Code Coverage made Easy

Software tests are an indispensable part of quality assurance. Without sufficient and documented tests, it is impossible to determine whether software is safe and functionally correct.

To check whether the entire code has been tested (i.e. the tests are complete), it is necessary to measure the code coverage. This makes it possible to determine how extensively the software has already been tested and which parts of the code still need to be tested to achieve the specified code coverage.
Code coverage indicates the ratio of tested code to total code. In simple terms, for example, code coverage is 75% if three out of four possible options are passed during the test. Complete code coverage initially means that 100 percent of the specified test level has been achieved.

For “non-critical” applications, it is often not helpful to aim for 100% code coverage. However, it is important to know why certain test cases have been executed and which parts of the code are being tested. Without this information, there is a high risk of redundant tests that repeatedly run through code that has already been tested, as well as the risk that important parts of the code are not checked at all. In non-critical application areas, the tests should therefore be neither “too few” nor “complete”, but “sufficient” in order to achieve a reasonable balance between costs and benefits.

The situation is different for safety-critical software. In the automotive, rail and aviation sectors, faulty software can have catastrophic consequences. In medical technology, patients rely on functioning pacemakers and intelligent insulin pumps. With the increasing criticality of the various devices and the possibility of attacking them from the outside, the demands on software tests are increasing.
Industry standards in safety-critical software development therefore stipulate precise requirements for code coverage. Products cannot be certified here without proof of sufficient test coverage.

Code coverage is therefore mandatory for safety-critical software and also makes sense for less critical software. However, measuring code coverage often seems complex and time-consuming, especially for beginners. For embedded software, there are additional challenges such as limited memory space.
However, if you pay attention to a few basic aspects, code coverage measurement quickly becomes easier and, with the use of a suitable tool, almost child's play.
However, the code coverage analyzers available on the market sometimes differ significantly in terms of handling and quality. The following criteria will help you select a suitable code coverage tool.

Requirements for Code Coverage Tools

Ease of Use
Even the best software is reluctantly (and therefore rarely) used if its utilization is unnecessarily complicated or not well thought out. This applies of course also to code coverage tools. Simple handling, on the other hand, can significantly increase user acceptance of a test coverage tool and help to save time and money. Ideally, the tool runs in the background, is built into a continuous integration process if necessary and does not create any additional work for the user during testing.

Comprehensibility of the Coverage Reports
Code coverage analysis usually generates large amounts of data, which should be processed by the coverage tool in a meaningful and interpretable way.
The information provided must be available both in machine-readable form for further automatic processing (e.g. XML) and in human-readable form (e.g. HTML) for a quick and good overview.

When looking at the coverage reports, the tester or quality manager should be able to see immediately which parts of the code have already been tested and where tests are still missing. With good coverage tools, the tester can easily recognize at source code level which test cases are still outstanding. By executing these missing tests, the code coverage can then be increased in a targeted manner. This also avoids unnecessary work caused by redundant tests.

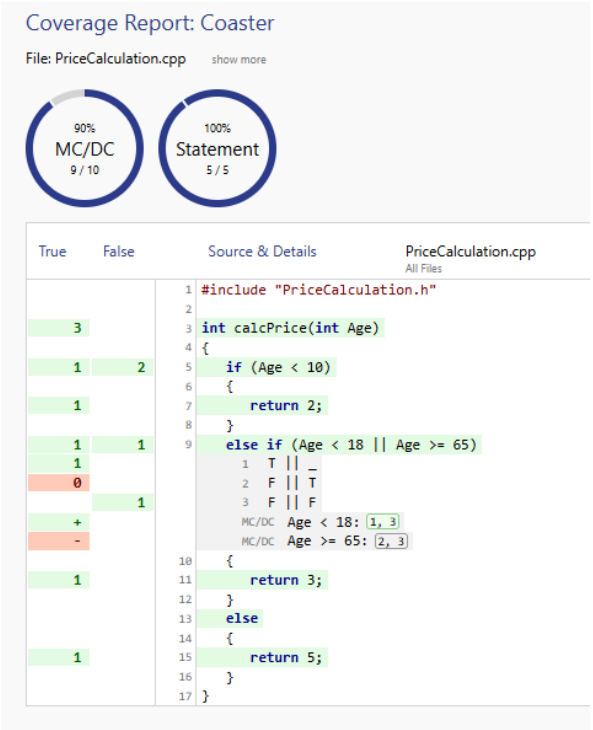

Fig. 1: Detailed information about the code coverage in the source code

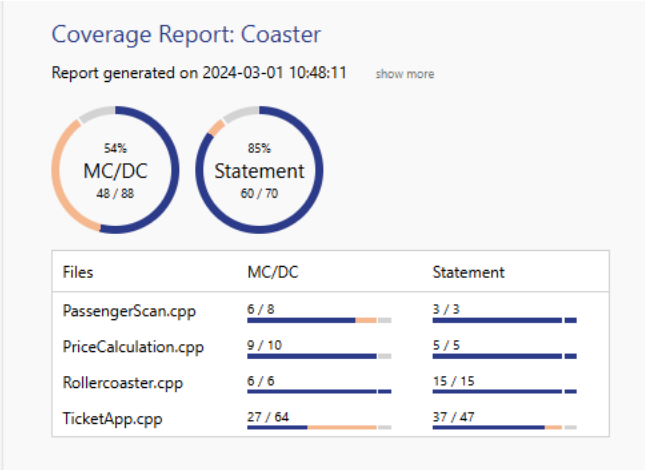

Fig. 2: Overview of the code coverage results

Comprehensible coverage reports are also important in connection with certifications or audits that may be required.

Possibility of Justifications

Often, certain parts of the code cannot be reached by test cases, e.g. due to defensive programming. In this case, the Code Coverage Analyzer should offer the possibility to explain and document missing coverage via justifications. The Code Coverage Analyzer Testwell CTC++ offers the possibility to store justifications via comments directly in the source code or alternatively to document them in files outside the source code. The coverage reports transparently show which parts of the code were tested and which were "explained" via justifications.

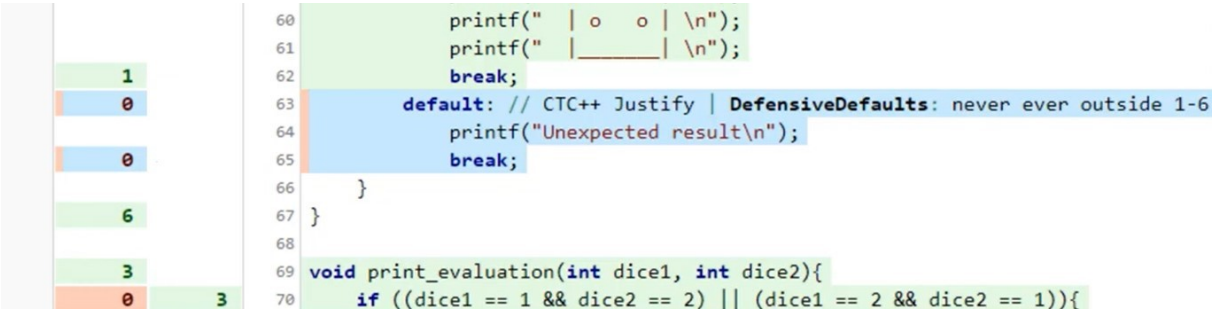

Fig. 3: The justifications stored as comments in the source code appear in blue in the report

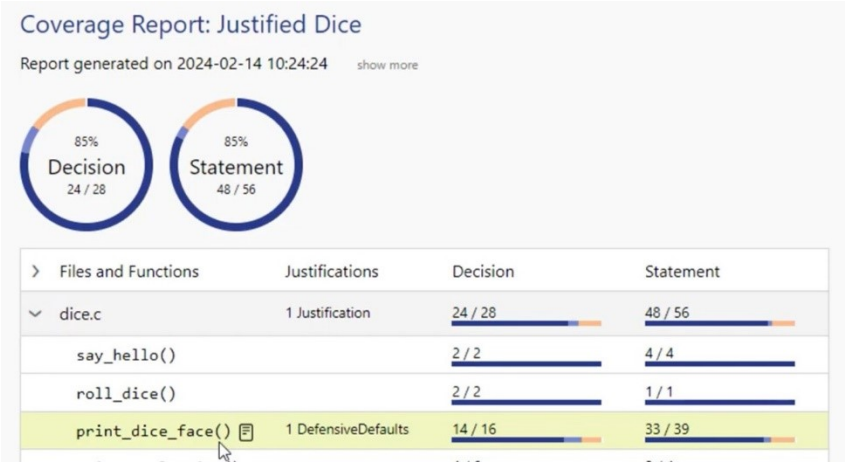

Fig. 4: In the overview, the justifications are highlighted in light blue, the code actually covered in dark blue

Support for "creative" programming styles

Some coverage tools have problems when analyzing language constructs that deviate from the usual standards or have a high nesting depth. However, a good tool for measuring test coverage should also be able to cope with a "creative" programming style.

Independence from Compilers

Of course, a code coverage tool must work with the compiler used in the project. However, it makes a lot of sense to use a tool that can be used independently of the compiler right from the start. Such tools can then be used in all projects, and it is also possible to change the compiler during the project if necessary. A coverage tool that can be used independently of the compiler can be used in many more ways and is therefore a worthwhile investment.

Flexible Integration

Even within a company, development environments and tool chains are often very heterogeneous. In addition, more and more tests are being automated in order to reduce costs and avoid sources of error caused by manual actions.

Automated tests and the automatic recording of code coverage can hardly be dispensed with, especially in large development projects. Automating the test on the build system is particularly important in agile development with continuous integration/continuous deployment (CI/CD). It must therefore be possible to integrate the coverage analyzer into the respective build process and into the execution of the tests seamlessly and without great effort. Tools such as Testwell CTC++, which can also be used via the command line, simplify integration and offer advantages when creating automated builds.

Support of higher Coverage Measures / Suitability for safety-critical Development

For testing of safety-critical software, standards such as ISO 26262 in the automotive sector, DO-178C in aviation and standards in rail transportation stipulate high coverage measures that, depending on the safety level, extend all the way to MC/DC coverage. It is therefore essential to ensure that the selected coverage tool supports the required coverage levels. In order to be able to use a solution in the long term, not only current but also foreseeable future requirements should be taken into account.

It is important to note that many coverage tools only measure decision or branch coverage at most and are therefore insufficient for safety-critical software development.

Depending on the Safety Integrity Level (SIL) for IEC 61508 or Automotive Safety Level (ASIL) for ISO 26262, Statement Coverage, Branch Coverage or MC/DC (Modified Condition/Decision Coverage) is required. The following applies to all standards: the more dangerous the effects of possible faults are, the stricter the degree of coverage required.
This can be seen from the table in the ISO 26262 standard.

The highest safety level ASIL D requires the highest degree of coverage: MC/DC. In the table, ++ stands for highly recommended, + stands for recommended.
Complete MC/DC coverage automatically implies complete branch coverage and complete branch coverage automatically implies complete statement coverage.

Methods		ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

Fig. 5: Requirements of ISO 26262 for Code Coverage for Unit Tests

For software for which there are no regulations in the form of standards, it is also important to determine in advance which code should be tested and according to which criteria. For this, it can be helpful to use the above-mentioned standards as a guide and to align the scope of the tests with the criticality of the system.

Certification and Qualification
Most safety standards also require the entire tool chain to be qualified. The aim here is to prove that both the coverage analyzer and the other tools used within the entire tool chain work properly.
If the tool is certified (e.g. by TÜV), it can generally be used in projects in accordance with IEC 61508, ISO 26262, and IEC 62304 without further qualification measures.
Only the aviation standard DO-178C prescribes a tool qualification. For this, manufacturers of code coverage tools provide support with qualification kits, special test cases and advice on the necessary qualification of the coverage analyzer. In this context, you should also check whether the selected coverage tool is already being used successfully in safety-critical projects.

Special Requirements for the development of embedded Software
There is usually only a small amount of memory available for the software of embedded devices. For cost reasons, the processors also often have limitations. This results in particular challenges with regard to code coverage measurement, which must be taken into account when selecting a coverage analyzer.

Low Instrumentation Overhead

Most coverage tools measure the code coverage via the instrumentation of the source code. The source code is enriched by the coverage tool with “counters” that count where and how often the relevant code parts were executed during testing.

In addition, a library must be implemented on the target to be tested, which, among other things, handles the data transfer to a host. Alos this increases the size of the original code.

When testing on embedded targets that have limited memory, it is therefore important to ensure that this instrumentation overhead is kept to a minimum.

The differences in memory requirements between the individual code coverage tools are sometimes considerable. The Code Coverage Analyzer Testwell CTC++ is very resource-friendly in this respect and offers the option of using smaller counters to significantly reduce memory space. The so-called bit coverage feature reduces the counter size from 32 bits to 16 or 8 bits, but then no longer measures how often a piece of code has been tested, but only whether it has been tested at all. This is sufficient to meet the requirements of the standards.

If the instrumentation overhead is still too high, the code coverage can be analyzed separately for individual code parts (partial instrumentation).

Check possible effects on the Processor

To save costs, not only the memory space of embedded devices is limited - the processors also often have limitations. Unfortunately, the instrumentation has an effect on the processor. In some cases, the defined timing may be exceeded, which can lead to faulty program sequences under certain circumstances.

For this reason, code coverage analyzers should also be evaluated with regard to their impact on timing.

The bit coverage feature mentioned above can also help here.

It also makes sense to perform the tests once with and once without coverage analysis to ensure that the coverage measurement has not led to changes in the program behaviour.

Conclusion

Code coverage is mandatory for safety-critical software development. However, measuring test coverage is also a good method for anyone who wants to improve their software quality in general to measure and increase the quality and validity of software tests. When selecting a code coverage analyzer, care must be taken to ensure that the tool meets the specified requirements. In addition, factors such as ease of use and professional assistance in the event of queries and support play an important role.

The suitability of a coverage tool for your own projects should always be checked as part of a tool evaluation. This will also give you an impression of the performance of technical support. Finally, it is also advisable to take a look at the manufacturer's customer references.

Used correctly, a good test coverage tool helps to significantly improve quality, increase the motivation of developers and testers and carry out tests in a cost-saving manner.

Coverage Levels at a Glance

There are numerous different test coverage levels. In general, the stricter and more detailed a test coverage level is, the greater the effort and cost required to achieve it.

Function Coverage
The function coverage measures whether all functions of the program have been called. This coverage is the “weakest” of the usual test coverage levels. As it ignores the inner workings of the software, its usefulness is quite low.

Statement Coverage
The statement coverage determines which statements were executed by the tests. This can also be used to detect dead code.

Decision Coverage / Branch Coverage
With this level of coverage, each decision must be tested at least once as true and once as false. For normal if statements, this means that every branch must have been executed.

Condition Coverage
The condition coverage considers composite decisions in detail. For decisions that consist of several atomic conditions that are composed of Boolean operators, each of these conditions must be tested individually as “true” and “false”.

Multicondition Coverage and Modified Condition/Decision Coverage (MC/DC)
With multi-condition coverage, all possible true/false combinations for composite decisions must be tested. In the case of multiple conditions within a decision, this usually requires an impractically high number of test cases. Therefore, in practice and in the standards, Modified Condition/Decision Coverage (MC/DC) is relevant, in which the number of test cases is reduced, and the significance of the test coverage remains sufficiently high. All atomic conditions of a composite condition are used for MC/DC. For each of the atomic conditions, a test case pair is tested that leads to a change in the overall result of the composite condition, whereby only the truth value of the atomic condition under consideration changes, while the truth value of the other atomic conditions must remain constant. In practice, this achieves the same purpose as multicondition coverage - but with fewer test cases.

Further Information:

Verifysoft Technology: https://www.verifysoft.com
Testwell CTC++ Code Coverage Analyzer: https://verifysoft.com/en_ctcpp.html

Author

Klaus Lambertz is the founder and CEO of Verifysoft Technology GmbH, a development company for software testing tools based in Offenburg (Germany). Its main product is the Code Coverage Analyzer Testwell CTC++, which is successfully used in numerous safety-critical software projects in 45 countries worldwide.