Information Quality Testing

Anna Wingkvist[1], Morgan Ericsson[2], and Welf Löwe[1] and Rüdiger Lincke[1]

[1] School of Computer Science, Physics, and Mathematics,
Linnaeus University,
Växjö, Sweden
{anna.wingkvist|welf.lowe|rudiger.lincke}@lnu.se
[2] Department of Information Technology,
Uppsala University,
Uppsala, Sweden
morgan.ericsson@it.uu.se

Abstract. When a new system, such as a knowledge management system or a content management system is put into production, both the software and hardware are systematically and thoroughly tested while the main purpose of the system — the information — often lacks systemic testing. In this paper we study how to extend testing approaches from software and hardware development to information engineering. We define an information quality testing procedure based on test cases, and provide tools to support testing as well as the analysis and visualization of data collected during the testing. Further, we present a feasibility study where we applied information quality testing to assess information in a documentation system. The results show promise and have been well received by the companies that participated in the feasibility study.

Keywords: Information quality, Testing, Quality assessment.

1 Introduction

A knowledge management system (KMS) is used to capture and preserve the knowledge of an organization. The knowledge can be facts, figures, rules, procedures, etc. Simple examples of information that a KMS can contain include an organization wide phone book or instructions on how to file a travel expense report. The KMS can be thought of as an instruction manual for the organization.

The quality of the KMS can be defined by its fitness for use. If it is unstable and often crashes, it can be considered to be of poor quality. The same holds if it is difficult or slow to use. If the "knowledge" captured by the KMS is incomplete, hard to understand, or even incorrect, the perceived quality of the KMS will be affected too. The first two sources of quality issues, i.e., the quality of the hardware and software, are well understood and often dealt with in a systemic way. There are distinct phases for testing and maintenance in all software lifecycle models. Information testing on the other hand is not that well-understood and often not considered in a systemic way, but rather in an ad hoc fashion; if

people find an error they will either report it to some information maintainer or correct it themselves.

The purpose of a knowledge management system is to store information and there should be systemic ways to assess the quality of this information. In this paper, we aim to extend testing practices used for quality assurance of hardware and software to also include information. We have previously worked on the quality assessment and assurance of technical documentation (e.g., Wingkvist et al. [11]). Some aspects of information, such as facts and figures, can easily be verified or even automatically validated by the KMS. However, other aspects, such as general information or procedures can be more difficult to test. We rely on procedures from software and usability testing, i.e. create test cases and have testers with access to the KMS attempt to perform a task. We measure how they performed the task, how long it took, what parts of the information of the KMS they used, etc.

The rest of the paper is organized as follows. In Section 2 we introduce testing. We then extend testing to information quality in Section 3, where we introduce information test cases and a procedure for information testing. In Sections 4 and 5 we describe the tools used to test information as well as a study that tests the feasibility of our approach. Section 6 presents related work and Section 7 concludes the paper.

2 Background and Basic Notions

This section defines the notations we use in the paper and gives a brief background on Knowledge Management Systems (KMS), software testing and information quality. Software testing and information quality are discussed in more detail in Section 6.

2.1 Knowledge Management Systems

Throughout this paper we distinguish the information contained in a KMS from the KMS software system. The former will be referred to as the *information*, the latter as the *system*. For the system site, we further distinguish the editing system for creating the information from the production system for presenting the information. For brevity, we refer to the former as the *editor* and latter as the *browser*. In fact, there are systems like DocFactory[3], which we apply in our feasibility study that integrate both editor and browser in one system. For understanding the paper, it is however sufficient to assume that the browser is a standard Web browser.

2.2 Software Testing

Software testing is commonly applied to assure quality of software systems. The principle of testing software is as follows. First, a number of test cases are derived

[3] `http://www.sigmakudos.com/data/services/docfactory/`

from the use case specifications of the software system (which, in turn, is derived from the requirements of the system). Second, the system is tested using the test cases.

Every intended use of the software system is simulated by one or more test cases. Each test case consists of one or more "uses" of (often calls to) functions of the system and a set of expected results. A test case should also include ways to compare the actual results with the expected results.

The system is tested by executing the test cases and comparing the actual result to the expected. If the comparison succeeds (e.g. the actual and expected results are the same) the test case succeeds. If all the test cases of the test suite succeed, the testing is successful.

Step one, the definition of the test cases, is done manually as part of the system development. Step two, the testing, is to a large extent done automatically as part of the build process.

The quality of the test suite has an impact on the quality of testing and ultimately on the quality of the system under development: the test cases should (i) cover all use cases and (ii) cover the whole source code of the system. Unfortunately, for theoretical reasons and because of resource restrictions (such as time), complete testing of a system (covering all use cases and the whole code of the system) is not possible in general. Hence, testing can be considered an optimization process, where the goal is to maximize (i) and (ii) using limited resources.

2.3 Information Quality

High information quality requires both:

- *Suitability for the intended use* including properties like relevance, completeness, correctness, i.e., correspondence to the world described, understandability, non-ambiguousness, accessibility, suitably of presentation, and
- *Suitability for the production/maintenance* (i.e. the inteded use of the information by the owner) including properties like appropriateness of meta-data (e.g., data types, database rules, schemata, conventions), conformance of data to meta-data, non-redundancy of information, etc.

The former implies high value of the information for a user while the latter implies low information production/maintenance costs for the owner. The latter may be enforced by including analysis and restrictions in the editing systems [11]. The former is hard to guarantee by such means.

3 Information Quality Testing

One way to assure the quality of a software system is to test it and compare how it performs with respect to a set of requirements and use cases. The same idea can be applied to information quality. In this section we introduce a method of *information testing* by adopting software systems testing.

In order to test the information of a Knowledge Management System (KMS), we assume that a number of use cases have been defined. We further assume that the system is of high quality (assured by extensive software systems testing).

We begin by adopting the notions of test suites and test cases. *Information test cases* are similar to software test cases, but focus on the quality of the information, i.e. how suitable it is for its intended use. We then adopt the notion of testing, i.e. procedures for carrying out the tests included in an *information test suite*. Finally, we adopt the notions of test quality and test coverage.

3.1 Information Test Cases

Test cases for information define a usage scenario of the KMS together with the expected result. Since information test cases are designed to test the suitability for the intended use by humans, test users should be involved. As common goal of a KMS is to make the knowledge of an organization or a community manageable more efficiently than solutions without system support, a measure of efficiency should also be involved. More precisely, an information test case defines:

(1) A set of test users of the KMS representing a general class of users for that use case,
(2) A set of questions representing questions that this class of users ought to get answered by the KMS,
(3) The set of correct answers expected,
(4) The expected time needed to find the correct answer to the question, and
(5) Optionally, the intended interactions with the KMS.

Consider a company KMS, for example an intranet. One function of the KMS is to provide searchable contact information, such as e-mail addresses and telephone numbers for all the employees. A use case defined for the KMS would specify that given some information, a user should be able to search the information and find the correct e-mail address and/or phone number. Information test cases defined for this use case would assure the correctness of the lookup and search functions, as well as the information returned by them. An information test case might be to find the e-mail addresses and phone numbers of a number of employees within a certain time limit, for example:

(1) Set of users: a sample of 10 employees of the company serving as test persons.
(2) Question: What are the e-mails and telephone numbers of three concrete employees A, B, and C?
(3) Answers: e-mails and telephone numbers of A, B, and C.
(4) Time: 5 minutes.
(5) Intended interaction: enter name A in the search field, push search button, follow the first link returned form the search engine; enter name B in the search field, etc.

An information test suite is a set of information test cases.

3.2 Information Testing

Assume we have defined an information test suite. For each information test case, there is a description of the users, the question the users should answer as well as the expected answer(s). There is also a time limit and an intended interaction with the system.

In order to perform a information test case, a set of users that match the criteria are each confronted with the question. The question should be solved by using the KMS. The given answers and response times are compared with the expected values. Optionally, the browser activity can be monitored during the interaction with the KMS and the interactions of the users can be compared to the expected interaction.

An information test case succeeds to 100% if all test users give the correct answer to the question in time (optionally, using the expected interaction steps). An information test suite succeeds to 100% if all its information test cases succeeded to 100%. Gradual success can be defined for each test user who exceeds the time bounds or, optionally, deviates from the expected interaction. Gradual success for an information test case can be defined based on the success, gradual success, and failure, respectively, of the test users involved. Accordingly, gradual success for the whole information test suite can be defined.

While the test users must execute the information test case manually, the (gradual) success can then be computed automatically. A comparison of times of an actual interaction and the expected response can be done automatically. The comparison of the actual and expected answers and, if given, interactions of a test user deserve some explanations.

Comparing the given answers with the expected is trivial for simple answers (like an employee's contact information in our example). For more complex questions, the test designer can define multiple-choice questions (with one correct answer), which can be evaluated automatically. If the KMS is not only aiming to provide knowledge but also for skills, the test designer can define a task instead of a question and observe the execution of that task by the test users. For some tasks, success can be determined automatically. For instance, the task of sending e-mail to a specified address can be assessed automatically, while the task of fixing an engine correctly (probably) cannot.

Comparing the actual with the expected interactions of a test user is not trivial. When a test user interacts (inputs or mouse clicks, for example) with the browser of a KMS, the internal state of the KMS changes and output might be generated. Hence, user interaction can be defined as a *trace*, i.e. a sequence of input, output and states of the browser. Such a sequence can be used to specify the expectation and to monitor the actual interactions of a test user. In order to automatically test interaction, there is a need to collect and compare such traces.

3.3 Quality of Information Testing

The information test suite should contain all information test cases, and information testing should test the full information test suite. Further, the information

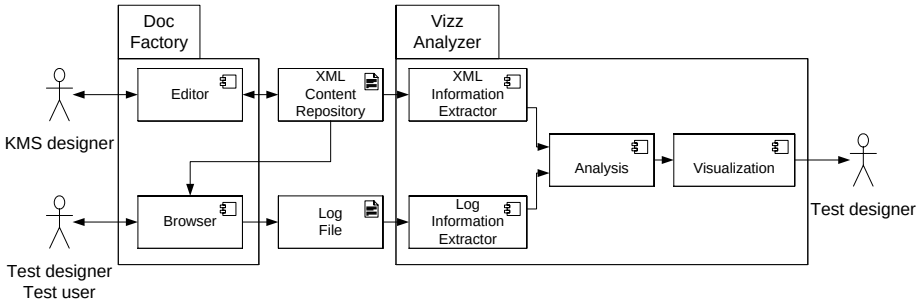


Fig. 1. VizzAnalyzer analyses and visualizes content and usage information of a DocFactory KMS.

test suite should have coverage of the complete KMS. The former can only be assessed manually by assuring that any use case is covered by one or more information test cases. The latter cases covered by one or more test cases. The latter can be assessed automatically using the usage traces monitored in the browser and comparing them to the static structure of the KMS.

Consider the example of find e-mail addresses and telephone numbers of employees. The information test case tests the search functionality and the usefulness of the information returned by the search. From this perspective, the test is complete. However, it does not test the e-mail addresses and telephone numbers of all the employees of the company. From this perspective, the test is not complete, and in order to make it complete, the test users would need to search for every employee.

4 Implementation

In the previous section, we claim that both the user interaction and the system coverage can be automatically monitored and compared with the expectations. This section describes a prototype implementation that confirms these claims.

The prototype combines the VizzAnalyzer analysis and visualization tool [6] with the DocFactory content management system. VizzAnalyzer analyzes and visualizes information captured in DocFactory, cf. Figure 1.

More precisely, DocFactory consists of an editor for creating and modifying content and a browser for viewing it. The content is captured in an XML repository. While browsing the content, DocFactory creates a log of the documents visited and the hyperlinks followed.

VizzAnalyzer begins by analyzing the static structure of the information, which results in a graph structure with individual documents (nodes) and their hierarchy and hyperlinks (edges). VizzAnalyzer then analyzes the dynamic access traces from the logs and adds them to the static structure. This is possible since the logs contain the same identifiers of documents and links as the XML repository. Then VizzAnalyzer compares the traces. Optionally, it visualizes the

structure (graph drawing) and the traces or trace differences by highlighting parts of the structure, cf. Figure 2 for two alternative visualizations.

Using the prototype, we get support for defining the expected traces: The test designer just "clicks" through the system demonstrating the intended use of the KMS. VizzAnalyzer captures the trace logged by DocFactory. Further, we get support for comparing actual and intended use: the test user tries to answer the question and the interactions are captured by DocFactory. VizzAnalyzer is then used to compare the logged trace to the intended trace that is included in the test case.

Finally, our protoype allows us to assess the quality of information testing by performing a system coverage analysis that relates (analytically or visually) the traces logged durign testing with all possible traces of the KMS.

5 Feasibility Study

In order to test the prototype implementation described in the previous section, we performed a feasibility study. The objective of the study was to test an early version of the documentation of a mobile phone. To this end, a technical documentation and a KMS do not differ that much: both are structured information repositories for gaining some knowledge (about a technical device and an organization, respectively).

The "developers" working on the documentation had a structure in mind when they created it, and they wanted to see how well it worked, i.e., its suitability for the production/maintenance. This was analyzed automatically [11]. The prototype documentation consists of 70 XML documents. These documents were analyzed by VizzAnalyzer to find the physical structure of the documentation, i.e., how the different documents reference each other.

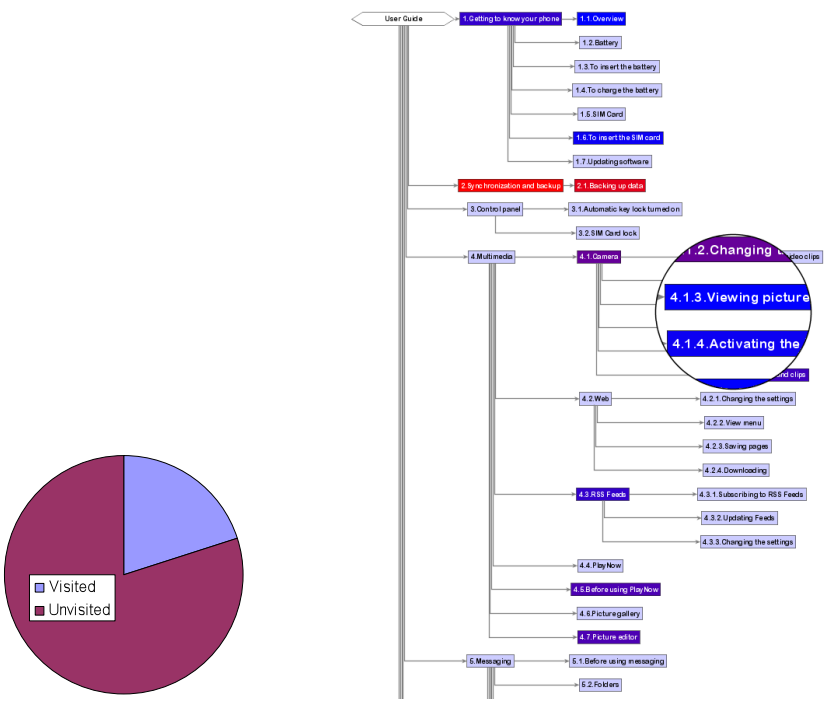
5.1 Information Test Cases

In order to test the documentation, we (informally) defined two information test cases. Test case (A) concerned adding contact details to the address book of the phone. Test case (B) concerned taking and sending photos. The test cases can formally be defined as follows:

(A1) Set of users: the authors of this paper.
(A2) Question: How to save address and phone number information?
(A3) Task: do it.
(A4) Time: 30 seconds.
(A5) Intended interaction: navigate directly to Document 2.1.

(B1) Set of users: the authors of this paper.
(B2) Question: How to take a photo and send it as an e-mail?
(B3) Task: do it.
(B4) Time: 1 minute.
(B5) Intended interaction: navigate to Document 4.1.1, then to 4.1.6.

Note that we replace the expected answer with a concrete task (A3, B3) that could be measured automatically using traces.



(a) Total test coverage. (b) Structure and documents visited.

Fig. 2. Two views on different levels of abstractions on the system test coverage.

5.2 Information Testing

During testing, we used the DocFactory to browse the documentation. Each document entry and exist was logged with a time stamp. This log constitutes the dynamic information. When the static structure and the dynamic usage information are combined, they form a picture of how the test users navigated the documentation. The combined information shows, for example, which documents were visited, for how long, in which order, and so on.

This result of testing is depicted by Figure 2(b), which presents a view that shows the interaction behavior during testing. It depicts the document structure (boxes correspond to individual documents) and encodes the time spent on individual documents in different colors: light-blue boxes are documents not visited at all; the color gradient from dark blue to red corresponds to how long the documents were visited (dark blue representing the minimum and red the maximum amount of time). In our test, the color gradient represents the span 1 to 20 seconds.

The (informal) evaluation of the test case A and B revealed the following. First, the test users could accomplish the tasks. Second, the test users exceeded the time limit for test case A. Third, other documents than the the intended

were used in test case B. This includes the fact that Document 4.1.1. was not visited. The test users intuitively understood how to take the photo, but finding information on how to send the photo was not straight forward.

In a real testing situation, one could draw the following conclusion. First, the documentation is by and large suitable for its use. Second, the description for saving the information was not well written (or the time bound was too ambitious). Third, the structure of the sections regarding the handling of the mobile phone as a camera was not ideal.

However, drawing these conclusions requires trust in the quality of testing itself.

5.3 Quality of Information Testing

The result of testing are depicted by Figure 2(a), which presents a high-level view of how much of the total documentation that was visited. During the test, we visited 17 out of the 70 documents, which resulted in a 20% coverage. This means that 80% of the documentation was not visited during this test.

In a real testing situation, one would consider the quality of such testing insufficient due to low coverage of the documentation system. Alternatively, if all intended use cases were actually covered by our information test cases A and B (which is not the case here obviously), one could consider that the documentation system contained redundant and useless information. Figure 2(b) offers hints to which parts of the documentation system should be covered by other test cases (and excluded from the system, respectively).

For a set of 70 XML documents, both static analysis and correlation of static and dynamic analysis are instantaneous. In order to see how well these analyses scale, we analyzed the full content as well. This full system contains 8674 XML documents; the analysis took less than a minute.

6 Related Work

This section presents related work. We begin by discussing how information quality is defined and assessed. We then discuss different approaches to software testing.

6.1 Information Quality

Weinberg [10] states that *"Quality is value to some person"*. This implies that defining quality can take objective as well as subjective views and stakeholders will perceive the same product or service as having different levels of quality. Still, testing is done to measure the quality of a product or a service.

For example, Kahn et al. [4] assign two values of quality: *conforming to specification*, and *meeting or exceeding consumer expectations*. And, in placing these two in relation to product and service they acknowledge that information can be regarded as both.

Information as a product is seen from an engineering perspective where the focus is on the activities needed to put and maintain data in databases. Hence, the process of keeping information up-dated resembles product enhancement. Information as a service is seen from a third party perspective where the focus is on activities when information is produced and consumed. Hence, the process of converting data into information resembles that of providing a service.

Product quality includes dimensions related to product features, and involves the tangible measures of accuracy, completeness, and absent of errors. Service quality includes dimensions related to the service delivery process as well as addressing the intangible measures of ease of manipulation, security and added value of the information to consumers. The two views of quality and the relation to product and service quality are summarized in Table 1.

Table 1. Classification of Quality (adapted from Kahn et al. [4]).

	Conforms to Specifications	**Meets or Exceeds Consumers Expectations**
Product Quality	Sound Information The characteristics of the information supplied meet IQ standards.	Useful Information The information supplied meets information consumers task needs. Text formatting
Service Quality	Dependable Information The process of converting data into information meets standards	Useable Information The process of converting data into information exceeds information consumer needs

Continuing on the analogy provided by Kahn et al. [4], Wang [9] also matched quality issues in product manufacturing of those in information manufacturing. While product manufacturing use raw materials as input and produce physical products, information manufacturing act on raw data to produce information products (cf. Table 2).

Table 2. Product vs. Information Manufacturing (adapted from Wang [9]).

	Product Manufacturing	**Information Manufacturing**
Input	Raw Materials	Raw data
Process	Assembly Line	Information System
Output	Physical Products	Information Products

Alavi and Leidner [1] provide the hierarchy structure between data, information and knowledge stating that data is raw numbers and facts, information is data that has been processed in some manner, and knowledge is authenticated information. Tuomi [8] argues that the hierarchy from data to knowledge is actually the inverse: knowledge must exist before information can be formulated and before data can form information. Tuomi argues that knowledge exists only inside of a person (a knower). Then, for people to share the same understanding of data or information, they must share a certain knowledge base. Another

implication is that systems designed to support knowledge in organizations may not appear radically different from other forms of information systems, but will be geared toward enabling users to assign meaning to information and to capture some of their knowledge in information and/or data. The knowledge needs to be expressed in such a way that it is interpretable by another individual. This is what testing a KMS is targeting at.

6.2 Software Testing

Testing can never completely identify all the deficits with a KMS but it could provide indications of the suitability of the system. Hence, testing is the process of attempting to make an assessment. Kaner et al. [5] make a point stating that testing is a skilled, fundamental multidisciplinary area of work and that software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test. When testing information quality and the characteristics sought after, the strategy provided by software testing can provide the stepping-stone for providing indication about the quality of a KMS. In doing this line of work different methods are used.

Testing can be done for several purposes using different methods. The first differentiation is based on how the tester can manipulate the artifact that is tested [7]. In a white box approach, the tester has full access to internal structures and can see and manipulate everything. The tester can, for example, inject errors to see what happens. On the other end of the spectrum, there is black box testing where the tester has no access to any internal information. Gray box testing is a compromise between the two, and allows for test cases to be designed using internal information, but the actual tests are black box.

Testing can happen on several different levels of specificity, for example module, system, etc. Unit testing is focused on testing a specific part [3], a well-defined unit, and is often done by the developers using a white box approach. The different units together form a system, and the system is tested on a system level. The focus is to verify that the system behaves according to the requirements. Testing can also focus on the integration. Integration testing [2] exist on both unit and system level and focus on testing how well they fit together. On the system integration testing level, the focus is to verify how well the system integrates with other pre-existing systems, for example.

While the purpose of testing is often functional, there exist non-functional (higher order) testing [7] as well. Common non-functional concerns that are tested include usability, security, and performance.

Information quality testing can use both white and black box testing, and exists on the different levels discussed above. For example, a technical writer working on a specific page might use tools to check spelling and hyperlinks, for example. This constitutes automatic white box unit testing. The scenarios we focus on in this paper are manual black box tests that focus on system and system integration. The complete documentation is tested, but the focus is also on how well the documentation integrates with the phenomenon or artifacts it documents. Similarly, the testing covers both functional and non-functional aspects.

While the functionality (i.e., how well the phenomenon or artifact is documented) is important, readability, understandability, etc. are equally important. At the moment, it is unclear how to separate the functional and non-functional concerns when testing information quality.

7 Conclusion

This paper defines an approach to information quality testing based on software and usability testing of. One or more information test cases that include tasks to be solved or questions to be answered are created and given to a number of test users. These test users use the information available in the KMS to solve the tasks or answer the questions included in the information test case(s). While they attempt to solve the task or answer the questions, data is collected. This data includes the time, the number of errors the test users perform, how they access the system, etc. This data is then forwarded to the persons responsible for information quality of the KMS.

The data gathered during the information testing is presented using several views that are suitable for different persons in the organization. One view might target the technical writers, while others target managers or information consumers. By offering different views, the different processes of defining and using a KMS get supported and information testing as such becomes more integrated into the organization.

Our approach has been well received by producers of technical documentation. The feasibility study described in this paper was carried out in collaboration with Sigma Kudos and they see great potential in our approach to information testing. Our current feasibility study uses technical documentation. We consider the content management system and the technical documentation to be equivalent to a KMS, but we still need to test with an actual KMS. In order to do this, we need to extend the tools to work with popular KMS implementations.

There is also a need to extend the feasibility study to include more test users and test cases. The current study uses team members and small test cases. In order to validate the approach, we need to perform more and larger tests, with more diverse groups of testers.

Acknowledgment

We want to acknowledge the Swedish Research School of Management and IT (MIT), Uppsala, for supporting Anna Wingkvist. While, Morgan Ericsson, is supported by the Uppsala Programming for Multicore Architectures Research Center (UPMARC). Also, we acknowledge the Knowledge Foundation for financing part of the research with the project, "Validation of metric-based quality control", 2005/0218. We would like to extend our gratitude to Applied Research in System Analysis AB (ARiSA AB, `http://www.arisa.se`) for providing us with the VizzAnalyzer tool and to Sigma Kudos AB, `http://www.sigmakudos.com`) for providing us with the DocFactory and raw data.

Bibliography

[1] M. Alavi and D. E. Leidner. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, 25(1):107–136, 2001. ISSN 02767783.

[2] B. Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990. ISBN 0-442-20672-0.

[3] R. V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 0-201-80938-9.

[4] B. K. Kahn, D. M. Strong, and R. Y. Wang. Information quality benchmarks: product and service performance. *Commun. ACM*, 45(4):184–192, 2002. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/505248.506007.

[5] C. Kaner, J. Bach, and B. Pettichord. *Lessons Learned in Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 0471081124.

[6] W. Löwe and T. Panas. Rapid Construction of Software Comprehension Tools. *International Journal of Software Engineering and Knowledge Engineering*, 15(6):905–1023, Dec. 2005. Special Issue on Maturing the Practice of Software Artifacts Comprehension, Ed. Nenad Stankovic, World Scientific Publishing.

[7] G. J. Myers and C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004. ISBN 0471469122.

[8] I. Tuomi. Data is more than knowledge: implications of the reversed knowledge hierarchy for knowledge management and organizational memory. *J. Manage. Inf. Syst.*, 16(3):103–117, 1999. ISSN 0742-1222.

[9] R. Y. Wang. A product perspective on total data quality management. *Commun. ACM*, 41(2):58–65, 1998. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/269012.269022.

[10] G. M. Weinberg. *Quality software management (Vol. 1): systems thinking*. Dorset House Publishing Co., Inc., New York, NY, USA, 1992. ISBN 0-932633-22-6.

[11] A. Wingkvist, M. Ericsson, W. Löwe, and R. Lincke. A metrics-based approach to technical documentation quality. In *Proceedings of the 7th International Conference on the Quality of Information and Communications Technology*, 2010. forthcoming.