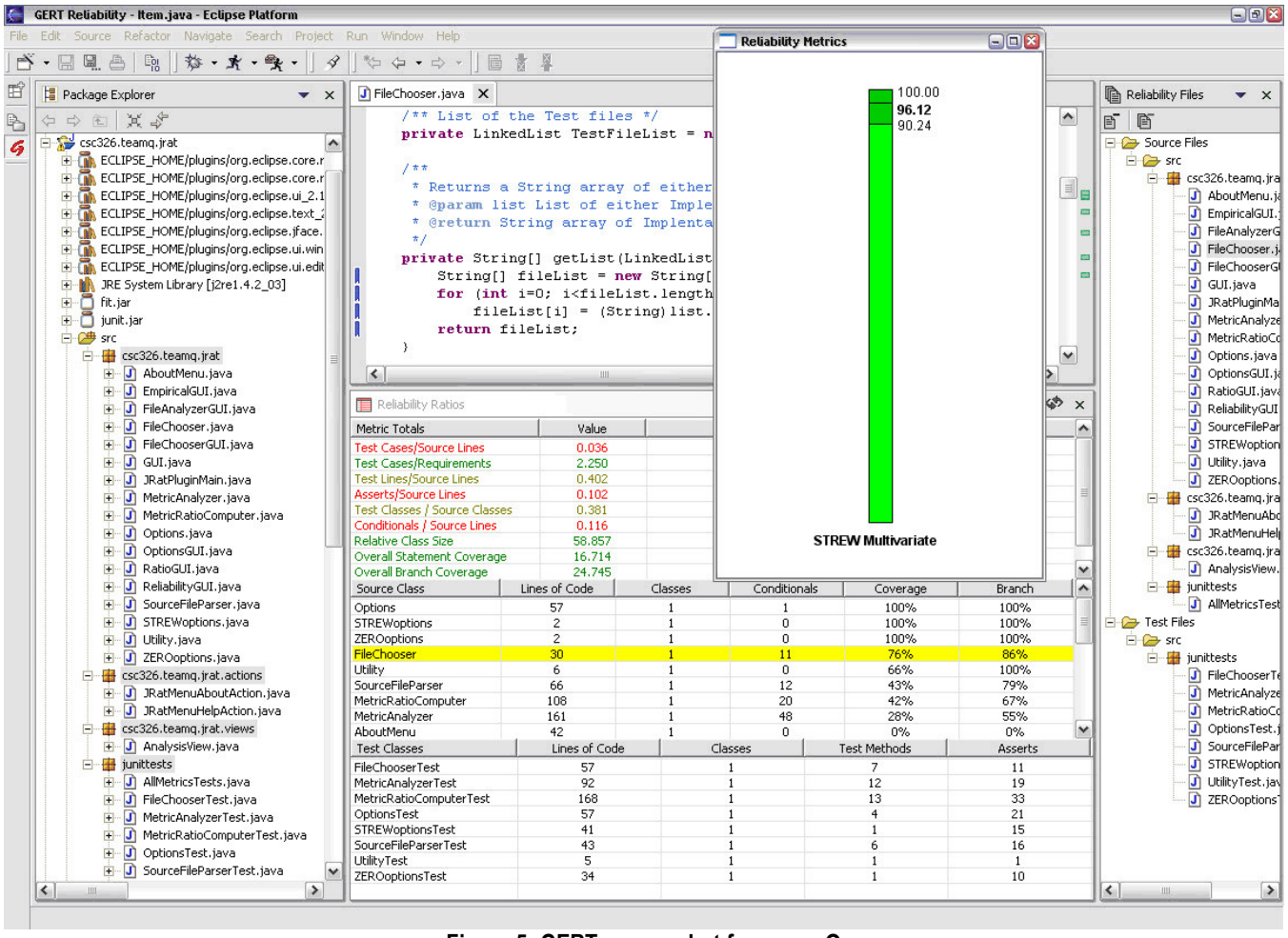

Figure 5: GERT screen shot for group Q

Table 6: Metric Values Group Q (SLOC = Source Lines of Code)

Qty Tst/ SLOC	Qty Tst/ Qty Req	Tst LOC/ SLOC	Qty Assert/ SLOC	Qty Tst Class/ Qty Src Class	Qty Cond/ SLOC	SLOC/ Qty Classes
0.036	2.250	0.402	0.102	0.381	0.116	58.857

The STREW reliability estimate (96%) is very close to the Nelson-calculated actual reliability (97%). Our examination of the project is that it is well organized. But unlike the project of Group A, several assertions are included in teach test method. Like the project of group A, the values of R1, R4, and R6 display in red. More test cases and assertions should be written to bring the values of the ratios up. The values of R3 and R5 display in yellow whereas R2 and R7 display in green.

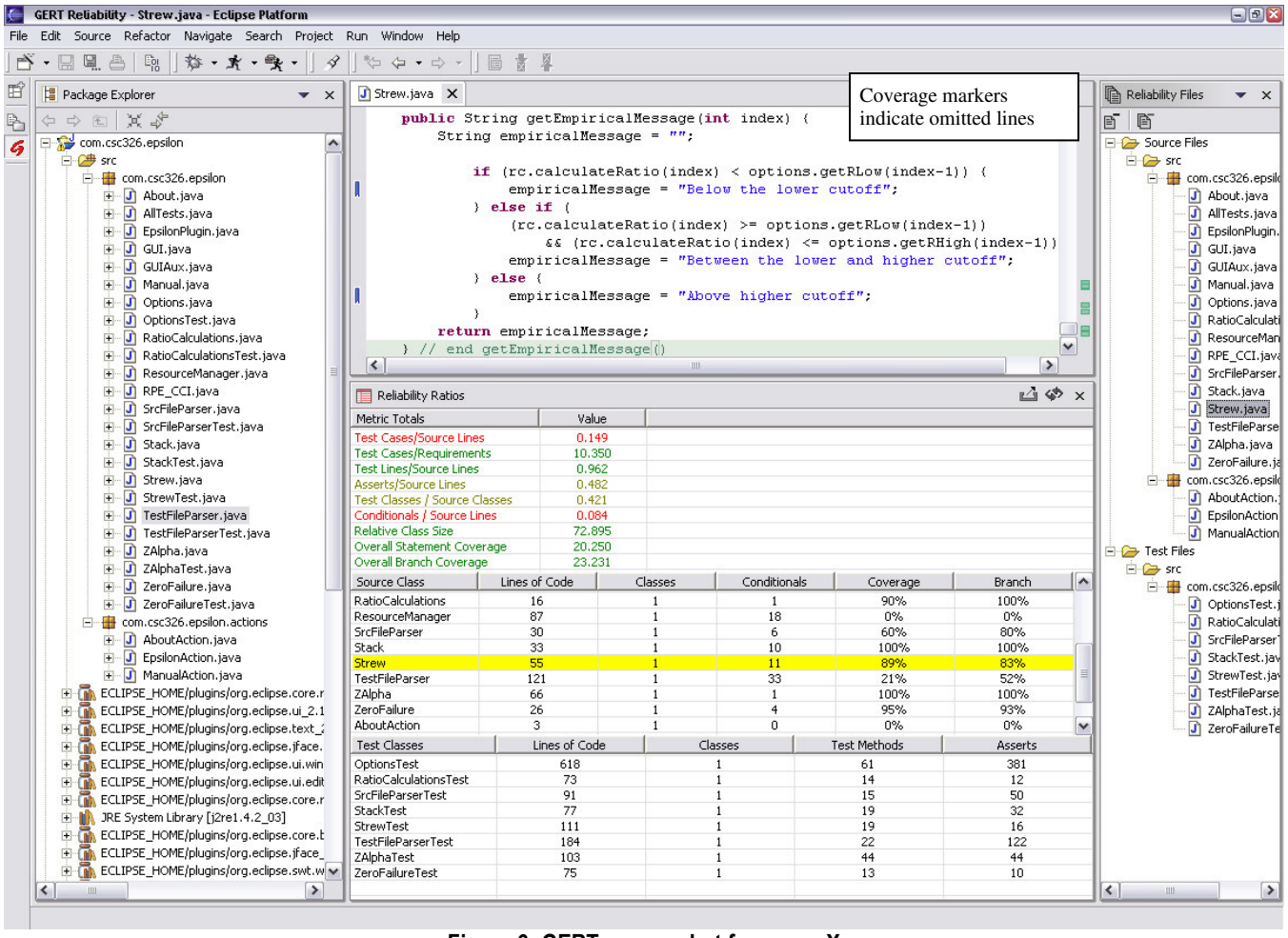

Figure 6: GERT screen shot for group X

Table 7: Metric Values for Group X (SLOC = Source Lines of Code)

Qty Tst/ SLOC	Qty Tst/ Qty Req	Tst LOC/ SLOC	Qty Assert/ SLOC	Qty Tst Class/ Qty Src Class	Qty Cond/ SLOC	SLOC/ Qty Classes
0.149	10.350	0.962	0.482	0.421	0.084	72.895

Among the four projects that we examined, the test files in this project were the best written. The project contains a larger number of test methods and many assertions in each test method. The numbers in the test file table of the "Reliability Ratios" view substantiate our intuition. As we can see, there are a total of 207 test cases and 667 assertions in eight test files, much more than those of the other projects we had examined above. Consequently, the color-coded feedback shows that the value of the number of assertions/source lines of code (R4) is yellow, while this metric in all the other projects all display in red. In addition, R5 displays in yellow as well. Two out of seven metrics, R1 and R6, display in red. R2, R3, and R7 display in green. GERT shows very high reliability point estimate, 100% by STREW Reliability Model. The high STREW reliability may be explained by the unusually high number of assertions, which was uncharacteristic of other class members (whose projects were used to build the model) and might have caused model to overestimate reliability. The actual reliability was 93%.

Referring to the screen shot, Strew.java has been selected. Its row in the metrics table is highlighted, and its source is displayed in the editor window at the top of the screen with the results of running the coverage module. The screen shot indicates that the test cases did not execute the first or last clause of the if-else if-else statement.