A Benchmark Suite for Evaluating the Performance of the WebODE Ontology Engineering Platform

Raúl García-Castro, Asunción Gómez-Pérez

Ontology Engineering Group, Laboratorio de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid, Spain
{rgarcia,asun}@fi.upm.es

Abstract. Ontology tools play a key role in the development and maintenance of the Semantic Web. Hence, we need in one hand to objectively evaluate these tools, in order to analyse whether they can deal with actual and future requirements, and in the other hand to develop benchmark suites for performing these evaluations. In this paper, we describe the method we have followed to design and implement a benchmark suite for evaluating the performance of the WebODE ontology engineering workbench, along with the conclusions obtained after using this benchmark suite for evaluating WebODE.

1. Introduction

In order for the Semantic Web to consolidate steadily, it needs the support of technology that allows to create and to maintain it. The continuous development of ontology editors and ontology tools for managing ontologies is an indication of this fact. These tools implement different knowledge models with different underlying knowledge representation paradigms, manage large upper level and general ontologies, and range from standalone to web-based applications. To be able to decide what ontology tools are needed for fulfilling actual and future requirements, we need to objectively evaluate them with regard to their quality attributes.

Evaluation studies and benchmark suites for ontology technology are still a bit scarce. For this reason, we think that there is a need to construct benchmark suites for ontology tools, in order to be able to objectively assess their quality and to allow for a better integration of this technology into other information systems.

The volume of ontologies and the amount of users that work concurrently with ontology tools increases continuously. Therefore, the performance of these tools emerges as one of the quality attributes to take care of.

As the development activity is one of the main ontology life cycle activities [1], we will first deal with the appraisal of ontology development tools. In this work, we focus on the WebODE ontology engineering workbench [2], evaluating its performance in terms of execution efficiency [3]. WebODE's global performance is inferred from the performance of the methods of its ontology management API, which allow managing the ontology components defined in the WebODE knowledge model (concepts, relations, instances, axioms, constants, bibliographic references, and imported terms).

The contents of this paper are the following:

Section 2 presents the state of the art in evaluating ontology development tools.

Sections 3 to 7 describe the method that we have followed to design and implement a benchmark suite for evaluating WebODE's performance and the analysis performed after executing it.

Section 8 presents the conclusions obtained and the related future work.

2. Related Work

Ontology technology has improved enormously since the creation of the first environments in the mid-1990s. In general, ontology technology hasn't been the object of evaluation studies but, as the use of this technology spreads, in the last few years many studies involving ontology tools evaluation have been performed. Most of these studies deal with the evaluation of ontology development tools.

Some authors have proposed a general framework for the evaluation of ontology development tools. To this group belongs the work presented by: Duineveld and colleagues [4], the deliverable 1.3 of the OntoWeb project [5], the experiments performed in the First International Workshop on the Evaluation of Ontology-based Tools (EON2002) [6], and Lambrix and colleagues [7].

Other authors have focused in specific criteria regarding ontology development tools: Stojanovic and Motik [8] analyzed the ontology evolution requirements fulfilled by the tools, Sofia Pinto and colleagues [9] evaluated the support provided by the tools in ontology reuse processes, the experiments of the Second International Workshop on Evaluation of Ontology-based Tools (EON2003) [10, 11, 12, 13, 14] involved the interoperability of the tools, and Gómez-Pérez and Suárez-Figueroa [15] evaluated the ability of the tools to detect taxonomic anomalies.

As a general comment, evaluation studies concerning ontology development tools have been of qualitative nature. To be able to objectively assess and compare these tools, evaluation criteria must be defined and benchmark suites must be developed in order to perform formal experiments that deal with quantitative data.

3. Evaluating WebODE

WebODE is a scalable workbench for ontological engineering that provides services for editing and browsing ontologies, importing and exporting ontologies to classical and semantic web languages, evaluating ontologies, mapping ontologies, etc. [2].

Because of the lack of work that deals with evaluating the performance of ontology development tools, the motivations for carrying out this study have been:

- To define a **method** for evaluating WebODE's performance.
- To obtain a **benchmark suite** for evaluating WebODE.
- To **evaluate** WebODE using the benchmark suite.

All these motivations converge in a single long-term goal: to achieve a **continuous improvement** in the platform's quality.

The method we have used to evaluate the performance of WebODE is composed of the following steps:

- To identify the evaluation goals, elements, and metrics.
- To design and implement the benchmark suite.
- To run the benchmark suite.

- To analyze the results obtained after running the benchmark suite.

4. Identification of the Evaluation Goals, Elements, and Metrics

In order to identify the elements and metrics that will be considered in the evaluation we have chosen the Goal/Question/Metric (GQM) paradigm[1] [16]. The idea beyond the GQM paradigm is that any software measurement activity should be preceded by the identification of a software engineering goal, which leads to questions, which in turn lead to actual metrics.

The WebODE ontology management API provides methods to insert, update, remove, and query the components of the WebODE knowledge model[2]. As the services provided by WebODE use these methods for accessing WebODE ontologies, the performance of these services strongly depends of the performance of the API methods. Therefore, our goal is to **evaluate the performance of the methods provided by the WebODE ontology management API**. Table 1 presents the questions and the metrics derived from this goal according to the GQM paradigm. The analysis of the results of executing the benchmark suite will provide answers to these questions.

Table 1. Questions and metrics obtained through the GQM approach

Question	**Metric**
Q1: Which is the actual performance of the WebODE API methods?	Execution time of each method
Q2: Is the performance of the methods stable?	Variance of the execution times of each method
Q3: Are there any anomalies in the performance of the methods?	Percentage of execution times out of range in each method's sample
Q4: Do changes in a method's input parameters affect its performance?	Execution time with parameter A = Execution time with parameter B
Q5: Does WebODE's load affect the performance of the methods?	WebODE's load versus execution time relationship

In summary, the elements to evaluate are the **72 methods** of the WebODE ontology management API, and the metric to use is the **execution time** of the methods over incremental load states.

5. Design and Implementation of the Benchmark Suite

Several authors have enumerated the desirable properties of a benchmark suite [17, 18, 19]: generality, representativeness, transparency, interpretability, robustness, scalability, portability, accessibility, and repeatability. These properties have been the basis of the requirements definition for the benchmark suite.

[1] Other approaches are Quality Function Deployment [20] and Software Quality Metrics [21].
[2] http://kw.dia.fi.upm.es/wpbs/WebODE_API_methods.html

5.1. Requirements for the Benchmark Suite

In order to have a **generic** and **representative** benchmark suite, the benchmarks that compose it use every WebODE API method, performing current operations over WebODE ontologies.

To be able to compare each method's execution time, the methods must be executed under the same conditions. Therefore, we have defined the execution environment and the load state of WebODE, having fixed both before running each benchmark.

The benchmarks and their results must be **transparent** and **interpretable**. Each benchmark executes just one method and stores the wall clock time elapsed while running the method. The only other operation performed by a benchmark is to restore the load state of WebODE in case it changed during the benchmark execution.

Furthermore, as each method has its input parameters, one or more benchmarks have been defined for each method according to variations in these input parameters. So, from the 72 API methods we got **128 benchmarks**[3].

As the benchmarks must be **robust** and **scalable**, they have been parameterized according to two parameters:

- **Load factor (X).** The load factor of WebODE's load state when executing a benchmark.
- **Number of iterations (N).** The number of consecutive executions of a method in a single benchmark. This parameter defines the number of sample measurements obtained after executing a benchmark.

For example, from the method *insertTerm(String ontology, String term, String description)*, that inserts concepts in an ontology, we defined two benchmarks regarding the different input values of the method. These benchmarks were parameterized according to the load factor (X) and the number of executions (N):

- benchmark1_1_08. It inserts X concepts in an ontology. This is repeated N times.
- benchmark1_1_09. It inserts 1 concept in X ontologies. This is repeated N times.

In order to have a **portable** benchmark suite, the benchmarks have been implemented in Java, using only standard libraries and with no graphical components.

The benchmark suite must also be **accessible** and **repeatable**, anyone should be able to replicate the experiments and achieve the same conclusions. Therefore, the benchmark suite source code and the results obtained in this evaluation are published in a public web page[4].

5.2. Definition of the Execution Environment

As the workload used in the evaluation must be characterized accurately [22], we have defined the execution environment with the variables that influence the execution time of a method: hardware configuration, software configuration, computer's load, and WebODE's load.

The WebODE's load variable has been the only one whose values have been altered. As we are not interested in the other three variables' effect in the execution times, these variables have taken fixed values during the execution of the benchmarks. Furthermore, in order to avoid other non-controlled variables that may

[3] http://kw.dia.fi.upm.es/wpbs/WPBS_benchmark_list.html

[4] http://kw.dia.fi.upm.es/wpbs/

affect the results, the computer used for executing the methods has been isolated: without network connection nor user interaction. Next, we define these variables and the values that they took when running the benchmarks.

- **Hardware configuration.** It is the hardware configuration of the computer where WebODE is running. The computer was a Pentium 4 2.4 Ghz monoprocessor with 256 Mb. of memory.
- **Software configuration.** It is the configuration of the operating system and of the software needed to execute WebODE. It was the following, using each system's default configuration: Windows 2000 Professional Service Pack 4; SUN Java 1.4.2_03 (the benchmarks were compiled with the default options); Oracle version 8.1.7.0.0 (the Oracle instance's memory configuration was changed to: Shared pool 30 Mb., Buffer cache 80 Mb., Large pool 600 Kb., and Java pool 32 Kb.); Minerva version 1 build 4; and WebODE version 2 build 8.
- **Computer's load.** It is the load of the computer where WebODE is running. This load was the corresponding to the computer just powered on, with just the programs and services needed to run the benchmarks.
- **WebODE's load.** It is the underlying database's load where WebODE ontologies are stored. The generation of this load is explained in the next section.

5.3. Workload Generation

We mentioned before that WebODE's load state must be the same for every benchmark execution. This common load state must also allow to execute the benchmarks with different load factors (X) and with no errors. Therefore, WebODE's initial load state has been worked out from each benchmark's execution needs.

Each **benchmark's minimum load state** has been defined as the minimum ontology components that must exist in WebODE in order to execute the benchmark with no errors. For example, considering the four benchmarks whose methods insert and remove concepts in an ontology, Table 2 shows each of these benchmark's minimum load state, being X the load factor, and Table 3 shows the minimum load state of the four benchmarks.

Table 2. Minimum load states of the benchmarks whose methods insert and remove concepts

Benchmark	Operation	Minimum load state
benchmark1_1_08	Inserts X concepts in an ontology	1 ontology
benchmark1_1_09	Inserts a concept in X ontologies	X ontologies
benchmark1_3_20	Removes X concepts from an ontology	1 ontology with X concepts
benchmark1_3_21	Removes a concept from X ontologies	X ontologies with one concept

Table 3. The minimum load state of the benchmarks shown in Table 2

Benchmarks	Minimum load state
benchmark1_1_08, benchmark1_1_09, benchmark1_3_20, and benchmark1_1_21	1 ontology with X concepts and X ontologies with1 concept

Therefore, the **benchmark suite initial load state**[5], used when executing all the benchmarks, has been defined as the union of all the benchmarks' minimum load states, and is composed of all the ontology components that must exist in WebODE in order to execute every benchmark with no errors.

[5] http://kw.dia.fi.upm.es/wpbs/WPBS_workload_generation.html

6. Execution of the Benchmark Suite

The 128 benchmarks that compose the benchmark suite have been run ten times with increasing load factors (X=500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, and 5000) and with a number of iterations (N) of 400.

With the aim of checking that 400 iterations is a valid sample size, we have run several benchmarks with higher and lower number of iterations and we have confirmed that the results obtained are virtually equivalent. We haven't used a higher sample because the slight precision improvement would mean a much higher duration of the benchmark suite execution.

As with a load factor (X) of 5000 we obtained enough data to be able to determine the methods' performance, the benchmarks haven't been executed with higher load factors.

The results of running a benchmark are N (in this case 400) measurements of the execution time of the method used in a benchmark. These results are stored in a text file in a hierarchical measurement data library, so as to access them easily.

7. Analysis of the Results

The conclusions obtained when analyzing the results of a benchmark suite execution are usually temporarily limited [23]. As the API methods will undergo changes, the results just inform about WebODE's current performance, not its future one.

In order to obtain information that can be used to make decisions, we must apply statistical analysis techniques to the execution results [24]. From the N measurements of the execution time of a method in a benchmark we can obtain:

- **Graphs** that show the behavior of the execution time.
- **Statistical values** worked out from the sample of N measurements.

Observing the graphs of the execution times measured in a benchmark, we saw that execution times are usually constant. One example of this can be seen in Figure 1, where the execution times of benchmark1_3_20 when running the method *removeTerm* 400 times with a load factor of 5000 are shown.

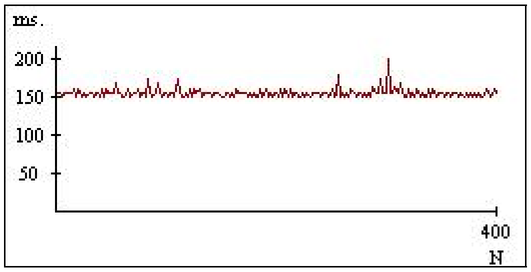

Figure 1. Execution times of benchmark1_3_20 with X=5000 and N=400

The next issue was to find which statistical values could be used to describe the execution times' samples. First of all, we ran normality tests over the measurements obtained after executing the benchmarks. As the distributions of the measurements were non-normal, we could not rely on usual values like mean and standard deviation for describing these measurements. Therefore, we used robust statistical values like the **median**, the **upper and lower quartiles**, and the **interquartile range** (upper minus lower quartile).

To obtain the number of measurements out of range, we calculated the **percentage of outliers** in each sample. The traditional method is to consider as potential outlier values the measurements beyond the upper and lower quartiles adding and subtracting respectively 1.5 times the interquartile range [25]. As the Java method used for measuring time (*java.lang.System.currentTimeMillis()*) in the Windows platform has a precision of tens of milliseconds, in the results we frequently encountered interquartile ranges of zero milliseconds. This caused to be considered as outliers every determination that differed from the median. With the objective of fixing this precision fault, we have augmented the interquartile range when calculating the outliers to include half the minimal granularity (5 milliseconds) in both boundaries.

An example of the statistical values obtained can be seen in Table 4. This table shows the values obtained for the four benchmarks whose methods insert and remove concepts in an ontology. All the statistical values and graphs can be looked up and downloaded from the benchmark suite web page.

Table 4. Statistical values of the benchmarks whose methods insert and remove concepts

	Load	**N**	**UQ**	**LQ**	**IQR**	**Median**	**% Outliers**	**Function**
benchmark1_1_08	5000	400	60	60	0	60	1,25	y=62,0-0,0090x
benchmark1_1_09	5000	400	912	901	11	911	1,75	y=910,25-0,0030x
benchmark1_3_20	5000	400	160	150	10	150	1,25	y=155,25-0,0030x
benchmark1_3_21	5000	400	160	150	10	151	0,25	y=154,96-0,0010x

Next, we show the conclusions drawn after analyzing the data, answering the questions previously proposed in Table 1.

7.1. Finding out Methods' Performance

In order to be able to clearly differentiate the execution times, we have analyzed the data obtained from running the benchmarks with the maximum load factor used, X=5000, and with a number of iterations (N) of 400. We use the median of the execution times of a method in a benchmark as an indicator of its **performance**.

The medians of the execution times of all the API methods range from 0 to 1051 milliseconds. Figure 2 shows the histogram of these medians, where we clearly see a group of values higher than the rest. The execution times of this group belong to 12 benchmarks that execute 8 methods (as different benchmarks have been defined for each method). These 8 methods, with a median execution time higher than 800 ms., have been selected for the improvement recommendations. The rest of the methods have median execution times lower than 511 ms., being most of them around 100 ms.

Bearing in mind the kind of operation that the methods carry out (inserting, updating, removing, or selecting an ontology component), we did not find significant differences between the performances of each kind of method.

Taking into account what kind of element of the knowledge model a method manages (concepts, instances, class attributes, instance attributes, etc.), in the slowest methods' group are present almost every method that manages relations between concepts. Methods that manage instance attributes also have high execution times, and the rest of the methods behave similarly, only standing out the methods that manage imported terms and references as being the ones with lower execution times.

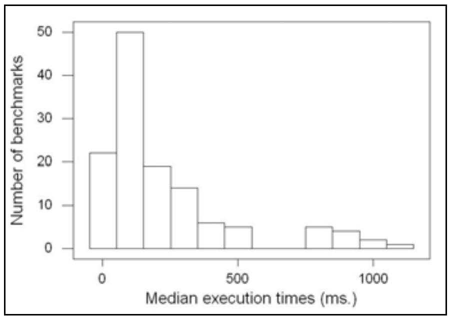

Figure 2. Histogram of the medians of the execution times

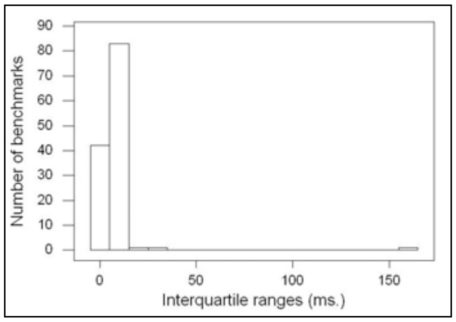

Figure 3. Histogram of the interquartile ranges of the execution times

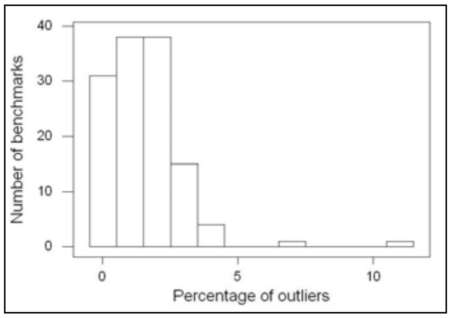

Figure 4. Histogram of the percentage of outliers of the execution times

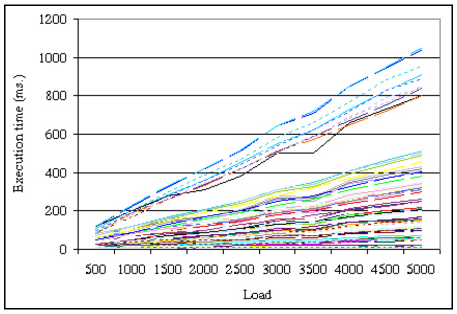

Figure 5. Evolution of the execution times when increasing WebODE's load

Regarding the **spread** of the execution times of the methods, we analyzed the interquartile range (IQR) of the execution times of the methods. Figure 3 shows the histogram of the IQRs of the execution times. Almost every method has an IQR from 0 to 11 ms. Having into account that the granularity of the measurements is of 10 milliseconds, we can state that the execution times have a low spread. The only exceptions are *removeTermRelation* (benchmark1_3_09) with an IQR of 19, *addValueToClassAttribute* (benchmark1_1_14) with an IQR of 30, and *getAvailableOntologies* (benchmark1_4_01) with an IQR of 160. This last method has been selected for the improvement recommendations due to its atypical IQR value.

In order to detect **anomalies**, we generated the histogram of the percentage of outliers in the execution times of the methods, shown in Figure 4. Most of the benchmarks have from 0 to 3.75% of outliers. These values confirm the lack of anomalies except the peaks in the execution times that can be seen in the graphs. The only methods to emphasize are *openOntology* (benchmark2_01), with 11.5% of outliers, and *addValueToClassAttribute* (benchmark1_1_15), with 7% of outliers. These two methods have been selected for the improvement recommendations.

Studying whether **changes in a method's parameters** affect its performance, we have observed that in 21 methods the performance varies when changing its input parameters. This variation is lower than 60 milliseconds except in five methods that show a difference in their execution times when changing parameters from 101 to 851 ms., and have been selected for the improvement recommendations.

7.2. Establishing Load-Performance Relationship

To study WebODE's load effect in performance, we analyzed the medians of the execution times of the methods from a minimum load state (X=500) to a maximum load state (X=5000), and with a number of iterations (N) of 400. We estimated the function that these medians define by **simple linear regression** and considered its **slope** in order to examine the relationship between the load and the execution time of the methods.

Figure 5 shows the plot of every benchmark's median execution time with the different load factors. As can be seen, the 8 methods whose execution times are higher than the rest are also the methods whose performance is more influenced by the load. To be precise, the slope of these methods' function is greater than 0.15, and the slope of the rest of the methods' function ranges from 0 to 0.1. As we stated before, these methods have been selected for the improvement recommendations.

7.3. Development of Improvement Recommendations

Once the data has been analyzed, the next step is the development of the improvement recommendations. These recommendations include those methods whose execution times:

- Have a median execution time higher than 800 ms.
- Have an interquartile range greater than 150 ms.
- Have more than a 5% of outlier values.
- Vary more than 100 ms. when modifying its input parameters.
- Increase when augmenting load with a slope greater than 0.15.

Table 5 shows the 12 of the 72 WebODE's API methods included in the improvement recommendations, and the reasons for their inclusion.

Table 5. Methods in the improvement recommendations

	Execution time > 800 ms.	Interquartile range > 150 ms.	Outlier values > 5%	Execution time variation > 100 ms.	Slope when increasing load > 0.15
removeTermRelation	X				X
getInheritedTermRelations	X				X
insertTerm	X			X	X
insertRelationInstance	X			X	X
openOntology	X		X		X
getAdHocTermRelations	X				X
getTermRelations	X				X
getAvailableOntologies	X	X			X
addValueToClassAttribute			X		
insertConstant				X	
updateSynonym				X	
getInstances				X	

8. Conclusions and Future Work

In this paper we have set out the method we followed to develop a benchmark suite for assessing the temporal performance of the WebODE ontology engineering workbench. We have also stated how we executed this benchmark suite and the main conclusions obtained after analyzing the collected results.

The main achievement obtained after performing this study is that we have precisely determined WebODE's performance, identifying:

- The slowest methods.
- The methods with high spreaded execution times.
- The methods with anomalies in their execution times.
- The effect of changing a method's parameters in its performance.
- The link between WebODE's load and its methods' performance.

The analysis of the results showed that changes in a method's input parameters significantly affected its performance. This fact must be taken into account when defining benchmark suites, either for WebODE or for other systems.

Besides being able to **evaluate** WebODE's performance, the benchmark suite that we have developed will allow us to:

- **Monitor** WebODE, being able to observe the performance of critical elements and how changes in the platform affect its performance.
- **Diagnose** future problems in WebODE.

The benchmarks that compose the proposed suite just execute one method and store its execution time. One way of improving the study could be making these benchmarks execute different kinds of synthetic requests to WebODE, in order to study not just the stability of the methods but the stability of the whole platform.

Once we know the performance of WebODE regarding its ontology management API, we could work out the approximate performance of the services and applications that use this API. This could be done either by empirically obtaining the frequency of use of real services and applications or by defining use frequencies for each different kind of application.

In this work we have just evaluated the execution efficiency of the WebODE API methods. There are many other WebODE attributes that we would be interested in measuring like reliability, usability, or functionality (to cite just a few samples); and future studies will focus in them.

Although this benchmark suite has been designed specifically for WebODE, we plan to extend it to other ontology engineering platforms (KAON, OntoEdit, Protégé-2000, etc.). This could be done either by finding commonalities between the ontology management APIs of the different platforms or by means of a common management API such as OKBC [26].

Acknowledgments

This work is partially supported by the IST project KnowledgeWeb (IST-2004-507482) and by the IST project Esperonto (IST-2001-34373).

References

1. [Fernández-López et al., 1997] M. Fernández-López, A. Gómez-Pérez, N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, 1997, pp 33-40.

2. [Arpírez et al., 2003] J.C. Arpírez, O. Corcho, M. Fernández-López, A. Gómez-Pérez. WebODE in a nutshell. AI Magazine. 24(3), Fall 2003, pp. 37-47.
3. [IEEE, 1991] IEEE-STD-610 ANSI/IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. February 1991.
4. [Duineveld et al., 1999] A.J. Duineveld, R. Stoter, M.R. Weiden, B. Kenepa, and V.R. Benjamins. Wondertools? a comparative study of ontological engineering tools. In Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), Banff, Canada, 1999. Kluwer Academic Publishers.
5. [Ontoweb, 2002] Ontoweb deliverable 1.3: A survey on ontology tools. Technical report, IST OntoWeb Thematic Network, May 2002.
6. [Angele and Sure, 2002] J. Angele and Y. Sure (eds.). Evaluation of ontology-based tools. In Proceedings of the 1st International Workshop EON2002, Sigüenza, Spain, September 2002. CEUR-WS.
7. [Lambrix et al., 2003] P. Lambrix, M. Habbouche, and M. Pérez. Evaluation of ontology development tools for bioinformatics. Bioinformatics, 19(12):1564-1571, 2003.
8. [Stojanovic and Motik, 2002] L. Stojanovic and B. Motik. Ontology evolution within ontology editors. In Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON2002), Sigüenza, Spain, October 2002.
9. [Sofia Pinto et al., 2002] H. Sofia Pinto, Duarte Nuno Peralta, and Nuno J. Mamede. Using Protégé-2000 in reuse processes. In Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON2002), Sigüenza, Spain, October 2002.
10. [Corcho et al., 2003] O. Corcho, A. Gómez-Pérez, D.J. Guerrero-Rodríguez, D. Pérez-Rey, A. Ruiz-Cristina, T. Sastre-Toral, and M.C. Suárez-Figueroa. Evaluation experiment of ontology tools' interoperability with the WebODE ontology engineering workbench. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
11. [Isaac et al., 2003] A. Isaac, R. Troncy, and V. Malais. Using XSLT for interoperability: DOE and the travelling domain experiment. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
12. [Calvo and Gennari, 2003] F. Calvo and J.H. Gennari. Interoperability of Protégé 2.0 beta and OilEd 3.5 in the domain knowledge of osteoporosis. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
13. [Fillies, 2003] C. Fillies. Semtalk EON2003 semantic web export / import interface test. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
14. [Knublauch, 2003] H. Knublauch. Case study: Using Protégé to convert the travel ontology to UML and OWL. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
15. [Gómez-Pérez and Suárez-Figueroa, 2004] A. Gómez-Pérez and M.C. Suárez-Figueroa. Evaluation of RDF(S) and DAML+OIL import/export services within ontology platforms. In Proceedings of the Third Mexican International Conference on Artificial Intelligence, pages 109-118, Mexico City, Mexico, April 2004.
16. [Basili et al., 1994] V.R. Basili, G. Caldiera, D.H. Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering, 2 Volume Set Willey, 1994, pp 528-532.
17. [Bull et al., 1999] J.M. Bull, L.A. Smith, M.D. Westhead, D.S. Henty, R.A. Davey. A Methodology for Benchmarking Java Grande Applications. EPCC, June 1999.
18. [Shirazi et al., 1999] B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, E. Huh. DynBench: A Dynamic Benchmark Suite for Distributed Real-Time Systems. IPDPS 1999 Workshop on Embedded HPC Systems and Applications, San Juan, Puerto Rico, April 1999.

19. [Sim et al., 2003] S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: A challenge to software engineering. In Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pages 74-83, Portland, OR, 2003.
20. [Dean, 1992] E.B. Dean. Quality Function Deployment for Large Systems. Proceedings of the 1992 International Engineering Management Conference, Eatontown NJ, October 1992, pp 317-321.
21. [Boehm et al., 1976] B.W. Boehm, J.R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In Proceedings of the Second International Conference on Software Engineering. San Francisco, 1976, pp 592-605.
22. [Dongarra et al., 1987] J. Dongarra, J.L. Martin, J. Worlton. Computer benchmarking: paths and pitfalls. IEEE Spectrum, Vol. 24, N. 7, July 1987, pp 38-43.
23. [Gray, 1993] J. Gray. The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann, 1993.
24. [Fenton, 1991] N.E. Fenton. Software Metrics A Rigorous Approach. Chapman & Hall, London, UK, 1991.
25. [Mendenhall and Sincich, 1995] W. Mendenhall and T. Sincich. Statistics for Engineering and the Sciences, 4th Edition. Englewood Cliffs, NJ. Prentice Hall, 1995.
26. [Chaudri et al., 1997] V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, J.P. Rice. The Generic Frame Protocol 2.0. Technical Report, Stanford University, 1997.