Towards the improvement of the Semantic Web technology

Raúl García-Castro

Ontology Engineering Group, Departamento de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid, Spain
rgarcia@fi.upm.es

Abstract. The Semantic Web technology needs to be thoroughly evaluated for providing objective results and to attain a massive improvement in their quality in order to be consolidated in the industrial and in the academic world. This paper presents software benchmarking as a process to carry out over the Semantic Web technology in order to improve it and to search for best practices. It also describes a software benchmarking methodology and provides recommendations for performing evaluations in benchmarking activities.

1 Introduction

The Semantic Web technology has improved considerably since the development of the first tools in the nineties and, although it has been mainly used in research laboratories, in recent years companies have started to be interested in the Semantic Web technology and in the applications using this technology.

In order to consolidate this technology, both in the industrial and in the academic world, it is necessary that the Semantic Web technology reaches a maturity level where it can comply with the quality requirements required by the industry. Therefore, the Semantic Web technology needs both to be thoroughly evaluated for providing objective results and to attain a massive improvement in their quality.

Up to now, the evaluation of the Semantic Web technology was seldom made but, as its use has spread, in the last few years numerous studies involving the evaluation of this technology have appeared. *This paper encourages researchers first to increase the quality of their evaluations and, second, to aim for collective improvements in their technology by means of benchmarking it.*

The notion of software benchmarking presented in this paper is that of a continuous process for improving software products, services, and processes by systematically evaluating and comparing them to those considered to be the best. Although software evaluations are performed inside benchmarking activities, benchmarking provides some benefits not obtained from software evaluations.

This idea of benchmarking as a process to search for improvement and best practices derives from the idea of benchmarking in the business management community. However, it differs from some Software Engineering approaches in which benchmarking is considered a software evaluation method for system comparisons.

This paper also presents a methodology for carrying out software benchmarking activities that promotes the research community to drive the benchmarking, since this community includes the experts that have developed the current technology.

In the Knowledge Web[1] European Network of Excellence, different benchmarking studies are being carried out over several types of Semantic Web technologies, focusing on industry and research interests and needs. Here we present how this methodology has been used in one of these benchmarking activities.

This paper is structured as follows. Sections 2, 3 and 4 present what software evaluation is, the different conceptions of benchmarking, and our conception of software benchmarking, respectively. Section 5 presents some properties of evaluations when they are performed in software benchmarkings, and Section 6 describes what a benchmark suite is and its desirable properties. Sections 7 and 8 provide an overview of the software benchmarking methodology and show how this methodology has been used for benchmarking the interoperability of ontology development tools. Finally, Section 9 draws some conclusions from this paper.

2 Software Evaluation

Software evaluations play an important role in different areas of Software Engineering, such as Software Measurement, Software Experimentation or Software Testing. In this paper, we present a view that covers all these areas, not focusing on any of them.

According to the ISO 14598 standard [1], software evaluation is *the systematic examination of the extent to which an entity is capable of fulfilling specified requirements*; considering software as not just as a set of computer programs but also as the produced procedures, documentation and data.

Software evaluations can take place all along the software life cycle. They can be performed during the software development process by evaluating intermediate software products or when the development has finished.

Although evaluations are usually performed inside the organisation that develops the software, other groups of people, such as users or auditors, that are independent of the organisation can perform the evaluations. The use of independent third parties in software evaluations can be very effective, but these evaluations are much more expensive for the organisations [2].

The goals of evaluating software depend on each specific case, but they can be summarised from [3–5] in the following:

- To **describe** the software in order to understand it and to establish baselines for comparisons.

[1] http://knowledgeweb.semanticweb.org/

– To **assess** the software with respect to some quality requirements or criteria and determine the degree of desired qualities of the software product and its weaknesses.
– To **improve** the software by identifying opportunities for improving its quality. This improvement is measured by comparing it with the baselines.
– To **compare** alternative software products or different versions of a same product.
– To **control** the software quality by ensuring that it meets the needed level of quality.
– To **predict** in order to take decisions, establishing new goals and plans for accomplishing them.

The methods to follow for evaluating software vary from one author to another and from one Software Engineering area to another. Nevertheless, from the methods proposed in the areas of Software Evaluation [1, 6], Software Experimentation [7–9], Software Measurement [4, 10], and Software Testing [11] we can extract a common set of tasks to carry out in software evaluations:

1. To establish the evaluation requirements, setting its goals, the entities to evaluate, and their relevant attributes.
2. To define the evaluation, explaining the data to collect, the evaluation criteria and the mechanisms to collect data, implementing these mechanisms.
3. To produce the evaluation plan.
4. To execute the evaluation and to collect data.
5. To analyse the collected data.

3 Benchmarking in the Literature

In the last decades, the word benchmarking has become relevant within the business management community. The definitions well known are those due to Camp [12] and Spendolini [13]. Camp defined benchmarking as *the search for industry best practices that lead to superior performance*, while Spendolini expanded Camp's definition by adding that benchmarking is *a continuous, systematic process for evaluating the products, services, and work processes of organisations that are recognised as representing best practices for the purpose of organisational improvement.* In this context, best practices are good practices that have worked well elsewhere, are proven and have produced successful results [14].

These definitions highlight the two main benchmarking characteristics:

– Continuous improvement.
– Search for best practices.

The Software Engineering community does not share a common benchmarking definition. Some of the most representative ones are:

– Kitchenham [15] and Weiss [16] define benchmarking as a software evaluation method suitable for system comparisons. For Kitchenham, benchmarking is *the process of running a number of standard tests using a number of alternative tools/methods and assessing the relative performance of the tools in those tests*; and for Weiss benchmarking is *a method of measuring performance against a standard, or a given set of standards.*

– Wohlin et al. [17] adopt the business benchmarking definition, considering benchmarking as a continuous improvement process that strives to be the best of the best through the comparison of similar processes in different contexts.

4 Software Benchmarking

In this paper, we have adopted the ideas of continuous improvement and search of best practices from business management benchmarking, which have led us to consider software benchmarking as a continuous improvement process instead of as a punctual activity. Equally important are the notion of software comparisons through evaluations and the need for a systematic procedure for carrying out the benchmarking activity.

This notion permits us to define **software benchmarking** as a continuous process for improving software products, services and processes by systematically evaluating and comparing them to those considered to be the best.

However, this definition does not limit the entities to be considered in the benchmarking (software products, services or processes), the phase of the software life cycle when benchmarking is performed, or who is responsible for carrying out the benchmarking. Nevertheless, software benchmarking is usually performed over software products already developed and carried out by their developers.

The reason for benchmarking software products instead of just evaluating them is to obtain several benefits that cannot be obtained from software evaluations. A software evaluation shows us the weaknesses of the software or its compliance to quality requirements. If several software products are involved in the evaluation, we also obtain a comparative analysis of these products and recommendations for users. When benchmarking several software products, besides all the benefits commented, we gain continuous improvement of the products, recommendations for developers on the practices used when developing these products and, from these practices, those that can be considered best practices.

5 Software Evaluation in Benchmarking Activities

Making software evaluations is not a straightforward task but, as it is a topic that has been thoroughly examined both in theory and practice, several authors have proposed different recommendations for carrying them out [3, 8, 9, 18, 19].

These recommendations are also applicable software evaluations that take place in benchmarking activities. However, when defining this kind of software evaluations, some additional recommendations must also be taken into account.

The most important recommendation that must be considered is that benchmarking evaluations must be **improvement-oriented**. Their intended results are not only to compare the different software products, but to know how to improve them. This requires that the evaluations obtain not just some comparative results but the practices that produced these results.

Benchmarking evaluations should be as **general** as possible, taking into account that they will be performed by different groups of people in different locations and over different software.

Benchmarking is a process driven by a community, and to gain credibility and impact its evaluations should also be **community-driven**.

Benchmarking evaluations should be **reproducible** as they are intended to be used not just by the benchmarking partners but by the whole community. This requires the evaluation to be detailed, using public data and procedures.

Performing evaluations consumes significant resources and, in benchmarking, these evaluations must be performed by several groups of people. Therefore, they should be as **inexpensive** as possible using common evaluation frameworks or limiting the scope of the evaluation.

Furthermore, as benchmarking is a continuous process, benchmarking evaluations should have a **limited scope**, leaving other objectives for the next benchmarking iterations. A broader evaluation scope does not entail better results, but it does entail more resources.

As the next section shows, most of these recommendations should also be fulfilled by benchmark suites. Therefore, it is advisable to **use benchmark suites** in benchmarking evaluations.

6 Benchmark Suites

A benchmark suite is a collection of benchmarks, being a benchmark *a test or set of tests used to compare the performance of alternative tools or techniques* [20].

A benchmark definition must include the following:

– The **context** of the benchmark, namely, which tools and which of their characteristics are measured with it.
– The **requirements** for running the benchmark, namely, the tools (hardware or software), data, or people needed.
– The **input variables** that affect the execution of the benchmark, and the values that they will take.
– The **procedure** to execute the benchmark and to obtain its results.
– The **evaluation criteria** used to interpret these results.

A benchmark suite definition must include the definition of all its benchmarks. Usually, all these benchmarks share some of their characteristics, such as the context or the requirements. In that case, these characteristics are defined at the benchmark suite level, and not individually for each benchmark.

6.1 Desirable properties of a benchmark suite

The following properties, extracted from the work of different authors [21, 22, 20, 23], can help either to develop new benchmark suites or to assess the quality of different benchmark suites before using them.

Although a good benchmark suite should have most of these properties, each evaluation will require that some of them be considered before others.

It must also be considered that achieving a high degree of all these properties in a benchmark suite is not possible since the increment of some has a negative influence over others.

Accessibility A benchmark suite must be accessible to anyone interested. This involves providing the necessary software to execute the benchmark suite, its documentation, and its source code in order to increase transparency.
The results obtained when executing the benchmark suite should be made public so that anyone can apply the benchmark suite and compare his/her results with the ones available.

Affordability Using a benchmark suite entails a number of costs, commonly human, software, and hardware resources. The costs of using a benchmark suite must be lower than those of defining, implementing, and carrying out any other experiments that fulfil the same goal.
Some ways of reducing the resources consumed in the execution of a benchmark suite are: automating the execution of the benchmark suite, providing components for data collection and analysis, or facilitating its use for different heterogeneous systems.

Simplicity The benchmark suite must be simple and interpretable. It must be documented so anyone who wants to use it must be able to understand how it works and the results that it yields. If the benchmark suite is not transparent enough, its results will be questioned and it could be interpreted incorrectly. To ease the process, the elements of the benchmark suite should have a common structure and use and common inputs and outputs. Measurements should have the same meanings across the benchmark suite.

Representativity The actions that perform the benchmarks composing the benchmark suite must be representative of the actions that are usually performed on the system.

Portability The benchmark suite should be executed on a variety of environments as wide as possible, and should be applicable to as many systems as possible.
It should also be specified at a high enough level of abstraction to ensure that it is portable to different tools and techniques and that it is not biased against other technologies.

Scalability The benchmark suite should be parameterised to allow scaling the benchmarks with varying input rates.
It should also scale to work with tools or techniques at different levels of maturity. It should be applicable to research prototypes and commercial products.

Robustness The benchmark suite must consider unpredictable environment behaviours and should not be sensitive to factors not relevant to the study. When running the same benchmark suite several times on a given system under the same conditions, the results obtained should not change considerably.

Consensus The benchmark suite must be developed by experts who apply their knowledge of the domain and are able to identify the key problems. It should also be assessed and agreed on by the whole community.

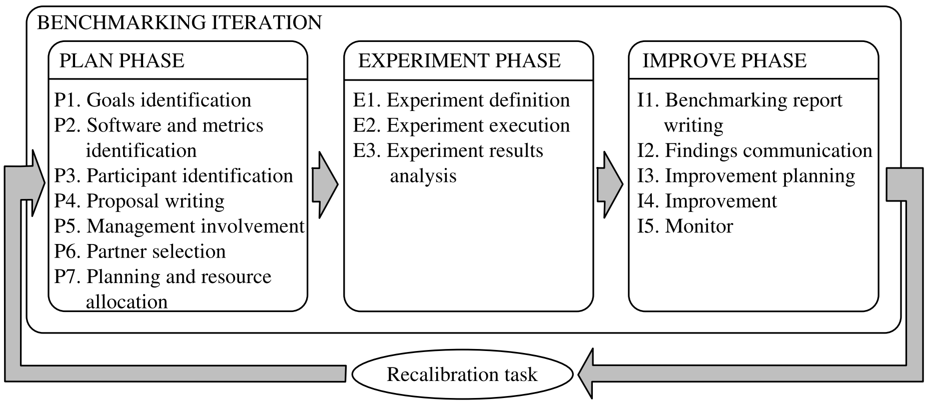
7 Software Benchmarking Methodology

This section summarises the software benchmarking methodology developed by the author in the Knowledge Web European Network of Excellence. A detailed description of this methodology can be found in [24].

This methodology has been inspired by works in different fields of quality improvement. The main input for this methodology were the benchmarking methodologies existing in the business management community and their notions of continuous improvement and best practices. Nevertheless, evaluation and improvement processes proposed in the Software Engineering area were also considered, such as the ones cited in Section 2.

The software benchmarking methodology defines a benchmarking process with the main phases to carry out when benchmarking software and provides a set of guidelines to follow. Therefore, this methodology has a twofold use: to assist in carrying out software benchmarking activities, or to know, at a certain point of time, which is the actual progress of a benchmarking activity.

The benchmarking process defined in this methodology is composed of a benchmarking iteration that is repeated forever. Each iteration, as shown in Figure 1, is composed of three phases (*Plan*, *Experiment* and *Improve*) and ends with a *Recalibration* task.



BENCHMARKING ITERATION
PLAN PHASE
P1. Goals identification
P2. Software and metrics identification
P3. Participant identification
P4. Proposal writing
P5. Management involvement
P6. Partner selection
P7. Planning and resource allocation
EXPERIMENT PHASE
E1. Experiment definition
E2. Experiment execution
E3. Experiment results analysis
IMPROVE PHASE
I1. Benchmarking report writing
I2. Findings communication
I3. Improvement planning
I4. Improvement
I5. Monitor
Recalibration task

Fig. 1. The software benchmarking methodology

7.1 Plan phase

The *Plan* phase is composed of the tasks that must be performed for preparing the benchmarking proposal, for obtaining support from the organisation management, for finding other organisations that participate in the benchmarking, and for planning the benchmarking. These tasks are the following:

P1. Goals identification. In this task, the *benchmarking initiator* (the member or members of the organisation who become/s aware of the need for benchmarking) must identify the benchmarking goals according to the organisation goals and strategies and the benefits and the costs of performing benchmarking. There can be different goals when performing benchmarking in an organisation, such as assessing the performance and improving the quality of the software over time, comparing the software with the software that is considered the best, or establishing or creating standards by analysing the different existing software.

P2. Software and metrics identification. In this task, the *benchmarking initiator* must perform an analysis of the software products developed in the organisation in order to understand and document them, identifying their weaknesses and their functionalities that need improvement.
Then, the benchmarking initiator must select the products that will be benchmarked, the functionalities that are relevant for the study and the evaluation criteria that will be used to assess these functionalities, according to the organisation's software analysis, the benchmarking goals, the benefits and costs identified in the previous task, and other factors considered critical in the organisation such as quality requirements, end user needs, etc.

P3. Participant identification. In this task, the *benchmarking initiator* must identify and contact the members of the organisation involved in the selected software and functionalities (managers, developers, end users, etc.) and other relevant participants from outside the organisation (customers or consultants).
The benchmarking initiator must compose the benchmarking team and, usually, he will be a member of it. This team should be small and should include those organisation members whose work and interest are related to the software and have an understanding of the software and valuable experience with it.
The members of the benchmarking team must be aware of the time that they will spend in the benchmarking activity and of their responsibilities and must be trained in the tasks that they will have to perform.

P4. Proposal writing. In this task, the *benchmarking team* (and the *benchmarking initiator*, if he is not part of the team) must write a document with the benchmarking proposal. This proposal will be used as a reference along the rest of the benchmarking process.
The benchmarking team must consider the different intended readers of the benchmarking proposal, that is, organisation management, organisation developers, members of partner organisations, and the benchmarking team themselves.

The proposal must include all the relevant information about the process: the information identified in the previous benchmarking tasks (goals, benefits, costs, software, metrics, members involved, and benchmarking team), a description of the benchmarking process, and a more detailed description of the benchmarking costs with the resources needed in the benchmarking such as people, equipment, travel, etc.

P5. Management involvement. In this task, the *benchmarking initiator* must bring the benchmarking proposal to the *organisation management*. This task is of great importance because management's approval is needed to continue with the benchmarking process. Management's support will also be needed in the future when implementing changes based on the benchmarking either in the software or in the organisation's processes that affect the software.

P6. Partner selection. In this task, the *benchmarking team* must collect and analyse information about the software products that are comparable to the selected one and about the organisations that develop them. The benchmarking team must select the software that will be considered in the benchmarking study according to its relevance and use in the community or in the industry and its use of the latest technological tendencies, its public availability, etc. In order to obtain better results with the benchmarking, the chosen software should be those considered the best.

Then, the benchmarking team must make contact with someone from the organisations that develop these software products to see whether they are interested in becoming *benchmarking partners*. These benchmarking partners will also have to establish a benchmarking team and to carry the benchmarking proposal to their own organisation management for approval.

During this task, the benchmarking proposal will be modified including the partner's opinions and needs. This will result in an updated benchmarking proposal that, depending on the magnitude of the modifications, should be presented again to each partner organisation's management for approval.

P7. Planning and resource allocation. In this task, the *organisation managements* and the *benchmarking teams* from each partner must define the planning of the rest of the benchmarking process, considering the different resources that will be devoted to the benchmarking, and they must reach a consensus on it. This planning must be considered and integrated into each organisation's planning.

7.2 Experiment phase

The *Experiment* phase is composed of the tasks where the experimentation over the software products that are considered in the benchmarking is performed. These tasks are the following:

E1. Experiment definition. In this task, the *benchmarking teams* from each partner must define the experiment that will be performed on each of the software products and they must reach a consensus on it.

The experiment must be defined according to the benchmarking goals, the software selected functionalities and their corresponding criteria as stated in the benchmarking proposal. The experiment must also provide objective and reliable data of the software, not just of its performance but also of the reasons of its performance, and it must be defined taking into account its future reuse.

The benchmarking teams must also define and agree on the planning that will be followed during the experimentation, which must be defined according to the benchmarking planning previously defined.

E2. Experiment execution. According to the experimentation planning defined in the previous task, the *benchmarking teams* must perform the experiments defined on their software products.

The data obtained from all the experiments must be compiled, documented, and expressed in a common format in order to facilitate its future analysis.

E3. Experiment results analysis. In this task, the *benchmarking teams* must analyse the results obtained in the experiments, identifying and documenting any significant differences in them, and they must determine the practices that lead to these different results trying to identify whether, among the practices found, some of them can be considered best practices.

Then, the benchmarking teams must write a report with all the findings obtained during the experimentation, namely, experimentation results, differences in the results, practices and best practices found, etc.

7.3 Improve phase

The *Improve* phase is composed of the tasks where the results of the benchmarking process are produced and communicated to the benchmarking partners; and the improvement of the different software products is performed in several improvement cycles. These tasks are the following:

I1. Benchmarking report writing. In this task, the *benchmarking teams* must write the benchmarking report. This report is intended to provide an understandable summary of the benchmarking carried out and must be written bearing in mind its different audiences: managers, benchmarking teams, developers, etc. from all the partner organisations.

The benchmarking report must include an explanation of the process followed with all the relevant information from the updated version of the benchmarking proposal, and the results and conclusions of the experiments presented in the experiment report, highlighting the best practices found in the experimentation and including any best practices found in the community.

The benchmarking report must also include the recommendations of the benchmarking teams for improving the software products according to the experiment results, to the practices found, and to the community best practices.

I2. Findings communication. In this task, the *benchmarking teams* must communicate, in several meetings, the results of the benchmarking to their organisations and, particularly, to all the involved members that were identified when planning the benchmarking. The goals of these meetings are:
   - To obtain feedback from the involved members about the benchmarking process, the results and the improvement recommendations.
   - To obtain support and commitment from the organisation members for implementing the improvement recommendations in the software.

   Any feedback received during these communications must be collected, documented and analysed. This analysis may finally result in having to review the work done and to update the benchmarking report.

I3. Improvement planning. The last three tasks of the *Improve* phase (*Improvement planning*, *Improvement* and *Monitor*) form a cycle that must be performed separately in each organisation. It is in these tasks where each organisation benefits from the results obtained in the benchmarking.

   In this task, the *benchmarking teams* and the *organisation managements* from each partner must identify, from the benchmarking report and the monitoring reports, which are the changes needed to improve their software products. Besides, they must forecast the improvement to be obtained after performing these changes.

   Both the organisation management and the benchmarking team must provide mechanisms for ensuring the accomplishment of improvements in the organisation and for measuring the software functionalities. These last mechanisms can be obtained from the *Experiment* phase.

   Then, they must define the planning for improving the benchmarked software, considering the different resources devoted to the improvement. This planning must be then integrated into the organisation planning.

I4. Improvement. In this task, the *developers* of each of the software product must implement the necessary changes in order to achieve the desired results. They must measure the state of the software before and after implementing any changes using the measurement mechanisms provided by the benchmarking team in the previous task. They must compare the resulting measurements with those obtained before implementing the changes and with the improvement forecast.

I5. Monitor. In each organisation, the *benchmarking team* must provide the *software developers* with means for monitoring the organisation's software. The software developers must periodically monitor the software and write a report with the results of this monitorisation. These monitorisation results can cause a need for new improvements in the software and the beginning of a new improvement cycle, having to perform again the two previously mentioned tasks: *Improvement Planning* and *Improvement.*

7.4 Recalibration task

The recalibration task is performed at the end of each benchmarking iteration. In this task, the *benchmarking teams* must recalibrate the benchmarking process applying the lessons learnt while performing it. In that way, the organisation (and the whole community) achieves improvement not just in the software, but also in the benchmarking process. This recalibration is needed because both software and organisations evolve over time.

8 Using the software benchmarking methodology

The software benchmarking methodology presented in the previous section has been used, within Knowledge Web, in different benchmarking activities, which are being (and will be) performed to improve ontology tools and to offer recommendations on these tools for both research and industry users.

One of these activities is benchmarking the interoperability of ontology development tools using RDF(S) as interchange language[2], with the goal of evaluating and improving the interoperability of these tools.

The experiment defined in the benchmarking involved the use of three benchmark suites for evaluating the RDF(S) importers of the tools, their RDF(S) exporters, and the interoperability between each pair of tools. In these benchmark suites, users were asked to import, export and interchange ontologies using their tools and to record the behaviour of the tools in predefined templates.

Seven tools have participated in the benchmarking, four of which are ontology development tools: KAON, OntoStudio, Protégé using its RDF backend, and WebODE, and three are RDF repositories: Corese, Jena and Sesame. Although the primary targets of the benchmarking were ontology development tools, the benchmark suites defined just require the tools to be capable of importing and exporting RDF(S).

Tool improvement has occurred before the *Improve* phase of the methodology because developers were able to detect problems and correct their tools while executing the benchmark suites.

Nevertheless, the manual execution of the experiments and analysis of the results causes the benchmark suites to be costly. Sometimes tool developers have automated the execution of the benchmark suites, but not always.

A description of the three benchmark suites used in the benchmarking can be found in [25]. [26] includes how the benchmarking methodology was instantiated for the RDF(S) interoperability benchmarking, how the benchmark suites were defined, the detailed benchmarking results of each tool, and recommendations for ontology developers, for software developers, and for anyone interested in performing benchmarking activities.

This methodology is also being used for benchmarking the interoperability of ontology development tools using OWL as interchange language[3]. Furthermore, a proposal for using this methodology for benchmarking the performance and the scalability of ontology development tools can be found in [27].

[2] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/
[3] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/owl/

9 Conclusion

This paper states a need for evaluating and benchmarking the Semantic Web technology and provides some references that can be helpful in these activities.

It also presents the author's view on software benchmarking and compares it with other existing evaluation and benchmarking approaches.

For carrying out benchmarking activities, a benchmarking methodology is presented that can help in assessing and improving software. And the use of benchmark suites is advised when performing evaluations in benchmarking.

One of the strong points on benchmarking is its community-driven approach. Benchmarking is performed by the experts in the community and the benefits obtained after performing it will have an effect in the whole community.

Nevertheless, benchmarking is not the solution to every case. A preliminary step would include the assessment of whether it is the correct approach or not, being useful when the goals are to improve the software and to obtain the practices performed by others.

Acknowledgments

This work is partially supported by a FPI grant from the Spanish Ministry of Education (BES-2005-8024), by the IST project Knowledge Web (IST-2004-507482) and by the CICYT project Infraestructura tecnológica de servicios semánticos para la web semántica (TIN2004-02660). Thanks to Rosario Plaza for reviewing the grammar of this paper.

References

1. ISO/IEC: ISO/IEC 14598-1: Software product evaluation - Part 1: General overview. (1999)
2. Rakitin, S.R.: Software Verification and Validation, A Practitioner's Guide. Artech House (1997)
3. Basili, V., Selby, R., Hutchens, D.: Experimentation in software engineering. IEEE Transactions on Software Engineering 12 (1986) 733–743
4. Park, R., Goethert, W., Florac, W.: Goal-driven software measurement - a guidebook. Technical Report CMU/SEI-96-HB-002, Software Engineering Institute (1996)
5. Gediga, G., Hamborg, K., Duntsch, I. In: Evaluation of Software Systems. Volume Encyclopedia of Computer Science and Technology, Volume 44. (2002) 166–192
6. Basili, V.: Quantitative evaluation of software methodology. In: 1st Pan-Pacific Computer Conference, Melbourne, Australia (1985)
7. Basili, V.R., Selby, R.W.: Paradigms for experimentation and empirical studies in software engineering. Reliability Engineering and System Safety 32 (1991) 171–191
8. Perry, D., Porter, A., Votta, L.: Empirical studies of software engineering: a roadmap. In Finkelstein, A., ed.: The Future of Software Engineering, ACM Press (2000) 345–355
9. Freimut, B., Punter, T., Biffl, S., Ciolkowski, M.: State-of-the-art in empirical studies. Technical Report ViSEK/007/E, Visek (2002)

10. IEEE: IEEE Std 1061-1998 IEEE Standard for a Software Quality Metrics Methodology. IEEE (1998)
11. Abran, A., Moore, J., Bourque, P., Dupuis, R., eds.: SWEBOK: Guide to the Software Engineering Body of Knowledge. IEEE Press (2004)
12. Camp, R.: Benchmarking: The Search for Industry Best Practice that Lead to Superior Performance. ASQC Quality Press, Milwaukee (1989)
13. Spendolini, M.: The Benchmarking Book. AMACOM, New York, NY (1992)
14. Wireman, T.: Benchmarking Best Practices in Maintenance Management. Industrial Press (2003)
15. Kitchenham, B.: DESMET: A method for evaluating software engineering methods and tools. Technical Report TR96-09, Department of Computer Science, University of Keele, Staffordshire, UK (1996)
16. Weiss, A.: Dhrystone benchmark: History, analysis, scores and recommendations. White paper, EEMBC Certification Laboratories, LLC (2002)
17. Wohlin, C., Aurum, A., Petersson, H., Shull, F., Ciolkowski, M.: Software inspection benchmarking - a qualitative and quantitative comparative opportunity. In: Proceedings of 8th International Software Metrics Symposium. (2002) 118–130
18. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers (2001)
19. Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El-Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering 28 (2002) 721–734
20. Sim, S., Easterbrook, S., Holt, R.: Using benchmarking to advance research: A challenge to software engineering. In: Proceedings of the 25th International Conference on Software Engineering (ICSE'03), Portland, OR (2003) 74–83
21. Bull, J.M., Smith, L.A., Westhead, M.D., Henty, D.S., Davey, R.A.: A methodology for benchmarking java grande applications. In: Proceedings of the ACM 1999 conference on Java Grande. (1999) 81–88
22. Shirazi, B., Welch, L., Ravindran, B., Cavanaugh, C., Yanamula, B., Brucks, R., Huh, E.: Dynbench: A dynamic benchmark suite for distributed real-time systems. In: Proc. of the 11 IPPS/SPDP'99 Workshops, Springer-Verlag (1999) 1335–1349
23. Stefani, F., Macii, D., Moschitta, A., Petri, D.: FFT Benchmarking for Digital Signal Processing Technologies. In: 17th IMEKO World Congress, Dubrovnik, Croatia (2003)
24. García-Castro, R., Maynard, D., Wache, H., Foxvog, D., González-Cabero, R.: D2.1.4 Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools. Technical report, Knowledge Web (2004)
25. García-Castro, R.: D2.1.5 Prototypes of tools and benchmark suites for benchmarking ontology building tools. Technical report, Knowledge Web (2005)
26. García-Castro, R., Sure, Y., Zondler, M., Corby, O., Prieto-González, J., Bontas, E.P., Nixon, L., Mochol, M.: D1.2.2.1.1 Benchmarking the interoperability of ontology development tools using RDF(S) as interchange language. Technical report, Knowledge Web (2006)
27. García-Castro, R., Gómez-Pérez, A.: Guidelines for benchmarking the performance of ontology management apis. In Gil, Y., Motta, E., Benjamins, R., Musen, M., eds.: Proceedings of the 4th International Semantic Web Conference (ISWC2005). Number 3729 in LNCS, Galway, Ireland, Springer-Verlag (2005) 277–292