USING SOFTWARE QUALITY ANALYSIS TOOLS IN THE DEVELOPMENT OF AN EQUIPMENT ACCESS LIBRARY

C.-H. SICARD, O. WALTER

PS Division, CERN, CH-1211 Geneva 23, Switzerland

ABSTRACT

In the process of re-engineering a critical code library, central to the CERN PS equipment access mechanism, a principal objective was to guarantee the best level of confidence for software quality and reliability. This paper presents the experience gathered in using modern software quality tools for this purpose. Test coverage completeness measures have been used to guarantee exhaustive test cases. Metrics for software defects found during the test, integration and operational phases are discussed.

The equipment access library is a central piece of software for the equipment controls and one of their lowest layers. Since all the applications of the upper layers are based on it, it has to be particularly reliable. While developping the applications, bugs linked to the library layer are difficult to identify. Therefore, it is necessary to ensure the best quality level within the development phase. Assessment of software quality is essential to meet this goal.

1. DEVELOPMENT CONSTRAINTS AND OBJECTIVES OF THE QUALITY CONTROL

1.1 Characteristics of an equipment access library

Control software is quite complex. It relies on many components, and especially on the equipment access library, which belongs to one of its lowest layers [Ref. 1]. The current version of the equipment access library amounts to 13500 lines and 2300 statements. The library is used by upper layer applications as well as by real-time tasks located on the control computers (front-end). Different levels of call are possible : high and low level interfaces are provided (Fig. 1).

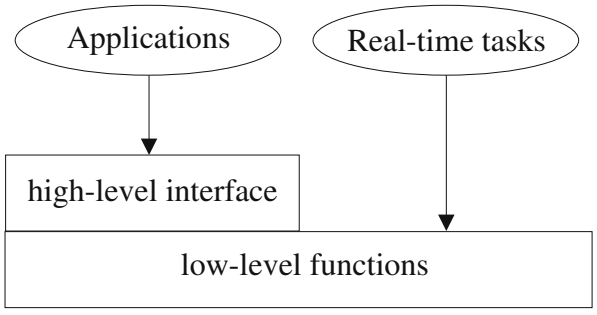

Figure 1. Interface levels for the equipment access library.

The equipment access library has also quite a long lifetime since it is used in many applications that do not change every year. Some upgrades of the library may be necessary during the operational period for maintenance reasons. Such changes have to be done with very short delays. This is why a quick and complete validating procedure is essential.

One of the main characteristics of the equipment access library is that it is data-driven. The descriptions of equipment classes are stored in a database which generates a data-table to be linked with the applications. A list of equipment classes is associated with each front-end computer.

1.2 Testing the library

According to these characteristics, a testing policy had to be designed. For the high-level interface, test case generation is simplified by the fact that a single calling sequence allows one to access a great number of functions (called properties). For the low-level interface, specific test procedures have to be designed for each function or group of related functions. The tests themselves can be data-driven with test data associated to each procedure. Furthermore, specific test classes have to be defined in order to test the new features of the library.

Thus, the conception of the tests implies the definition of test modules at the library side and the definition of test case data and test procedures at the test program side (Fig. 2).

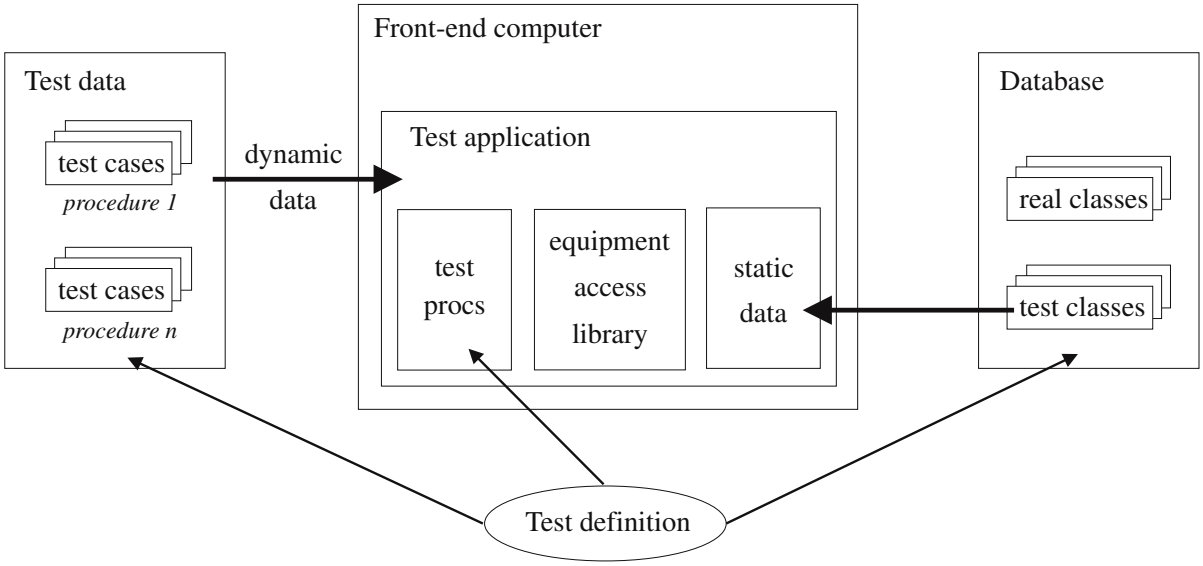

Figure 2. Testing the equipment access library.

The test suites may be used in two situations : while developping a new version of the library, or after a quick change for maintenance reasons. The whole functioning of the library is to be checked.

In the development phase, implementing new specifications will affect the test case suite. Since the compatibility is maintained with the previous version, at least for some time, the frame of the test programs should not change and some test cases are to be maintained to validate the the compatibility functions.

In the operational phase, the test cases used for validating the library are used again to check possible side-effects of the changes. Only minor changes are acceptable while the library is in operation. So the test cases do not have to and should not be updated, since the library has to conform to the specifications defined for the current operation period.

The test cycles with the development and maintenance phases are described below (Fig. 3).

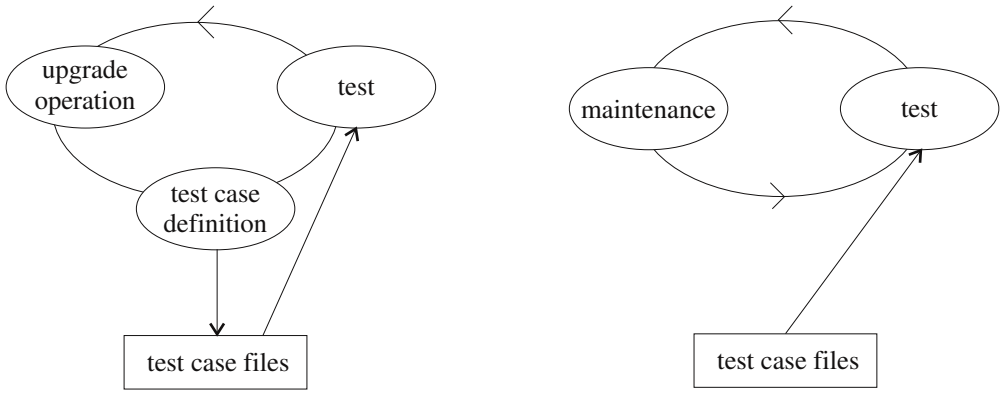

Figure 3. Test cycles for the equipment access library.

The validation of the specifications is the main criterion for software quality assessment. Using software quality analysis tools, the quality of the validation tests can be assessed. Through test coverage analysis, the completeness of the tests is measured, pointing out what portion of the code has been tested or not. A good quality level can be ensured if the tests performed successfully and if the coverage rate is close to 100%.

The other criteria for software quality assessment concern the implementation (efficiency of the code, maintainability, readability, simplicity, self descriptiveness, etc.). Through test coverage analysis, quality control allows one to detect unused portions of code, rare errors cases or redundancies. Through static analysis of the code, assessment of software quality can be performed with some metrics. Thus, general software improvement is possible.

2 METHODS AND TOOLS

2.1 Conception of the test programs

Two test programs have been developped. The first one uses the high-level interface. The test cases consist in the definition of the parameters for the entry point. The second test program uses the low-level interface. The test cases consist of specific procedures for each entry point with their corresponding data.

With this architecture, the maintenance operations on the library can be checked quickly. The test cases definitions used for the released version are applied to the upgraded version. If the tests complete successfully, it can be released. Since only minor changes are performed, the completeness of the tests is assumed sufficient.

The validation of a newly developped version of the library requires one to consider again the test cases, since the specifications have changed. If the low-level interface changed, new test procedures have to be developed. A sufficient quality level - completeness - for the tests should be ensured. The quality assessment is done by the tool, as explained in the next section.

2.2 Assessment of the tests

The quality of the tests can be measured by coverage rates on the library's code. First of all, the tests should give the expected results (in accordance with the specifications). Then, they should cover all portions of code. If not, some further tests may be required.

Different coverage rates are available with the tool used at CERN [Ref. 2]. The components of the library can be analyzed as a set of instruction blocks (IB), decision-to-decision paths (DDP) or linear code sequences and jumps (LCSAJ). The instruction blocks represent the statement sequences of the component. A decision-to-decision path is a set of statements between two control instructions. A linear code sequence and jump is a sequential portion of code that ends with a jump. It is characterized by a departure point, an arrival point corresponding to the jump and the destination point of the jump.

The IB coverage analysis is sufficient only for trivial applications. For the equipment access library, a DDP coverage rate close to 100% is required. Note that some LCSAJs may never be covered if the component has several control instructions with incompatible conditions. However, a LCSAJ coverage rate as high as possible is to be obtained.

The tool can trace the execution of the test programs and analyzes the results. An archive mechanism can gather the results of different executions. With an appropriate set-up, the execution results of the two test programs have been stored in the same archive, so that the tool is able to give a synthetical view of the analysis results. For each component, the IB, DDP and LCSAJ coverage can be given. The coverage distribution over all components can also be displayed.

2.3 A cycle to improve quality

The cycle used for quality control of a new release of the library is made up of three steps : (1) static analysis of the software, (2) conception of the test cases, and (3) assessment of test coverage. The objectives are to validate the specifications (through test programs) and to improve the software quality (see previous chapter).

In the first step, the different conditions and parameters for having access to all entry points are identified. When the cycle is first started, the specifications of the library are the start point for this operation. Then, the quality tool can be used on the components. The list of the decision-to-decision paths give a quite clear view of the different calling conditions for each component. For instance, in the code of [Fig. 4], two DDPs are identified by the tool. Different treaments are performed if the parameter is more or less than 1. The specification may only give the expected result of the function without mentioning how the result is obtained. Therefore, the function should be tested in each condition.

int entry_point(int nbels) {
  if (nbels > 1)
  { ... }
  else
  { ... }
}
	DDP:
	  [1] (nbels > 1)
	  [2] NOT (nbels > 1)

Figure 4. Decision-to-decision paths for a simple function.

In the second step, the test cases are designed in view of the previous results. This means that procedures, equipment modules and data are defined to test the library in the conditions that have been identified at step one.

The third step is the execution of the tests on the library. The tool traces the execution and generates result files. These files are archived, as explained in the previous section. The dynamic analysis results allow to have a global view of the test coverage and also a closer view on each component. The list of the DDP paths that have not been executed in a component can be retrieved easily, and then help improving the results when restarting step one.

Note that the analysis of the library by hand is still necessary since one cannot have access directly to all components, so that some conditions may never be fulfilled. When such conditions are detected, no further tests may be needed for some components. The quality report done on at the end of the development should list these particular conditions.

Some other criteria can be used to assess the improvement of the software. The quality tool can generate a quality report based on several quality metrics and on a quality model. Since the definition of the model is quite subjective and should rely on a large experience, only the comparison of the quality report for the new and the previous library versions can measure the possible quality improvement.

3. RESULTS

3.1 Difficulties

While designing the tests, it appeared that some error cases were difficult to create. For instance, simulating data-table corruption was not possible in most cases. Indeed, it would have require to write a piece of code nearly as long as the function to test, to corrupt the value retrieved in the data-table. Yet, these error cases do not represent the main part of the functions and the performed actions are simple, so that the verification of these parts of the code can be done by hand. The only restriction is that the synthesis cannot be done in the tool. The identified error cases have to be listed apart from the quality report.

Another difficulty was that the tool could only be run on a workstation and not on the target platform. The static analysis and the generation of the instrumented code could be done on the workstation, but the resources required for executing the instrumented code were not available on the front-end computer. Yet, the library was designed to be portable and the execution environment was available on workstations, allowing test quality assessment.

The complete coverage of the library's code was not possible for different reasons. First, they are incompatible conditions in the code, which prevent all LCSAJs to be tested. For instance, two successive <if> statements testing the same variable. Secondly, some error cases have not been tested. The LCSAJ coverage rate is likely to be low for big functions. A multiplication effect is to be noticed. When an instruction block or a decision-to-decision path has not been tested, many LCSAJs may not have been tested. Apart from these consideration, the tool proved to be quite slow when computing LCSAJ coverage rates. On a day-to-day basis, only the IB and DDP analysis was possible.

3.2 Improvement?

Between version 3.2 and 4.0 of the equipment access library [Ref. 3], the evolution of the complexity can be measured by the number of lines and statements (Fig. 5). A decrease of complexity can be observed between the two consecutive versions. The first reason is a simplification of some access methods. It is also the result of the removal of old portions of code and the improvement of the algorithms, which are directly connected to the use of the software quality analysis tools. Some new functionalities have also been added (more data-types handled, new class variables access methods, etc.). The overall improvement of software quality allowed a decrease in complexity while adding new functionalities.

	v 3.2	v 4.0
Number of lines	17000	13500
Number of statements	2700	2300

Figure 5. Evolution of the library complexity

The quality reports provided by the static analysis tool allow the comparison of the quality for the two versions. The components are classified in different categories in view of a quality model. The categories correspond to different quality levels obtained from quality metrics compared to reference values. Yet, as explained in chapter 2, the reference values are very particular to each kind of software, so that only a relative comparison between two versions may be relevant.

The quality reports for the versions 3.2 and 4.0 of the equipment access library are given in Fig. 6. Note that the percentage of components to be inspected, tested and rewritten may seem high (above 10% for version 3.2 and 9% for version 4.0). Actually, tests are performed on all the components and the dynamic analysis allowed to validate the completeness of the tests, with the restrictions listed above. So these results should not be taken as absolute - especially again because of the difficulty to have good reference values for the metrics.

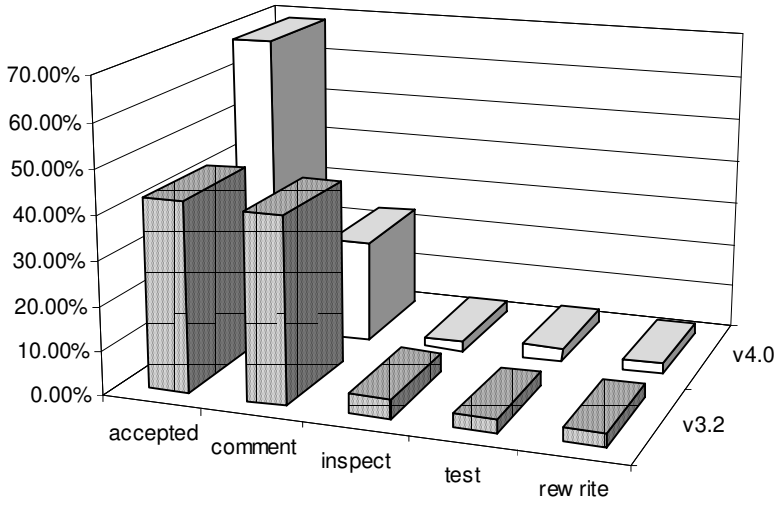

Figure 6. Quality reports for the equipment access library (v3.2 and 4.0).

The test coverage assessment allowed the detection of some unused portions of code corresponding to previous versions. While developping the tests, some bugs have also been found. The test cycle allowed one to discover errors in the specifications, uncomplete specifications and also coding errors. The great benefit of the test quality assessment is to ensure that nearly all the code has been tested and is error-free.

For the equipment access library, the method has given quite good results. See (Fig. 7) for the DDP and LCSAJ test coverage distribution.

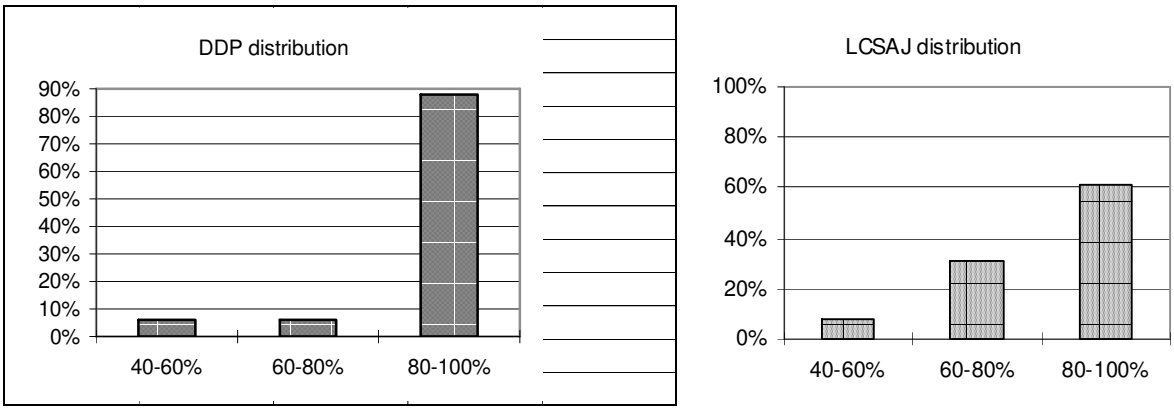

Figure 7. DDP and LCSAJ test coverage distribution (version 4.0).

4. ANALYSIS OF DEFECTS FOUND DURING TESTS

The systematic use of a source control tool, together with an error reporting system, allows a thorough analysis of the defects found during this development. Two main results from this analysis are proposed here.

4.1 Defects classification

Defects have been classified in several types (fig.8), each of which relates to a part of the development process, from design to documentation [Ref. 4].

This classification allows to evaluate what parts of the development process need improvement; in this particular case, we found that Function type defects are more important than expected during this development phase, indicating the need of more careful design validation procedures.

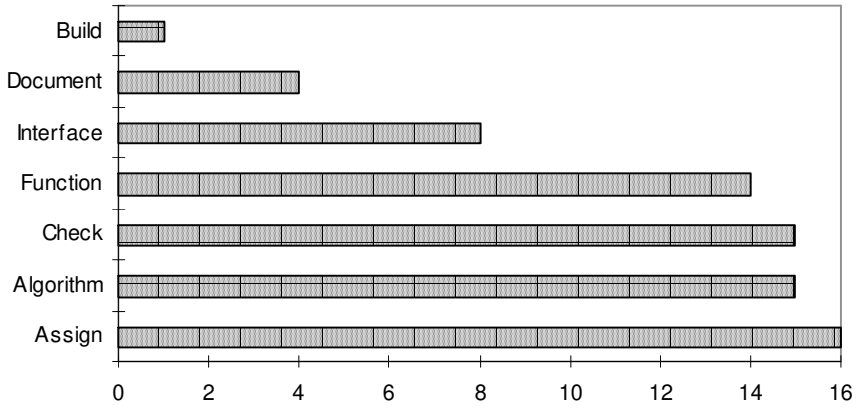

Figure 8.Defect type distribution

4.2 Error rate metrics

Metrics estimating the number of errors to be found in a given piece of code have been proposed by several authors. We applied such a metric [Ref. 5] and found a good correlation factor (0.65) between estimated and actual error rates among the 30 source files making this library.

We believe such metrics can be useful to point out the modules which, by showing a low rate of defects found compared to expectations, may need more thorough testing.

CONCLUSION

The use of software quality analysis tools allowed us to improve the development and to validate the test environment for the equipment access library. A minimal level of management of the software development process (source code control, fault history) is needed to evaluate the quality improvement obtained. Software quality assessment is to be recommended in the development process to ensure its reliability.

REFERENCES

[1] L.Casalegno, J.Cuperus and C.H.Sicard, "Process Equipment Data organisation in CERN PS Controls", ICALEPCS 89, Vancouver, BC, Canada, Oct 30 - Nov 3, 1989, Nucl. Instr. And Meth. A293 (1990) 412.

[2] Logiscope Release 3.3 Basic concepts, Verilog S.A. (August 1993).

[3] C.H.Sicard, J.Cuperus, O.Walter, "Control Module Handbook, version 4", CERN PS/CO Note 95-01.

[4] R.Chillarege, I.Bhandari, J.Chaar, ``Orthogonal Defect Classification - a concept for in-process measurements", IEEE Trans. on Software Engineering vol.18 no11, Nov 1992.

[5] M.H.Halstead - Elements of Software science - North Holland, Elsevier 1977.