

Dutch encouragement

Erik van Veenendaal and his fellow consultants Rogier Ammerlaan, Rob Hendriks, Vivian van Gansewinkel, Ron Swinkels and Mark van der Zwan of Improve Quality Services in Holland discuss their practical experiences of using testing standards

It is important to understand that standards are not just theory, but more often reflect industry best practices. Over the years a number of useful testing standards have been produced. They can support you in doing more effective and more efficient testing. We would like to encourage you to look at some of the standards discussed in this paper, and study them to see where and how they can support you. Remember, this paper has been written from a practical background. The standards have made our 'testing life' a little bit easier; now it's up to you!

BS7925-2 Software Component Testing

The BS 7925-2 standard focuses on software component testing. It contains a short guideline for a component test process. A large part of the standard is dedicated to test case design techniques, both black-box (eg equivalence partitioning, state transition testing) and white-box (statement coverage, decision coverage). In the annex, each technique is explained in detail using an elaborate example. Related coverage measures are also provided. Although the standard's name indicates that it can only be used for component testing, the black-box test case design techniques can be applied at almost any test level.

We have often used the standard as an aid to specifying test cases on system level. In particular, the descriptions of equivalence partitioning, state transition testing, boundary value analysis and syntax testing are very useful. The elaborate examples provided by the standard support the implementation process. Many test engineers have by now attended an ISEB Foundation Certificate course, which covers the BS7925-2 standard. This facilitates communication and implementation.

According to BS 7925-2, a model derived from the requirements specification is needed for each item you want to test. A model can be, for example, a state transition diagram, a cause-effect diagram or a table of inputs and equivalence classes. Having a good and complete model is essential for quality testing. To derive test cases from the model, BS7925-2 has often been used. However, in most projects, the test techniques are used slightly differently as described (eg the switch coverage rules of state testing were sometimes not

explicitly followed). The BS 7925-2 techniques are often combined with other test design techniques, eg test use case technique. Use case testing is typically a system test technique and currently not described in BS 7925-2. The standard could be improved by extending it to some informal and system level test design techniques. Overall, the BS 7925-2 standard has proved very helpful during the test design phase. It supports the testers in establishing thorough test designs.

IEEE 829 Test Documentation

IEEE 829 is a standard that describes a set of basic test documents, eg test plan, test case specification, test summary report, applicable for software testing in any environment. For each individual document the standard describes the structure and content. Additionally, an example of each type of test document, including implementation and usage guidelines are provided in the annex. This helps to understand the standard and supports the application.

The standard can be regarded as a useful guideline for test documentation. Since for each type of test document, the purpose, outline and content is provided, applying and implementing the IEEE 829 standard in a particular project and/or test phase is not too difficult. We've also found that the descriptions (purpose and outline) of the documents are highly usable in practice. The tester is still free to decide whether or not to implement all of the documents described in the IEEE 829 standard or even only a part of the document, which is considered relevant for the particular project. The tester can also determine how to apply the documents. The standard does not require a specific test design technique or test tool, and is therefore easy to use without many pre-conditions or environmental needs. Often companies have already created their own set of test documentation standards. In these situations the IEEE 829 standard can be used as a checklist to identify improvements in test documentation. An advantage of applying this IEEE 829 standard is that it covers a broad range of test documents, from test plan and test design specification to the test summary report. When testers within a specific project are all working according to this standard a consistent structure and terminology for all

documents will be achieved. No unnecessary overlap in documentation or inconsistent terminology between the different documents is to be expected, even if the testers have different background and experience, which is often the case within real-life projects. The only test documents missing in this standard are the test policy and test strategy documents. For this we currently used the descriptions provided by the ISEB Practitioner syllabus. So far, we have used the IEEE standard to create useful test plans, test design specifications, test case specifications, test logs and test incident reports in our projects. The common reference has facilitated communication, thus saving time and money.

IEEE 1028 Reviews and Inspections

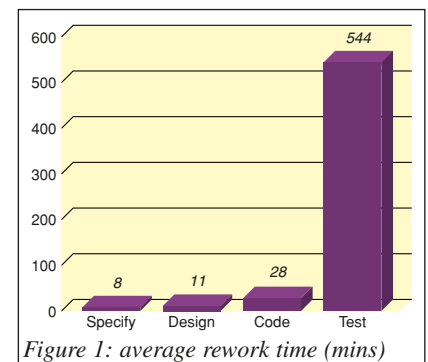


Figure 1: average rework time (mins)

The IEEE 1028 standard for software reviews describes a set of review types and processes applicable throughout the software life cycle. The standard distinguishes three main types of reviews: the technical review, the inspection, and the walkthrough. Per review type the process steps are described on a detailed level. The standard is completed with the description of the software audit process. The annex provides a comparison of the different review types to support the selection process of the appropriate type of review. The use of informal reviews seems to be common practice, but few organisations are applying reviews and inspections to their full extent. In practice review types are often mixed and simply named 'reviews' or 'inspections', resulting in a very time-consuming and ineffective process. A walkthrough, focused on achieving consensus, is something very different from an inspection, focused on formal defect finding. The major advantages of using IEEE 1028 are the clear

definitions and straightforward process steps it provides for the different review types based upon different objectives.

We have implemented reviews in many organisations based upon IEEE 1028. It provided a thorough reference framework and common terminology. It just doesn't make sense to define 'new' review types, where the standard already provides them for you. A disadvantage of the standard is that it does not explicitly address and explain a number of critical success factors for reviews, such as optimal 'checking rate', 'logging rate' and 'number of pages per review'. If more information on implementation is needed, one might refer to books like *Software Inspection* by Tom Gilb and Dorothy Graham and *Walkthroughs, Inspections and Technical Reviews* by Freedman and Weinberg. These books provide numerous tips on how to make IEEE 1028 even more applicable. As stated, we have used the standard to check and complete review and process descriptions within different environments. The common reference has helped us to create guidelines, untangle existing review processes and get the message across that different objectives ask for different review types. Reviewing, when applied correctly and early in the process, remains the most effective and efficient defect detection technique. (See also figure 1 'Average rework time' that shows recent data from one of our clients.)

ISO 9126 Software Quality

Without a solid definition on quality, communicating on this very broad term is useless. The most practical definition on software quality is provided by ISO 9126. The standard describes how software quality can be expressed by means of quality characteristics. This standard distinguishes six main characteristics (functionality, efficiency, usability, reliability, maintainability and portability) and 26 sub-characteristics. In addition to providing a definition framework for software quality, the ISO 9126 standard also provides a number of useful internal and external metrics that can be used by the tester as a source when defining completion criteria.

In practice non-functional quality is often not or only briefly specified in the requirements. This causes a problem for testing, as it has to make a judgement on product quality. ISO 9126 is an excellent checklist to use throughout the risk analysis phase, in order to identify the most important quality characteristics. Subsequently the standard is useful for specifying measurable completion criteria. This makes answering the question "Is the product good enough to release?" more straightforward and less subjective. Once the completion criteria have been defined for the various relevant quality characteristics, it appears to be easy to report on quality progress. The completion

criteria quantify quality in terms of metrics, and can be used during testing to report progress and status, eg a completion criteria for reliability is Mean Time Between Failure (MTBF) at 40 hours and the current MTBF is at 23 hours, thus the product cannot be released. This makes quality tangible and visible. The metrics provided by the standard can be customised for the project. This makes the standard flexible: one doesn't have to use all metrics and can easily add one's own metrics.

We have found the ISO 9126 standard very useful in our test projects. However, this standard should be used during the entire development process, because in each phase of the process quality related actions must be taken. In the requirement phase, quality must be defined. In the subsequent phases, activities must be executed to design and/or measure quality to assure that the product fulfils the targets. At the International Organisation for Standardization (ISO) research is continuing to improve the ISO 9126 standard. Current work is carried out towards improvements on specifications of quality requirements.

TMap (Test Management Approach)

TMap, the Test Management Approach to structured testing is applicable to both lower and higher level testing of software products. TMap provides detailed answers to the what, when, how, where and who questions of testing. To structure the organisation and execution of the test processes, TMap is based on four cornerstones:

- a development process-related life cycle model for the testing activities;
- solid organisational embedding;
- the right resources and infrastructure;
- usable techniques for various testing activities.

In recent years TMap has evolved and is now recognised as the standard for software testing in The Netherlands and Belgium. Most Dutch organisations, especially banks, insurance companies, pensions funds and government departments use TMap. To support the approach, books are available in various languages. TMap is applied most frequently as an approach to high-level testing (system and acceptance testing), and is probably less strong on component, non-functional and static testing. TMap is a generic, yet highly practical approach and consists of an extensive number of supporting techniques (together with BS7925-2 'all' techniques are covered), tools and procedures which may be selected for a specific test project. In the numerous projects where we have applied TMap, it has always been a methodology that fulfilled the needs of these projects, not only because of the availability of a well-documented approach, but

also for the great number of templates and useful tips that are available. On several projects it has been helpful during the definition phase for structured testing. The TMap approach has more or less served as our 'complete guide to high-level software testing' during many assignments at customers' sites.

Testing Maturity Model

The Testing Maturity Model (TMM) framework has been developed by the Illinois Institute of Technology as a guideline for test process improvement and is positioned as a complementary model to the CMM. Just like the CMM, the TMM also uses the concept of maturity levels for process evaluation and improvement. The TMM consists of five maturity levels that reflect a degree of test process maturity. For each maturity level, a number of process areas are defined. A process area is a cluster of related activities within the test process, eg test planning or test training. TMM addresses both static and dynamic testing, and with respect to dynamic testing both low-level and high-level testing are within its scope. The structure of the TMM is partly based on the CMM and the staged version of its successor: the Capability Maturity Model-Integrated (CMM-I). This is a major benefit for organisations that are already familiar with the CMM(I).

Our recent projects have involved supporting test organisations, working on embedded software and technical automation, in achieving TMM level 2. We found that the model is highly usable and focuses on the practical needs of most test projects, eg test planning, risk analysis, test design and incident management. (TMM makes use and references the IEEE testing standards.) Since TMM level 2 also addresses the test policy and test strategy, it helps to involve management in test process improvement. Having test performance indicators derived from business goals is necessary to show the added value of implementing TMM. The papers and book available through the Illinois Institute of Technology, but also the guidelines that we, together with several partners, developed, supported the implementation process and provided concrete and practical

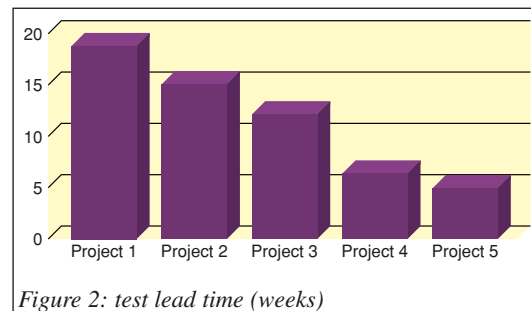


Figure 2: test lead time (weeks)

guidance for test improvement. Figure 2 shows a major reduction in system test lead time, reported by one of our clients who achieved TMM level 2 shortly after completing the final project shown on the graph. **PT**