

Electronic version of an article published in **iSQI (Hrsg) Proceedings of the 3rd World Congress for Software Quality 2005, Vol. 2, pp 213-222**

Copyright © [2005] International Software Quality Institute GmbH (iSQI)

<http://www.isqi.org/isqi/deu/conf/wcsq/3/>

Criteria for Software Testing Tool Evaluation – A Task Oriented View

T. Illes, A. Herrmann, B. Paech, J. Rückert

Institute for Computer Science, University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{[illes.herrmann.paech.rueckert](mailto:illes.herrmann.paech.rueckert@informatik.uni-heidelberg.de)}@informatik.uni-heidelberg.de

Abstract. Considering the immense variety of test tools available, both commercial and open source, an extensive evaluation of these tools with respect to their adequacy for a given organizational or project context seems to be impossible. On this account we present a systematic approach for deriving easily verifiable evaluation criteria for test tools. We define both, quality criteria and functional criteria. Using the TORE methodology we identify activities which potentially could be automated or at least supported by a test tool. Starting from these activities we derive evaluation criteria for test tools. We focus on criteria which can be evaluated considering only vendors' instructions, consequently without an extensive laboratory test. The result of our work is a reasonable list of criteria allowing an effective classification and pre-selection of test tools. In a first step we applied our criteria to evaluate three capture & replay tools. At this, our approach proved of value. The fundamental differences between the tools could be identified and additional criteria for this particular test automation technique could be defined.

1 Introduction

Browsing the web one can find an immense variety of tools supporting different activities performed during the test process. Even though there are over 600 test tools no approach exists for classifying these tools, apart from a very superficial categorization into test tools supporting test planning, test design, test execution, defect tracking or configuration management as mentioned in [20]. Therefore in product overviews plainest tools and extensive ones are grouped into the same category. In this paper we refine these criteria to allow a more detailed classification of tools according to the context of a special project without lapsing into the contrary by defining criteria, which are only applicable on the basis of extensive product evaluations. Suchlike criteria require a high effort so that we confine ourselves to criteria which allow a pre-selection only by consulting the vendor's instructions.

In order to determine *quality criteria* for test tools, we extend the characteristics of the quality model proposed by the standard ISO/IEC 9126 [10] by criteria related to vendor qualification as mentioned in [4]. For deriving a reasonable set of *functional criteria* we analyzed the test process applying the TORE methodology [16]. We identified that way the tasks performed while systematically testing a software system

and the corresponding roles. Refining these tasks we detected activities which potentially could be automated or at least supported by a test tool. The result of our work is a set of quality and functional criteria. For validating our results, we then exemplarily applied the criteria in order to evaluate three capture & replay tools.

Related Work. Apart from the criteria defined by Poston and Sexton [18] there is no founded approach for deriving evaluation criteria for test tools. Since the criteria proposed by Poston and Sexton focus on company specific criteria or on criteria requiring a high effort to be evaluated e.g. test effort or test quality (percentages of found defects), these criteria do not apply for a pre-selection of the test tools. Additionally, criteria specific to test tools are mentioned without a justification for their derivation. The criteria mentioned by Poston and Sexton represent a subset of the criteria systematically derived in our approach.

Commercial test tool evaluations offered by OVUM [15] and CAST [1], [2], and most of the non-commercial ones as proposed in [19] or [4] appraise only a small subset of the test tools/frameworks available. These works concentrate on a detailed evaluation of products offered by the market leaders requiring extensive evaluation (installation and use) of the test tools. In contrast, we focus on more coarse-grained criteria enabling an effective pre-selection for users. Furthermore, our approach does not restrict the tools to be evaluated to a minimal set determined by the market leaders so that a wide range of test tools can be considered. The criteria defined in the CAST-reports are not available to us.

Overview. The remainder of this paper is organized as follows. Section 2 summarizes quality criteria for test tools. In section 3 we analyze the test process from a task oriented view and define activities performed during the test process. Section 4 summarizes the functional criteria derived from the activities identified in section 3. Section 5 then summarizes the result of an exemplary tool evaluation and section 6 concludes the paper and presents our intended future work.

2 Quality Criteria for Testing Tools

For specifying quality criteria for testing tools, we used the ISO/IEC 9126 standard [10]. The ISO/IEC standard defines a framework for the evaluation of software quality, proposing a hierarchical model, which consists of six characteristics and corresponding sub characteristics for software product quality. Furthermore we supplemented these characteristics by criteria related to vendor qualifications proposed in [4]. These criteria consider the following additional aspects: general vendor qualifications, vendor support, licensing and pricing. Subsequently, we summarize the quality evaluation criteria we determined this way. The criteria Q1-Q6 represent the characteristics and sub-characteristics of the ISO/IEC 9126 standard, the criteria Q7-Q9 represent the criteria related to vendor qualifications. A refinement of these criteria can be found in [9] and [3].

- Q1 Functionality** suitability, accurateness, interoperability, compliance, security (1-5)
- Q2 Reliability** maturity, fault tolerance, recoverability (6-8)
- Q3 Usability** understandability, learnability, operability (9-11)

Q4 Efficiency	time behavior, resource behavior (12-13)
Q5 Maintainability	analyzability, changeability, stability, testability (14-17)
Q6 Portability	adaptability, installability, conformance, replaceability (18-21)
Q7 General vendor qualifications	
	22 maturity of the vendor, market share, financial stability
Q8 Vendor support	
	23 warranty, maintenance and upgrade policy
	24 regularity of upgrades, defect list with each release
	25 compatibility of upgrades with previous releases
	26 e-mail support, phone support, user groups
	27 availability of training, recommended training time, price
Q9 Licensing and pricing	
	28 open source or commercial
	29 licensing used, rigidity (floating node-locking license)
	30 price consistent with estimated price range
	31 price consistent with comparable vendor products

3 The Test Process: A Task Oriented View

We now consider functional criteria for testing tools. For this reason we analysed the test process. While the criteria proposed by Poston and Sexton [18] focus on organisational issues and the recommended functional criteria are coarse grained, our work aims at proceeding in a more general way by considering the whole test process so that we can derive more detailed and more test specific criteria. For a more abstract view on the test process we applied therefore the TORE (Task and Object-oriented Requirements Engineering) methodology proposed by Paech and Kohler in [16]. This methodology was designed to give guidance to the specification of user requirements on different abstraction levels. In the context of this work the task and the domain level are relevant. At *task* level, the most abstract of the four levels, user tasks and corresponding roles are identified and specified. These tasks are then refined by further activities at *domain* level.

We applied the TORE methodology to identify tasks and roles involved in the test process. Tasks are typically identified by analysing the current work of roles involved into the process and emphasis on the work context of these roles [16]. We identified the tasks by analyzing test process descriptions mentioned in standard textbooks such as Spillner [20, 17] and Mosley and Posey [14]. The basis of our work is the fundamental test process described in [20] consisting of planning, specification, test execution, capturing and analysing test results. We then detailed these tasks according to [14], resulting in a more precise view of the tasks concerning test design and test execution. Finally we added tasks continuously performed during the whole test process concerning the *tracking of defects* and the management of the *test ware* as mentioned in [17]. Test ware includes all artifacts such as test specifications, test scripts and test data, developed during the test process [17]. In a further step we refined these tasks by activities. Table 3 summarizes the main tasks and the corresponding roles involved in the test process we identified in this step.

Table 1. Tasks and roles involved in the test process.

ID	Tasks	Role(s)
A	Test planning and monitoring	Test manager
B	Designing Test Cases	Test designer
C	Constructing Test Cases	Test automator, test designer
D	Executing test cases	Tester
E	Capturing and comparing test results	Tester
F	Reporting test results	Tester
G	Tracking Software problem reports/defects	Tester, test manager, developer
H	Managing the test ware	Test configuration manager, test administrator

Subsequently we present refining activities performed during the corresponding task.

A: Test planning and monitoring. Considering that testing activities represent 30 – 40 % of all activities in the software life cycle [21], it is critical to plan and begin test activities as early as possible. As testing activities are complex, a sound planning and monitoring of testing activities is crucial to the success of the project. Thus, key activities of test planning include the tailoring of the organizational test process to the requirements of the current project, identifying the constraints of the current project¹, identifying the test strategy, prioritizing test activities, assessing risks, scheduling, identifying staffing and training needs, defining metrics to monitor the progress of test activities, defining pass/fail criteria. Key activities in project monitoring include collecting metrics, project tracking and adapting the test plan according to the current circumstances. Beside these activities a coordination of the interface to other processes in the software development life cycle is necessary.

B: Designing Test Cases. ²Key activities of designing test cases are to choose test techniques based on the defined test strategy, to identify test conditions, to derive and document logical test cases, and to identify the layout of the test data for logical tests. The design of test cases includes both, test cases for testing functional requirements (the “*what*”) but also quality aspects (the “*how*”) of the system under test (SUT). The ISO/IEC standard [10] proposes a framework for characterising quality aspects for software.

C: Constructing Test Cases. Key activities of constructing test cases include the implementation of test cases by defining concrete test cases and test data. In case of test automation test scripts are developed. A test script can realize required preconditions, the execution steps and set-up and clear-down facilities. Additionally, test scripts can implement generation procedures for test data and expected outcomes.

D: Executing Test Cases. The test cases designed and constructed in the phases before must be executed. These activities can be done manually (by executing the

¹ Constraints can concern e.g. programming paradigm, specific application characteristics such as web application, etc.

² A *test case* specification should include the test items, input specifications, output specifications, environmental needs, special procedural requirements and inter-case dependencies [7].

defined steps of the test case) or automated by running the test script. A semi-automated execution of the test scripts is also possible.

E: Capturing and comparing test results. Key activities of this task include the documentation of the executed test steps and outcomes and the verification of the test results by comparing expected and actual outcomes.

F: Reporting test results. Reporting activities aim at aggregating test information on different detail levels in a comprehensible and reproducible manner: Test reports document test results and their analysis [14]. Therefore, reports should have a customizable degree of detail with respect to the intended audience (managers, customer, and developers)

G: Tracking Software problem reports/defects. The key activities of this task concern a problem's or defect's traceability during their lifecycle. This lifecycle includes the recording and classifying of a defect/problem, the monitoring of the progress on corrective activities, the prioritization of defects to decide whether corrective actions should be started and regression testing activities of corrected defects. According to this, a defect passes through various states (new, opened, denied, analysing, correcting, testing, closed, flop) during its life cycle [20].

H: Managing the test ware. Key activities of this task include the management of the test ware (versioning, storage, sharing, ...), providing traceability between the elements of the test ware (requirements, logical test cases, concrete test cases, test data files, ...), tracing modifications on a test object and communicating changes, and creating "snapshots" of the test ware at the end of a test cycle in order to conserve the results of a test cycle for regression testing.

4 Deriving Functional Criteria for Test Tools

Based on the activities identified using the TORE methodology, we derived evaluation criteria for test tools. We identified those activities performed in the tasks described in section 3 that could be potentially supported, respectively automated by a test automation tool. The resulting criteria are designed in such a way, that they can be evaluated by "yes" or "no" to enable an efficient evaluation of a variety of test tools. For a more fine-grained classification, a scaled rating could be chosen (e.g. for a specific criteria the following scale is conceivable to express to what extent a feature provided by the particular tool supports the criteria: 0 – not supported, 1 – semi-automated, 2 – automated). Subsequently we list the functional criteria grouped by the tasks described in section 3.

A: Test planning and monitoring. Test tool provides support for:

1. customization of the organizational test process
2. particular programming paradigms³ and/or languages, operating systems, browser, network configuration
3. application specific characteristics, which require specific testing techniques⁴
4. testing special application domain (e.g. avionics, automotive, etc.)

³ E.g. component-based, imperative, model-driven, etc.

⁴ E.g. web-based, GUI testing, real-time testing, database testing, protocol conformance testing.

5. planning of the test process (scheduling, project tracking, risk management)
6. monitoring test activities
 - by tracking of the estimated and actual time/test case
 - by providing coverage metrics to measure the progress of testing activities
 - by providing metrics from different sources (e.g. requirements, test cases)
6. integration with other tools⁵

B: Designing Test Cases. Test tool provides support for:

7. designing test cases for the required test level (unit, integration, system)
8. selecting the test techniques
9. defining test conditions derived from the defined test techniques
10. defining templates for structuring the information specifying test cases
11. generation of logical test cases from semi-formal models^{6?}
12. generation of logical test cases from formal specifications (e.g. Z)
13. generation/derivation of test data layout⁷
14. optimizing the test case set⁸
15. designing test cases to test quality criteria of the application⁹
16. restricting the test case set (e.g. ranking by prioritisation of the test cases, risks assigned to test cases) in case of deadline constraints

C: Constructing Test Cases. Test tool provides support for:

17. editing test scripts
18. developing of test code conforming with accepted software engineering practices¹⁰
19. capturing of executable test cases
20. generation of concrete test cases from (semi-)formal models⁶
21. generation of (in)valid test data¹¹
22. generation of stubs, test drivers, mock objects,
23. simulating missing faulty system components

D: Executing Test Cases. Test tool provides support for:

24. setting-up and clearing-down of the test environment/pre condition and respectively the post conditions for a set of test cases
25. roll-back to initial in case of unexpected errors
26. execution of captured, captured & edited or manually implemented test cases for functional testing.
27. execution of captured test cases for testing quality criteria¹²

⁵ e.g. with other testing tools, project management tools, requirement management tools, etc.

⁶ E.g. UML scenarios, UML state charts, UML sequence diagrams.

⁷ E.g. template-based/code based/interface based/specification based/database-based (schema definitions)/using results of former test cycles.

⁸ E.g. minimizing test cases by concurrently maximizing coverage of requirements.

⁹ A possible set of quality criteria is defined by the ISO/IEC 9126 standard as described in section 2.

¹⁰ E.g. modularisation of the code, comments, declaration of variables and data types; parameter passing; separation of test data and definition of the course of events.

¹¹ E.g. code based; interface based; specification based; from databases (live data information, schema information), randomly, rule-based.

¹² By performing *static analysis* e.g. checking conformance of the source code to style guides or by performing *dynamic analysis* e.g. performance testing or load testing

28. stopping and continuation of the execution of a suspended test case

E: Capturing and comparing test results. Test tool provides support for:

29. logging information on executed test cases¹³

30. comparison facilities between specified and actual outcomes

F: Reporting test results. Test tool provides support for:

31. aggregation of logged test results

32. customizable, role specific amount of information

G: Tracking Software problem reports/defects. Test tool provides support for:

33. specifying problem reports/defects by using predefined templates

34. generating entries for recorded defects

35. prioritizing defects

36. tracking change requests/defects and their current status

37. generating statistical information

38. for regression testing¹⁴

H: Managing the test ware. Test tool provides support for:

39. management of the test ware¹⁵

40. traceability between the elements of the test ware¹⁶

41. by tracing modifications on a test object and communicating changes

42. the maintenance of the test data, of the test cases

43. for automated tests to be (re)used for regression testing/in other projects

44. snapshot facilities (by freeze a special state of the test ware)

5 Applying the Evaluation Criteria

In order to evaluate the applicability of the derived criteria, we exemplarily analysed and compared three capture and replay tools. The capture and replay approach is primarily applied to regression tests of web and GUI-based applications. Capture and replay tools are similar in their mode of operation consisting of four steps as described in [8] and [12]: (1) *Capture mode*, where all manual user interactions on the test object are captured; (2) *Programming*, where the captured interactions are mapped to a test script, (3) *Checkpoints*, where additional checkpoints are added to the test script, (4) *Replay-mode*, where captured scripts can be replayed. In step 4 results from former tests and current results are compared. In case they differ, the test fails. The test also fails if defined checkpoints are violated.

The three tools we evaluated are: *WinRunner* [13], *Rational Robot* [6], and *HTTrace* [11]. *WinRunner* is distributed by Mercury. This tool is about the market leader (54% of the market in test tools). *Rational Robot* is distributed by IBM and holds 17% of the market in test tools. *HTTrace* is not a commercial product, but developed for internal use for the company i-TV-T. We analyzed HTTrace because of its scientific relevance. Basis of our evaluation was the documentation provided by

¹³ E.g. executed steps, outcomes and exceptions but also memory/resource usage, network load, code coverage information.

¹⁴ E.g. by identifying test case set to be re-executed for regression testing

¹⁵ E.g. versioning, storage, sharing, and authorisation facilities.

¹⁶ E.g. requirements, logical test cases, concrete test cases, test data files, outcomes, etc.

the vendors on their web site. Additionally, we got feedback from Mercury and i-TV-T regarding our evaluation. Subsequently we present a brief summary of the evaluation results. A detailed report can be read in [5].

A: Test planning and monitoring. The evaluated tools do not focus on supporting test planning and monitoring activities.¹⁷

B: Designing test cases. Since the design of test cases does not represent a key feature of the evaluated test tools, most of these functions e.g. guidance in selecting test techniques/defining test conditions, derivation/generation of test cases or test data are not supported. For testing quality criteria of the SUT HTTrace provides performance tests. For stress tests an additional component, HTBlast, can be integrated. For WinRunner and Rational Robot, the additionally components *Load Runner and TestDirector* respectively *Rational Performance Tester* must be integrated for both, performance and stress test. Usability testing is not provided by any of the vendors.

C: Constructing test cases. Constructing test cases by capturing user interaction represents one of the key features of the tools where captured scripts can be manually edited. Expected outcomes are defined by outcomes in previous test runs. In case of regression testing current and previous outcomes are compared in order to decide if the test failed or passed. None of the tools supports the generation of test data (valid, invalid) or the generation of stubs or test drivers.

D. Executing test cases. All tools support the execution of test cases by running the captured/implemented test scripts. The separation of test data and test script allows scripts to be parameterized by different test data. Both, WinRunner but also Rational Robot provide this facility, HTTrace does not offer a solution for this problem.

E. Capturing and comparing results. Logging of outcomes occurs in a separate file. Each tool allows comparison of expected (defined by previous test runs) and actual outcome.

F: Reporting test results. Reporting features can be added to each tool by integrating additionally components.¹⁸

G. Tracking software problem reports/defects. Reporting features can be added by integrating additionally components¹⁹

H. Managing the test ware. WinRunner provides traceability of test cases to requirements by integrating the additional component *TestDirector*, and by Rational Robot by integrating *TestManager*. HTTrace provides rudimentarily this facility. Rational Robot provides a *Recovery Manager* for a programmable handling for exceptions in the SUT. WinRunner and HTTrace allow the SUT to be restarted after a crash.

¹⁷ The integration of additional components like *TestDirector* provided by Mercury and *Rational Test Manager, Rational Administrator* or *Rational Team Unifying Plattform* provided by Rational support project planning and monitoring facilities. For HTTrace no additional components supporting project planning are available.

¹⁹ WinRunner can be extended by *TestDirector*, Rational Robot by *Rational ClearQuest, Rational Team Unifying Plattform* or *Unified Change Management* and HTTrace by *Scarab* or *Tracilla* in order to provide defect tracking facilities. These additional components provide workflow support for automatic notification of changes to the state of a defect.

Summary. Key features of all three test tools are the construction of test cases by capturing and subsequently editing the test scripts and the execution of the recorded test scripts. Comparing previous and current outcomes regression tests can be efficiently performed. By integrating additional components, WinRunner and Rational Robot can be extended to provide test planning and monitoring as well as defect and reporting facilities. HTRace's strength lies on testing database applications by allowing the reset of consistent database states. Additionally, all three tools can be extended to provide support for testing quality attributes of the system under test e.g. performance.

By evaluating these tools we could identify supplementary criteria specific to testing web applications by the capture and replay technique: Object recognition (e.g. HTML-tables, frames, links, etc.), repository for managing objects, mapping of non standard objects to standard objects (e.g. checkboxes, buttons, etc.), masking facilities for particular areas on the screen, support for comparing images, OCR (optical character recognition) to extract text from images, etc. A complete list of criteria, specific to web testing and a detailed evaluation of the test tools can be found in [5].

6 Conclusion and Future Work

Applying the TORE methodology to the test process with the objective of deriving criteria for test tools proved to give a useful set of classification and selection criteria. The comparison of three capture & replay tools showed that the defined criteria are effectively applicable to the evaluation of test tools based on vendor's documentation. The evaluation revealed the basic differences between the test tools. Additionally, our criteria helped to derive criteria specific to a particular test technique, capture & replay in this case.

The evaluation criteria derived in our approach can be used for diverse purposes. At first, our criteria allow a pre-selection of test tools according to (coarse) initial requirements. Considering that a detailed evaluation is very time-consuming, an efficient pre-selection is crucial. Applying our criteria, such a pre-selection can be carried out effectively on the basis of vendor's information. Certainly, a definitive decision for using a special tool can only be made by a detailed investigation of the tools which passed the pre-selection process. Then, the evaluation criteria can be applied for conducting a survey on commonly used products and to derive the state of the art in testing tools as well as to identify gaps in the market and scientific challenges. Additionally, the proposed criteria can be used as a framework for classifying research results in the area of testing, especially in test automation.

Our future research focuses on providing a survey on test tools using our criteria to classify commercial and open source tools. Additionally, we aim at refining our criteria for testing activities specific to particular applications (e.g. object oriented testing, component based testing, GUI testing, etc.).

Acknowledgements. We want to thank Carsten Binnig, Lars Borner and Dima Suliman for their constructive suggestions, comments and references. We also thank Mercury and i-TV-T for supporting our evaluations activities.

References

1. CAST Tools. An Evaluation and Comparison, http://www.dpu.se/blocast_e.html (last visited: July 2005)
2. CAST: Testing Tools - CAST for the Millennium and Beyond, http://www.dpu.se/blott_e.html (last visited: July 2005)
3. Dörr, J., Punter, T., Bayer, J., Kerkow, D., Kolb, R., Koenig, T., Olsson, T., Trendowicz, A. : Quality Models for Non-functional Requirements, IESE report nr. 101.04./E, 2004
4. Dustin, E., Rashka, J., McDiarmid, D.: Quality web systems: performance, security, and usability, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2002
5. Herrmann, A., Paech, B., Rückert, J.: Bericht über die Herleitung von Qualitätskriterien für Test-Werkzeuge, unpublished.
6. IBM, Rational Robot product overview, <http://www-306.ibm.com/software/awdtools/tester/robot/index.html>, (last visited: July 2005)
7. IEEE standard 829-1998, IEEE Standard for Software Test Documentation, IEEE Computer Society, 1998.
8. imbus: Automatisierung von Softwaretests, 7 Fallstricke und wie man sie überwindet, www.imbus.de, 2002.
9. INCOSE: „INCOSE Requirements Management Tool Survey“, <http://www.paper-review.com/tools/rms/read.php> (last visited: January 2005)
10. International Standard ISO/IEC 9126: Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use, International Organization for Standardization, International Electrotechnical Commission, Geneva. (1991)
11. i-TV-T: i-TV-T product overview, <http://www.i-tv-t.de>, (last visited: January 2005)
12. Linz, T. Daigl, M.: How to Automate Testing of Graphical User Interfaces, Results of the ESSI PIE 24306, http://www.imbus.de/forschung/pie24306/gui/aquis-full_paper-1.3.html, (last visited: July 2005)
13. Mercury: WinRunner product overview, <http://www.mercury.com/de/products/quality-center/functional-testing/winrunner/>, (last visited: July 2005)
14. Mosley, D. J. and Posey, B. A.: Just Enough Software Test Automation, Prentice Hall, July 2002
15. Ovum Evaluates: Software Testing Tools, <http://store.ovum.com/detail.aspx?ID=1257>, (last visited: July 2005)
16. Paech, B., Kohler, K.: Task-driven Requirements in object-oriented Development, In J. Leite, J. Doorn, (eds.) Perspectives on Requirements Engineering, Kluwer Academic Publishers, 2003
17. Pol, M., Koomen, T., Spillner, A.: Management und Optimierung des Testprozesses, Dpunkt Verlag, 2. Aufl., 2002.
18. Poston, R.M.; Sexton, M.P.: Evaluating and selecting testing tools, IEEE Software, Volume: 9 Issue: 3, May 1992, S. 33 -42.
19. Robinson, Ray: Automation Test Tools, www.stickyminds.com, (last visited: July 2005), 2001
20. Spillner, A. and Linz, T.: Basiswissen Softwaretest, dpunkt.verlag, (2004)
21. Spillner, A.: From V-Model to W-Model – Establishing the Whole Test Process. Proceedings Conquest 2000 – 4th Conference on Quality Engineering in Software Technology, Nürnberg, 2000, S. 222-231