A Cost Model for Ontology Engineering

Elena Paslaru Bontas, Malgorzata Mochol
AG Netzbasierte Informationssysteme
paslaru@inf.fu-berlin.de
mochol@inf.fu-berlin.de

August 3, 2005

Technical Report B-05-03
Version 2

Abstract

In this report we propose a methodology for cost estimation for ontologies and analyze cost factors implied in the engineering process. We examine the appropriateness of a COCOMO-like parametric approach to ontology cost estimation and propose a non-calibrated ontology cost model, which is to be continuously refined along with the collection of empiric data on person month efforts invested in developing ontologies in real-world projects. We further describe the human-driven evaluation of the cost drivers described in the parametric model on the basis of the cost models' quality framework by Boehm[5]

Executive Summary

No	Item	Author	Date	Description
1	Version 1	Paslaru, Mochol	April 2005	First draft of the ontology cost model
2	Version 2	Paslaru, Mochol	August 2005	Added model validation methodology

Contents

1 Motivation

Though ontologies and associated ontology management tools have become increasingly popular in the last decades, the dissemination of ontologies and ontology-based applications as envisioned by the Semantic Web community requires fine-grained methodologies which are able to deal with both *technical* and *economical* challenges of ontology engineering. In order for ontologies to be built and deployed at a large scale one needs not only technologies and tools to assist the development process, but also proved and tested means to control the *overall engineering process*. A wide range of ontology engineering methodologies have emerged in the Semantic Web community[15, 16, 13, 40, 36]. Apart from minor differences in the level of detail adopted for the description of the process stages these methodologies define ontology engineering as an iterative process, which shows major similarities to the neighbored research field of software engineering. However existing methodologies do not cover a crucial aspect of the engineering process, which has gained significant attention in adjacent engineering areas because of its importance in real-world business contexts: the costs estimation using pre-defined cost models.

In order to precisely estimate the costs related to the ontology engineering process, there is a need for empirically tested cost models which exploit the results already achieved w.r.t. this issue in related engineering fields. In the same time a cost model for ontologies should take into account the critical factors and particularities of the ontology engineering process.

2 Estimating Costs for Ontology Engineering

Cost Estimating is defined as the art of predetermining the lowest realistic cost and price of an item or activity which assure a normal profit. In the case of Ontology Engineering a cost model aims at *predicting the costs (efforts in person months or duration) related to activities performed during the life cycle of an ontology.*

Estimating costs for engineering processes can be performed according to several methodologies[26, 32, 5]. Due to their limitations w.r.t. certain classes of situations these methods are often used in conjunction during the estimation phase:

Analogy Method The main idea of this methodology is the extrapolation of available data from similar projects to estimate the costs of the proposed project. The method is suitable in situations where empirical data from previous project is available and trustworthy and depends on the accuracy in establishing real differences between completed and current projects.

Bottom-Up Method This methodology involves identifying and estimating costs of individual project components separately and subsequently combining the outcomes to produce an estimation for the overall project. It

can not be applied early in the life cycle of the process because of the lack of necessary information related to the project components.

Top-Down Method In contrast to the bottom-up approach the top-down method relies on overall project parameters. For this purpose, the project is partitioned into lower-level components and life cycle phases beginning *at the highest level*. This method is more applicable to early cost estimates when only global properties are known, but it can be less accurate due to the less focus on lower-level parameters and technical challenges—usually predictable later in the process life cycle, at most.

Expert Judgment/Delphi Method The Delphi Method is based on a structured process for collecting and distilling knowledge from a group of human experts by means of a series of questionnaires interspersed with controlled opinion feedback. The involvement of human experts using their past project experiences is a significant advantage of the Delphi approach, while the most extensive critique point is related to the subjectivity of the estimations and the difficulties to explicitly state the decision criteria used by the contributing experts.

Parametric/Algorithmic Method This method involves the usage of mathematical equations based on research and historical data from previous projects. The method analyzes main cost drivers of a specific class of projects and their dependencies and uses statistical techniques to refine and customize the corresponding formulas. As in the case of the analogy method the generation of a proved and tested cost model using the parametric method is directly related to the availability of reliable and relevant data to be used in calibrating the initial core model.

The applicability of the mentioned cost estimation methodologies to ontology engineering depends of course on the characteristics of the ontology engineering process. In the following we examine the pro's and con's of each of these approaches given the current state of the art in Ontology Engineering:

Analogy Method The analogy method requires an accurate comparison function for ontologies and assumes that we are aware of cost information from previous ontology engineering projects. While several similarity measures for ontologies have already been proposed in the Semantic Web community (for example in [24]), no case studies on ontology costs are currently available.

Bottom-up Method This methodology implies the availability of cost information w.r.t. single engineering tasks, such as costs involved in the conceptualization of single concepts or in the instantiation of the ontology. Due to the lack of available information the bottom-up method can not be applied yet to ontology engineering.

Top-Down Method This method can be applied to predict the costs associated with different upper-level stages of the engineering process. Despite the young nature of the ontology engineering discipline, the Semantic Web community has made significant progresses in analyzing the ontology engineering process with its phases, their particularities and associated activities[15]. This is why we expect a top-down method to be appropriate to construct an ontology cost model. For this purpose, one needs techniques to evaluate the costs associated with the stages of the engineering process: the domain analysis, conceptualization, implementation etc.

Delphi Method The expert judgement method seems to be appropriate for our goals since large amount of expert knowledge w.r.t. ontologies is already available in the Semantic Web community, while the costs of the related engineering efforts are not. Experts' opinion on this topic can be used to compliment the results of other estimation methods.

Parametric Method Apart from the lack of costs-related information which should be used to calibrate cost estimation formula for ontologies, the analysis of the main cost drivers affecting the ontology engineering process can be performed on the basis of existing case studies on ontology building, representing an important step toward the elaboration of a predictable cost estimation strategy for ontology engineering processes. The resulting cost model has to be constantly refined and customized when cost information becomes available. Nevertheless the definition of a fixed spectrum of cost factors is important for a controlled collection of existing real-world project data, a task which is fundamental for the subsequent model calibration.

To summarize, *top-down, parametric and expert-based methodologies* can be partially used to develop a cost estimation process for ontologies. Due to the incompleteness of the information related to cost issues, a combination of the three methodologies is likely to overcome certain limitations of single methodologies. A bottom-up or an analogy approach would be relevant after defining and testing a higher-level cost estimation model and acquiring relevant cost information from real-world projects.

2.1 Ontology vs. Software Engineering

The overall process of developing a cost model for ontologies is not different from the process for estimating any other element of cost. In particular, the similarities between the ontology and software engineering processes make software cost estimation methods a reasonable starting point for the development of a cost estimation methodology for ontologies. In the following we account for the similarities and differences between these disciplines in order to establish the appropriateness of applying software cost models to ontology engineering.

Software and Ontology Engineering belong to the general area of Engineering, a discipline which is usually defined by the usage of scientific principles or methods to construct artifacts. In the first case the artifact is a software

system, while the latter aims at building ontologies, which can be deployed as such by ontology users or further embedded in software systems (the so-called ontology-based applications).

Software Engineering is a central discipline in computer science that provides methods to handle the complexity of developing software systems. It offers a systematic approach to the development, implementation and maintenance of software[1]. Software Engineering is also concerned with the practical problems of developing software systems. Software is not just the programmed code, the software product; it also includes all the associated work, from requirements to test specifications to deployment and maintenance. Several decades of research and development in Software Engineering showed that software projects must also focus on user requirements, design, testing, documentation, deployment and maintenance.

In a similar manner the young discipline of Ontology Engineering includes *scientific methods to create, deploy and maintain ontologies*. Though both fields share the same final goal – the generation of high-quality artifacts – and a similar development process (see Table 1), ontology engineering methodologies available so far do not address every phase of the the underlying process at the same level of detail and do not provide a fully-fledged tool environment to aid the engineering team in their attempt to create, maintain or use ontologies.

Software Engineering	**Ontology Engineering**
System analysis (requirements specification, reuse)	Domain analysis (requirements specification, knowledge acquisition)
System design (architecture)	Conceptualization (conceptual model)
Implementation (program code)	Implementation (specification of the conceptual model)
Test data generation	Ontology population
System testing (refinements)	Ontology evaluation (refinements)
System maintenance	Ontology maintenance

Table 1: Software Engineering vs. Ontology Engineering

Engineering software or ontologies is confronted with the same problems:

- requirements are often ill-defined and subject to frequent changes
- changes to the requirements specification and design have side-effects
- major costs are associated with the design/conceptualization phase and not with the implementation
- there is not agreed means to assure quality

[1]IEEE STD 610.12: IEEE Standard Glossary of Software Engineering Terminology

The main difference between the two engineering processes focuses on their outcomes: software vs. ontologies. A software system is a collection of computer programs and associated data and computer equipment that performs a specific function or purpose. It consists of interrelated software components which exchange data by means of interfaces and *produce a certain result or behavior given user-defined inputs*. An ontology specifies a commonly agreed conceptualization of a given universe of discourse. It is a collection of ontological primitives, usually concepts or classes connected to each other by relations and constrained by axioms, which may belong to specific sub-ontologies. Consequently the way to measure the size of a software system, usually expressed in lines of code or function/object points, can not be directly applied to ontologies. Due to the fact that the implementation of an ontology is mostly realized using tools (i.e. ontology editors), the main size factor to express the complexity of an ontology is not given by the actual size of an implementation in a specific representation language, but by the number of ontological primitives contained by conceptual model. Further on, in comparison to a software system, an ontology may be used in a twofold manner: it may be embedded in a software system for a specific purpose (e.g. to index domain-specific documents) or it may be used directly by domain experts without any mediator in form of some computer program (e.g. as a commonly agreed vocabulary). Nevertheless an ontology does not show any behavior—it does not produce an output for a given input as for a software system. This aspect has implications in the way software and ontologies are evaluated. While the evaluation of software (in the sense of software testing) is a mature discipline despite the difficulties in achieving a commonly agreed, general purpose software quality model, evaluating ontologies is still an open issue both from a technical and a user-centered point of view. Another important difference between software and ontology engineering is related to the particularities of the engineering process itself. While software implementation is one of the major cost driver in software engineering, the most challenging task when building an ontology is the conceptualization phase. Further on, the ontology population/instantiation, a particularity of ontology engineering vs. the related software field, is often associated with significant efforts depending on the particularities of the data to be aligned to the ontology. The project team involved in the construction of an ontology is relatively small compared to the common team size in software development. As a consequence of the importance of the conceptualization phase, the role of the user/domain expert in the ontology construction and his experience in the domain to be modeled are crucial factors for the success of ontology engineering projects. Finally, as for the knowledge of the authors, given the present state of the art of the ontology engineering area, the duration of ontology engineering projects is significantly shorter than the typical duration of software projects.

Despite these differences a cost estimation model for ontologies should benefit from the similar properties of the two engineering fields. The efforts associated with the engineering process still depend on the complexity of the resulting artifact i.e. in the size of the ontology to be modeled and in its functionality, and in the size of the software product respectively[5, 21]. Therefore a cost

model for ontology engineering should take into account the results achieved by the software engineering community and customize and extend them in order to cover the particularities of the ontology development task. In the following section we describe COCOMO II[3] as the main exponent of a comprehensive list of software cost estimation models in order to identify the ways it can be re-used for ontology engineering purposes. Detailed overviews and evaluations on this topic can be found in [5, 21, 8].

2.2 COCOMO II

COCOMO, acronym for Constructive Cost Model, is a simple method to estimate the costs i.e. the person months, man hours or duration arising in a software project. The original COCOMO model - COCOMO 81 - was first developed in 1981 by Barry W. Boehm[5]. COCOMO 81 provides cost estimation procedures for three classes of software: applications, program systems and embedded systems. The cost model became a standard tool in industrial software projects of the eighties. However, since the model was built especially for the cost prediction involved in the realization of largely custom and build-to-specification software, it soon became obsolete as new engineering methods and software system types arose[3]. A new version - COCOMO II - was developed by the end of the nineties in order to cope with the new requirements:

- new software life cycle processes
- software component reuse
- software re-engineering
- object oriented programming
- standard middleware usage

COCOMOII is intended as a cost model for both resource and time estimation. According to its authors it provides a flexible model calibration depending on user-defined requirements and abilities using understandable assumptions and definitions which avoid expensive misunderstandings. It can be customized to specific organizational settings in order to improve the reliability of its prediction.

Currently COCOMO II is one of the most popular empirical estimation techniques and, like its predecessor COCOMO 81, follows the openness principles (relationships, algorithms and interfaces are publicly available). The major reorganization of COCOMO arose out of the need to represent the effects of the new generation of software approaches. This rethinking helps to ensure that the results are applicable to all major business sectors of software development. The development of the new COCOMO was based on the model of the software practices marketplace (see Table 2).

The developers realized that the end-user programming aspect of the software practice marketplace model does not need a COCOMO model; instead a

End-User Programming		
Application generators and composition aids	Application composition	System integration
Infrastructure		

Table 2: Software practices marketplace model

simple activity-based estimate was sufficient. The application composition part of the marketplace is based on **object points**, i.e. a count of the screens, reports and third-generation language modules developed in the application. The other three sectors (application generator, system integration and infrastructure) are based on the mix of an **application composition model**, an **early design model** and a **post-architecture model**. Consequently the new COCOMO offers estimating capability at three levels of granularity to capture the three stages of the software development process, and provides three levels of precision models (see Table 3).

State	**Estimation model**	**Estimation basis**
pre-project / prototyping	application composition	object points
early design	early design	source statements or function points with 7 cost drivers
development	post-architecture	source statements or function points with 17 costs drivers

Table 3: COCOMO 2.0

The model assumes that the costs involved in a software project are significantly influenced by the size of the software product, usually expressed as the number of lines of code (KDSI- Kilo lines of delivered source instruction), and by a particular set of empirically determined cost drivers. The effort is expressed in **person months** (**PM**) i.e. the amount of time that one person spends on software development in a month. The starting point for the calculation is the average value $PM_{nominal}$ (nominal person months), which is further refined with a special increment for maintenance and reuse. Additionally COCOMO II specifies 22 cost drivers (7 in the Early Design and 17 in the Post-Architecture model respectively) which should be considered in the calculation of the nominal person months.

In the following we briefly describe the basic cost calculation principles in COCOMOII by means of the corresponding formulas:

Application Composition Model : intended for the application composition part of the marketplace, focuses on the positive and negative effects of using development tools in the software development process. In this part of the software development process the result depends on the so-

called **New Object Points - NOP** (points count adjusted for reuse) and the **Productivity Rate - PROD**.

$$PM = \frac{NOP}{PROD} \tag{1}$$

where

$$NOP = \frac{(object\_points) * (100 - \%Reuse)}{100} \tag{2}$$

where

- $\%Reuse$ - the percentage of screens, reports s.o. reused from previous applications pro-rated by degree of reuse;
- $PROD$ - depends on developers' experience and capability, adapting values from very low (4) to very high (50).

Early Design Model : used in the early phases of software development that are characterized by poor knowledge about the size of the product to be developed, the nature of the individuals involved in the creation process and the nature of the platform used. It is based on function points which measure software by quantifying the information processing functionality associated with the major external data or control input, output or file types [3]. The function-points costs estimation approach is based on the degree of functionality in a software and a set of individual project factors. For the combined Early Design cost drivers, the numerical values of the contributing Post-Architecture cost drivers are added together with the resulting totals allocated to an expanded Early Design model rating scale ranging from extremely low to extremely high (see Table 4).

The associated formulas apply both to the Early Design and to the Post-Architecture model:

$$PM = A * [Size']^B * \prod_{i=0}^{7} EM_i + PM_M \tag{3}$$

where

$$PM_M = \frac{ASLOC * \frac{AT}{100}}{ATPROD} \tag{4}$$

$$B = 1,01 + 1,01 * \sum_{j=1}^{5} SF_j \tag{5}$$

Early Design Cost Drivers	Post-Architecture Cost Drivers
RCPX-product reliability and complexity	**RELY**-required software reliability **DATA**-database size **CPLX**-product complexity **DOCU**-documentation match to life-cycle needs
RUSE-required reuse	**RUSE**-required reuse
PDIF-platform difficulty	**TIME**-execution time **STOR**-main storage constraint **PVOL**-platform volatility
PERS-personnel capability	**ACAP**-analysis capability **PCAP**-programmer capability **PCON**-personnel capability
PREX-personnel Experience	**AEXP**-application experience **PEXP**-platform experience **LTEX**-language and tool experience
FCIL-facilities	**TOOL**-use of software tools **SITE**-multisite development
SCED-schedule	**SCED**-schedule

Table 4: Early Design and Post-Architecture Cost Drivers

$$Size' = Size * (1 + \frac{BRAK}{100}) \tag{6}$$

$$Size = KNSLOC + KASLOC * (\frac{100 - AT}{100}) * AAM \tag{7}$$

$$AAM = \begin{cases} \frac{AA+AAF*(1+0/02(SU)(UNFM)}{100}, & \text{for } AAF \leq 0,05 \\ \frac{AA+AAF+SU*UNFM}{100}, & \text{for } AAF \geq 0,05 \end{cases} \tag{8}$$

where

AA percentage of reuse effort due to Assessment and Assimilation

AAF Adaptation Adjustment Factor

AAM Adaptation Adjustment Multiplier

ASLOC Adapted Source Lines of Code

AT Automated Translation

ATPROD Automatic Translation Productivity

KASLOC Thousands of Equivalent Source Lines of Code

BRAK Breakage. The amount of controlled change allowed in a software development before requirements are "frozen"

KNSLOC Thousands of Adapted Source Lines of Code

SF Scale Factors

SU Percentage of reuse effort due to Software Understanding

UNFM Programmer Unfamiliarity

Post-Architecture Model : used between the development of the software life-cycle architecture and the implementation phase. In this case the function points used in the Early-Design model are converted to lines of code, which are defined as logical source statements. The goal is to measure the amount of intellectual work put into program development, while difficulties arise when trying to define consistent measures across different implementation languages [3]. The cost drivers used in this model are divided into four categories: product, platform, personnel, and project (see Table 4).

3 ONTOCOM: An Ontology Cost Model

For the development of a cost model for ontologies we adopt a combination of three estimation methodologies, which are in our opinion applicable to ontology engineering according to the current state of the art in the field (see Section 2) . Apart from expert judgement, we start with a top level approach, by identifying upper-level sub-tasks of the ontology engineering process and define the associated costs using a parametric method. We distinguish among three areas, whose costs are to be defined separately:

Ontology Building includes all sub-tasks mentioned in Table 1 excepting the re-usage of available knowledge sources (termed as knowledge acquisition) and the ontology maintenance.

Ontology Maintenance involves costs related to getting familiar and updating the ontology.

Ontology Reuse accounts for the efforts related to the re-usage of existing (source) ontologies for the generation of new (target) ontologies and involves costs related to finding, evaluating and adapting the former ones to the requirements of the latter.

This upper-level distribution is of course subject of future refinements in order to increase the usability of the estimation method in real-world engineering projects. In particular, the ontology building area should be elaborated in the same top-down manner in order to partition this tedious and complex process down to a level in which the associated efforts can be reliably predicted. In this

case, the cost drivers relevant the overall ontology building process (see below) are to be aligned to the corresponding sub-phases and activities. [2]

As a consequence estimating the effort (in person months) related to ontology engineering is reduced to a sum of the costs arising in the building (with or without reuse) and maintaining ontologies:

$$PM \quad = \quad PM_B + PM_M + PM_R, \tag{9}$$

where PM_B , PM_M and PM_R represent the effort associated to building, maintaining and reusing ontologies, respectively.

The costs caused by ontology building are calculated in a COCOMO-similar manner as:

$$PM_x \quad = \quad Size_x * \prod CD_{xi} \tag{10}$$

Each of the three development phases is associated with *specific* cost factors. Experiences in related engineering areas[21, 5] let us assume that the most significant one is the *size of the ontology* involved in the corresponding process. In the formula above the size parameter $Size_x$ is expressed in thousands of ontological primitives – concept, relations, axioms and instances. For example for an ontology with 1000 concepts and 100 relations $Size$ will have the value 1.1. $Size_b$ corresponds to the size of the newly built ontology i.e. the number of primitives which are expected to result from the conceptualization phase. In case of ontology maintenance the size of the ontology ($Size_m$) depends on the expected number of modified items. For reuse purposes the relevant factor $Size_r$ is the size of the original source after being tailored to the present application setting. In particular this involves the parts of the source ontologies which have to be translated to the final representation language, the ones whose content has to be adapted to the target scope and the fragments directly integrated. The *cost drivers* CD_{xi} have a rating level (from very low to very high) that expresses their impact on the development effort. For the purpose of a quantitative analysis, each rating level of each cost driver is associated to a weight (effort multiplier - EM). The average EM assigned to a cost driver is 1.0 (nominal weight). If a rating level causes more development effort, its corresponding EM is above 1.0. If the rating level reduces the effort then the corresponding EM is less than the nominal value. For each cost driver we specified in detail the decision criteria which are relevant when assigning the corresponding effort multipliers (see below).

In the a-priori cost model a team of 3 ontology engineering experts assigned start values between 0.1 and 2 to the effort multipliers, depending on the contribution of the corresponding cost driver to the overall development costs. These parameters were derived after surveying recent literature and from empirical findings of various case studies in the ontology engineering field (such

[2] At this point we restrict to manual ontology building activities. Automatic ontology generation methods as those proposed in the area of ontology learning are not considered in this work yet.

as [27, 39, 33, 2, 1, 37, 38, 23, 4, 6, 35, 29, 14, 41]). These values are subject of further calibration on the basis of the statistical analysis of real-world project data. A list of the initial input values for the cost drivers (the so-called "a-priori cost model") is depicted in Appendix A.

The reuse process and its process measures are examined in Section 5. Maintenance cost estimation is discussed in Section 6

4 Cost Drivers for Ontology Building

Similar to the COCOMOII cost model we differentiate among product, process and personnel cost drivers. The product category accounts for the influence of product properties on the overall costs. The process category states the dimensions of the engineering process which are relevant for the cost estimation, while the personnel one emphasizes the role of team experience, ability and continuity for the effort invested in the process.

4.1 Product Factors

4.1.1 Instance: DATA

The population of an ontology and the associated testing operations might be related to considerable costs[7, 10, 12, 18, 34]. The measure attempts to capture the effect instance data requirements have on the overall process. In particular the form of the instance data and the method required for its ontological formalization are significant factors for the costs of the engineering process (Table 5). On the basis of a survey of ontology population and learning approaches, we

DATA	
Very Low	structured data, same repr. language
Low	structured data with formal semantics
Nominal **High**	semi-structured data e.g. databases, XML semi-structured data in natural language, e. g. similar web pages
Very High	unstructured data in natural langauge, free form

Table 5: Instance Ratings DATA

assume that the population of an ontology with available instance data with an unambiguous semantics can be performed more cost-effective than the processing of relational tables or XML-structured data. Further on, the extraction of ontology instances from poorly structured sources like natural language documents is assigned the highest value magnitude, due to the complexity of the task itself and of the pre-processing and post-processing activities.

The rating does not take into consideration any costs related to eventual mapping operations which might be required to integrate data from external resources. For example, if the data is provided as instances of a second ontology,

be that in the same representation language as the one at hand or not, the estimation of the DATA cost driver should account for the efforts implied by defining a mapping between the source and the target ontology as well. In this case, the parameter is to be multiplied with an increment M (Mapping), as depicted in Table 6 below.

M Increment for DATA	**Ontology Mapping**
0.0	no mapping necessary
0.2	direct mapping
0.4	concept mapping
0.6	taxonomy mapping
0.8	relation mapping
1.0	axiom mapping

Table 6: M Increment for DATA

The M factor increments the effect of the DATA measure: an 1.0 M increment causes a 100% increase of the DATA measure while an 0.0 one does not have any influence on the final value of DATA.

4.1.2 Ontology Complexity: OCPLX

The complexity of the ontology to be built is the major cost factor in the product category. The OCPLX measure is intended to account for those ontology features which increase the complexity of the engineering outcome. After examining the phases of a common ontology building process, we identified three main sources of complexity: the domain analysis, the conceptualization and the instantiation of the ontology. Table 7 illustrates the ratings given for the three categories w.r.t. intrinsic ontology properties like representation language or size. The overall ontology complexity is the averaged weighted sum of the mentioned area.

4.1.3 Required Reusability: REUSE

The measure attempts to capture the effort associated with the development of a reusable ontology. Reusability is a major issue in the ontology engineering community, due to the inherent nature of ontologies, as artifacts for knowledge sharing and reuse. Currently there is no commonly agreed understanding of the criteria required by an ontology in order to increase its reusability. Usually reusability is mentioned in the context of application-independency, in that it is assumed that application-dependent ontologies are likely to imply significant customization costs if reused. Additionally several types of ontologies are often presumed to endue an increased reusability: core ontologies and upper-level ontologies describing general aspects of the world are often used in alignment tasks in order to ensure high-level ontological correctness. The Formal Ontological Analysis of Guarino[17] also mentions 3 levels of generality, which might be

OCPLX	Domain Analysis	Conceptualization	Instantiation
Very Low		concept list	
Low	small size short req. list coarse grained appl. indep.	taxonomy	structured data
Nominal	middle size middle req. list few conflicting req.	properties	semi-structured data
High	large size large req. list many conflicting req. fine grained	instances	unstructured data
Very High		axioms	

Table 7: The Ontology Complexity Ratings OCPLX

associated with different reusability degrees: upper-level ontologies are used as ontological commitment for general purpose domain and task ontologies, while the latter two are combined to realize so-called application ontologies, which are used for particular tasks in information systems. According to these considerations the rating for the REUSE measure is depicted in Table 8.

	Very Low	Low	Nominal	High	Very High
REUSE	for this appl.	for this appl. type	appl. independent domain ont.	core ont.	upper level ont.

Table 8: Reusability Ratings REUSE

4.1.4 Documentation match to lifecycle needs: DOCU

The DOCU measure is intended to state the additional costs caused by detailed documentation requirements. Likewise COCOMOII we differentiate among 5 values from very low (many lifecycle needs uncovered) to very high (very excessive for lifecycle needs) as illustrated in Table 9.

	Very Low	Low	Nominal	High	Very High
DOCU	many LC needs uncovered	some LC needs uncovered	right-sized to LC needs	excessive for LC needs	very excessive for LC needs

Table 9: Ratings for Documentation Costs

4.2 Personnel Factors

4.2.1 Ontologist/Domain Expert Capability: OCAP/DECAP

The development of an ontology requires the collaboration between a team of ontology engineers (ontologists), usually with an advanced technical background, and a team of domain experts that provide the necessary know-how in the field to be ontologically modeled. These cost drivers account the perceived ability and efficiency of the single actors involved in the process, as well as their teamwork capabilities.

	Very Low	**Low**	**Nominal**	**High**	**Very High**
OCAP/DECAP	15%	35%	55%	75%	95%

Table 10: Capability Ratings of the Engineering Team OCAP/DECAP

These approximations are adopted from the corresponding ACAP/PCAP measures in COCOMO, because we assume a homogeneous capability assessment in software and ontology engineering. In comparison to COCOMO, the programmer capability is ignored since it is reduced to the usage of ontology management tools and thus covered by LEXP/TEXP (see Subsection 4.2.3).

4.2.2 Ontologist/Domain Expert Experience: OEXP/DEEXP

These measures take into account the experience of the engineering team consisting of both ontologists and domain experts w.r.t. the ontology engineering process. They are not related to the abilities of single team members, but relate directly to the experience in constructing ontologies and in conceptualizing a specific domain respectively.

	Very Low	**Low**	**Nominal**	**High**	**Very High**
OEXP	2 months	6 months	1 year	1.5 years	3 years
DEEXP	6 months	1 year	3 years	5 years	7 years

Table 11: Experience Ratings for Ontologists and Domain Experts OEXP/DEEXP

The COCOMO experience ratings were modified according to the particularities of the ontology engineering process: while ontology engineering is relatively new in the computer science community (thus maximal rating 3 years), the experience of the domain experts in their particular field is rated according to a different scale (an expert in a certain domain is supposed to have a minimal 3 year experience).

4.2.3 Language and Tool Experience: LEXP/TEXP

The aim of these cost drivers is to measure the level experience of the project team constructing the ontology w.r.t. the conceptualization language and the

ontology management tools respectively. The conceptualization phase requires the usage of knowledge representation languages with appropriate expressivity (such as Description Logics or Prolog), while the concrete implementation is addicted to support tools such as editors, validators and reasoners. The distinction among language and tool experience is justified by the fact that while ontology languages rely on established knowledge representation languages from the Artificial Intelligence field and are thus possibly familiar to the ontology engineer, the tool experience implies explicitly the previous usage of typical ontology management tools and is not directly conditioned by the know-how of the engineering team in the KR field. The maximal time values for the tool experiences are adapted to the ontology management field and are thus lower than the corresponding language experience ratings (see Table 12).

	Very Low	**Low**	**Nominal**	**High**	**Very High**
LEXP	2 months	6 months	1 year	3 years	6 years
TEXP	2 months	6 months	1 year	1,5 years	3 years

Table 12: Language and Tool Experience LEXP/TEXP

4.2.4 Personnel Continuity: PCON

As in other engineering disciplines frequent changes in the project team are a major obstacle for the success of an ontology engineering process within given budget and time constraints. Due to the small size of the project teams we adapted the general ratings of the COCOMO model to a maximal team size of 10 (Table 13).

	Very Low	**Low**	**Nominal**	**High**	**Very High**
PCON	50%	35%	25%	15%	10%

Table 13: Ratings for the personnel turnover PCON

4.3 Project Factors

4.3.1 Support tools for Ontology Engineering: TOOL

The usage of ontology management tools is an essential success factor in every ontology engineering process. Apart from the implementation phase, which is worst-case performed by manually feeding the conceptual model to some ontology editor, the evaluation of the ontology can not be imagined without the utilization of validation and reasoning tools. A few support tools for the domain analysis and the conceptualization phases have also been proposed, but their general acceptance and compatibility to other tools are still an open issue (see Table 14).

	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	reasoner validator	editor	APIs for implement.	tool for ont. population	tool for ont. evolution, moderately integrated	completely integrated tool

Table 14: Ratings for support tools for ontology engineering TOOL

4.3.2 Multisite Development: SITE

Constructing an ontology requires intensive communication between ontology engineers and domain experts on one hand and between domain experts for consensus achievement purposes on the other hand. This measure involves the assessment of the communication support tools (Table 15).

	Very Low	Low	Nominal	High	Very High
SITE	mail	phone, fax	email	teleconference, occasional meetings	frequent F2F meetings

Table 15: Ratings for multisite ontology development SITE

4.3.3 Required Development Schedule: SCED

This cost driver takes into account the particularities of the engineering process given certain schedule constraints. Accelerated schedules (ratings below 100%, see Table 16) tend to produce more efforts in the refinement and evolution steps due to the lack of time required by an elaborated domain analysis and conceptualization. Stretch-out schedules (over 100%) generate more effort in the earlier phases of the process while the evolution and refinement tasks are best case neglectable.

	Very Low	Low	Nominal	High	Very High
SCED	75%	85%	100%	130%	160%

Table 16: Required Development Schedule SCED

For example, a high SCED value of 130% (Table 16) represents a stretch-out of the nominal schedule of 30% and thus more resources in the domain analysis and conceptualization.

5 Extensions of ONTOCOM for Reuse

Though there is yet no fine-grained methodology to reuse existing ontologies in the Semantic Web community, the main steps and the associated challenges involved in the process are well-accepted by current ontology-based projects[30, 39]. This process is, however, related to significant costs and efforts, which may currently outweigh its benefits. First, as in other engineering disciplines, reusing some existing component implies costs to find, get familiar with, adapt and update the necessary modules in a new context. Second building a new ontology means partially translating between different representation schemes or performing scheme matching or both.

For our cost estimation model we assume that relevant ontologies are available to the engineering team and, according to the mentioned top-level approach and to some case studies in ontology reuse[27, 28, 33, 38, 1] we examine the following two phases of the reuse process w.r.t. the corresponding cost drivers:

- ontology evaluation: get familiar with the ontology and assess its relevance for the target ontology
- ontology customization: translate the sources to a desired format, eventually extract relevant sub-ontologies and finally integrate them to the target ontology

For the evaluation phase the engineering team is supposed to assess the relevance of a given ontology to particular application requirements. The success of the evaluation depends crucially on the extent to which the ontology is familiar to the assessment team. The customization phase implies the identification/extraction of sub-ontologies which are to be integrated in a direct, translated and modified form, respectively. In the first categories sub-ontologies are included directly to the target ontology. The re-usage of the second category is conditioned by the availability and the appropriate costs of knowledge representation translators, while the last category involves modifications of the original model in form of insertions, deletions or updates at the ontological primitives level.

5.1 Cost Drivers for Ontology Reuse

5.1.1 Ontology Understandability: OU

Reusing an ontology and the associated efforts depend significantly on the ability of the ontologists and domain experts to understand the ontology, which is influenced by two categories of factors: the complexity of the conceptual model and the self-descriptiveness or the clarity of the conceptual model. Additionally, in case of the ontology engineer the comprehensiveness of an ontology depends on his domain experience, while domain experts are assumed to provide this know-how by definition.

Factors contributing to the complexity of the model are the size and expressivity of the ontology and the number of imported models together with the

complexity of the import dependency graph. The clarity of the model is mainly influenced by the human-perceived readability.

	Very Low	**Low**	**Nominal**	**High**	**Very High**
Complexity	complex DG large ont. complex RL	taxon. DG large ont. complex RL	taxon. DG middle ont. moderate RL	no imports middle ont. simple RL	no imports small ont. simple RL
Clarity	no concept names, RL know-how no comm. no metadata	concept names, RL know-how no comm. no metadata	concept names, RL tool 30 % comm. no metadata	concept names, RL tool 60% comm. no metadata	concept names, RL tool 90% comm. metadata

Table 17: Complexity and Clarity Levels for Ontology Understanding

The complexity of the ontology depends on three factors: the size of the ontology, the expressivity of used representation language and the structure of the import graph – containing imported ontologies. The import graph structure (DG - dependency graph) can be divided into simple, as in taxonomical tree structures and complex, as in non-tree structures. Further on, the complexity of the used syntax (RL in Table 17) is termed to be simple for common taxonomical hierarchies, moderate if further property types are used and complex in the case of restrictions and axioms. The third ontology complexity driver is related to the size of the ontology: small ontologies are supposed to contain up to 100 ontological primitives, middle ontologies contain up to 1000 concepts, while ontologies with more than 1000 concepts are classified as large in our model (see Table 17)

The clarity categorization depends on the readability/meaningfulness of ontological primitives, the technical know-how required by the representation language and the availability of natural language comments (comm.) and definitions. The understandability of an ontology can be increased significantly when ontological primitives are given meaningful names in a natural language which is familiar to the ontology engineer and the domain expert respectively. Further on, a self-descriptive representation language does not cause significant impediments in dealing with an ontology, especially when user-friendly tools are available.

5.1.2 Ontologist/Domain Expert Unfamiliarity: UNFM

The effort related to ontology maintenance decreases significantly in situations where the human user works frequently with the particular ontology. This measure accounts for this dependency and distinguishes among 6 levels as depicted in Table 18.

The UNFM factor increments the effect of the Ontology Understanding measure: an 1.0 UNFM increment causes a 100% increase of the OU measure while an 0.0 one does not have any influence on the final value of OU (see Table 18).

OU Increment for UNFM	Level of Unfamiliarity
0.0	self built
0.2	team built
0.4	every day usage
0.6	occasional usage
0.8	little experience
1.0	completely unfamiliar

Table 18: OU Increment for UNFM

5.1.3 Ontology Evaluation: OE

This measure accounts for the real effort needed to evaluate the ontology for reuse purposes (see Table 19). The measure assumes a satisfactory ontology understanding level and is associated solely with the efforts needed in order to decide whether a given ontology satisfies a particular set of requirements and to integrate its description into the overall product description.

Rating Scale for the OE Increment	Level of activity required
Very Low	none
Low	basic documentation
Nominal	some testing and documentation
High	considerable testing and documentation
Very High	extensive testing and documentation

Table 19: Rating Scale for the Ontology Evaluation Increment

5.1.4 Ontology Modification: OM

This measure reflects the complexity of the modifications required by the reuse process after the evaluation phase has been completed.

	Very Low	Low	Nominal	High	Very High
OM	few, simple modifications	some simple modifications	some moderate mod.	considerable modifications	excessive modifications

Table 20: Efforts for modification

5.1.5 Ontology Translation: OT

Translating between knowledge representation languages is an essential part of a reuse process. Depending on the compatibility of the source and target

representation languages, as well as on the availability and performance of the translating tools (amount of pre- and post-processing required)[20, 9], we differentiate among 5 values as depicted in Table 21.

	Very Low	Low	Nominal	High	Very High
OT	direct	low manual effort	some manual effort	considerable manual effort	manual effort

Table 21: Efforts for translation

5.1.6 Reuse formula

The cost drivers mentioned above flow in the person months calculation of the reuse process as follows:

$$PM_R = Size_R * \prod CD_i \tag{11}$$

The reused size is divided into the size of the directly integrated, translated and modified components with different cost drivers.

$$\begin{aligned} Size_R \quad = \quad & Size_{dir} * (OU * UNFM + OE) + \\ & Size_{trans} * (OU * UNFM + OE + OT) + \\ & Size_{mod} * (OU * FM + OE + OM) + \\ & Size_{transmod} * (OU * UNFM + OE + OT + OM) \end{aligned} \tag{12}$$

6 Extensions of ONTOCOM for Maintenance

The maintenance cost model is similar to its reuse counterpart, but it does not take into consideration translation and evaluation costs. The maintenance process still implies efforts to get familiar with the ontology as well as the real costs of the update, insert or delete operations. We obtain the following computation formula for the maintenance person months:

$$PM_M \quad = \quad Size_M * \prod CD_i \tag{13}$$

where $Size_M$ is the sum of the added and modified ontology fragments, which are influenced by the OU, $UNFM$ and OM cost factors as follows:

$$\begin{aligned} Size_M \quad = \quad & Size_{added} * OU * UNFM + \\ & Size_{modified} * (OU * UNFM + OM) \end{aligned} \tag{14}$$

7 Evaluation

The parametric approach described in this report is subject of further refinements towards a reliable method for estimating the costs of ontology engineering. The most important evaluation criterium is of course the reliability of its predictions, which depends on the amount of accurate project data used to calibrate the model i.e. to adjust the values of the modifiers and to identify eventual correlated cost drivers. However, a comprehensive evaluation of the model should go beyond the evaluation of its functionality (i.e. the accuracy of its estimations) and address issues related the quality of its usage in typical ontology engineering scenarios, as suggested in common quality frameworks for information systems (such as [31, 25, 19, 22]; see [11] for a comprehensive survey on this topic).

For a comprehensive evaluation of the model we rely on the quality framework for cost models by Boehm, which is a list of required and desirable features for these category of models, showing many similarities with established frameworks for evaluating information or data models[11]. For our purposes we adapted this framework to the particularities of ONTOCOM and ontology engineering and used parts of its in order to assess the quality of the a-priori and the a-posteriori cost models, respectively.

The evaluation of the cost model is intended to be performed in two steps. In the first one we evaluated the relevance of the mentioned cost drivers for the purpose of predicting costs arisen in ontology engineering projects. The remaining aspects of the framework relate to its capability of reliably fulfilling its goal (i.e. that of estimating engineering efforts) and will be applied in a second step on the a-posteriori model resulting from the calibration of the preliminary one.

The original quality framework by Boehm[5] consisted of the 10 features, which we divided these features into two categories, depending on their relevance to the a-priori and the a-posteriori model, respectively. The meaning of the quality criteria has been adapted to the scope of ONTOCOM.

a-priori evaluation

definition has the model clearly defined the costs it is estimating and the costs it is excluding? Does the estimate include the cost of management, training, domain analysis, conceptualization, implementation, testing, maintenance? Does the model clearly define the decision criteria used to specify the ratings of the cost drivers? Does the model use intuitive and non-ambiguous terms to denominate the cost drivers it involves?

objectivity does the model avoid allocating most of the cost variance to poorly calibrated subjective factors? Are the cost drivers defined using objective decision criteria, which allow an accurate assignment of the corresponding cost driver ratings?

constructiveness can a user tell why the model gives the estimates it does?

detail does the model easily accommodate the estimation of new process models or is it conceived for a particular ontology engineering process? Does it give accurate phase and activity breakdowns? Does the model take into consideration factors related to the main tasks of the engineering process? Do these sub-tasks correspond to the process model applied in your engineering process? Which phases should be further covered by the model in order to increase its usability?

stability do small differences in inputs produce small differences in output cost estimates?

scope does the model cover the class of projects whose costs you want to estimate? Is it representative for a wide class of ontology engineering projects?

ease of use are the model inputs and options easy to understand and specify? Is it easy for you to assess a rating to a cost driver based on the associated decision criteria?

prospectiveness does the model avoid the user of information which will not be well known until the project is complete? Can the model be applied in early phases of the project as well?

parsimony does the model avoid the use of highly redundant factors or factors which make no appreciable contribution to the results?

a-posteriori evaluation

all items of the former category, plus

fidelity, since this requirement will definitely not be fulfilled after collecting reliable data from previous projects used to refine the values of the cost drivers and to discover eventual correlations between them.

The a-priori evaluation is currently being performed on the basis of a two-phased structured process, in which domain experts from industry and academia are interviewed on these topics. The evaluation activity is monitored by two ontology engineers, with a solid background in the field and deep knowledge of the cost estimation model to be validated. The interviewed expert team consists of approximately 10 members with similar knowledge background in the engineering field, who were trained in advanced in order to understand the main ideas of the ONTOCOM model. The results of the first phase of the evaluation, are currently collected and analyzed by the team conducting the study. They will be used to trigger the second phase of the Delphi process, striving for the achievement of a common opinion on the modifications the preliminary model should be subject of. The final analysis of the interviews will result in a new version of the a-priori cost model, whose content has been validated according to the judgement of the expert team. The a-posteriori evaluation of the model

will be performed in a similar manner, but it depends on the availability of historical project data used to calibrate the parametric equations underlying the model.

8 Conclusions and Future Work

This report aims at developing a prototypical cost estimation model for the ontology engineering area. The model is intended to predict cost arisen during ontology engineering processes by analyzing the costs factors caused by the end product, the process itself and the engineering team. Starting from an analysis of general-purpose estimation methodologies and of the most prominent exponent in the area of software engineering, COCOMO, we propose a methodology to deduce ontology costs and examine costs factors implied by ontology building, reuse and maintenance.

In order to establish a cost estimation model we apply a composite methodology starting with a top-level analysis of the engineering steps, whose costs are at first predicted using a COCOMO-like parametric method and are to be empirically validated using expert judgement. At present we defined a preliminary parametric model, which is to be calibrated on the basis of real project data and refined by engineering experts.

Acknowledgements This work is a result of the cooperation within the Semantic Web PhD-Network Berlin-Brandenburg and has been partially supported by the KnowledgeWeb - Network of Excellence, by the Project A Semantic Web for Pathology" funded by the DFG (German Research Foundation) and by the Knowledge Nets" project, which is part of the InterVal- Berlin Research Centre for the Internet Economy, funded by the German Ministry of Research BMBF.

A Non-calibrated Values of the Cost Drivers in ONTOCOM

The initial input values for the cost drivers in the ONTOCOM model are illustrated in Tables 22 and 23.

Cost Drivers	Rating				
	Very Low	Low	Nominal	High	Very High
DATA	0,80	0,90	1	1,30	1,60
REUSE	0,70	0,85	1	1,15	1,30
DOCU	0,70	0,85	1	1,15	1,30
OCPLX	0,70	0,85	1	1,30	1,60
OCAP	1,30	1,15	1	0,85	0,70
DECAP	1,30	1,15	1	0,85	0,70
OEXP	1,30	1,15	1	0,85	0,70
DEEXP	1,30	1,15	1	0,85	0,70
LEXP	1,60	1,30	1	0,90	0,80
TEXP	1,50	1,25	1	0,90	0,80
PCON	1,30	1,15	1	0,85	0,70
TOOL	1,60	1,30	1	0,90	0,80
SITE	1,30	1,15	1	0,85	0,70
SCED	1,30	1,15	1	0,85	0,70

Table 22: Cost Drivers for Ontology Building and their ratings

Cost Drivers	Rating				
	Very Low	Low	Nominal	High	Very High
OU	1,80	1,40	1	0,90	0,80
OE	0,70	0,85	1	1,30	1,60
OM	0,80	0,90	1	1,20	1,40
OT	0,70	0,85	1	1,30	1,60

Table 23: Cost Drivers for Ontology Reuse/Maintenance and their ratings

B Cost Model Validation - Questionnaire

B.1 Part 1 - General questions about the ontology engineering process

- Which were the most challenging aspects of the ontology engineering process according to your experiences and why? (Table 24)?
- Which phase of the ontology engineering process (domain analysis, conceptualization, implementation, instantiation, evaluation & refinement or maintenance) was the most complex and time and resource intensive? Why?

Engineering Phase	Difficulties encountered
Domain analysis (requirements specification knowledge acquisition)	
Conceptualization (conceptual model)	
Implementation (specification of the conceptual model)	
Instantiation	
Evaluation & Refinement	
Maintenance	

Table 24: Challenging aspects

- Which cost factors (e.g. team experience and know-how in the ontology representation language and tools) were relevant in each phase of ontology engineering process (Table 25)?

Engineering Phase	Cost factors
Domain analysis (requirements specification knowledge acquisition)	
Conceptualization (conceptual model)	
Implementation (specification of the conceptual model)	
Instantiation	
Evaluation & Refinement	
Maintenance	

Table 25: Relevant factors

- When would you characterize an existing ontology as complex?
- Which factors affect the understandability of an existing ontology? In your opinion when makes an existing ontology difficult to understand?
- The ontology engineering process requires the cooperation between ontology and domain experts. What was the level of experience of the team with respect to engineering ontologies in your case? Please mark the right answer(s). Please mark the right answer(s) in the Table 26.

Experience	**Ontology engineer**	**Domain expert**
Theoretical knowledge about ontologies and ontology engineering process (they have not built an ontology themselves yet)		
Practical knowledge (they have already worked with /developed ontologies)		
No experiences with ontologies		
Domain knowledge		

Table 26: Level of Experience

B.2 Part 2 - Questions to pre-selected costs factors

- 2.1. How do you judge the importance of the following factors with respect to the costs (time and resources) arisen during the process of ontology development? Please mark the right answer (Tab. 27, 28, 29).

Product Factors				
	Not relevant	Relevant	Very relevant	No opinion
Amount of instance data/ complexity the instantiation (DATA) *Explanation*: this factor concerns the form of the instance data and the method used for the extraction of ontology instances and their integration to the ontology				
Required reusability of the ontology (REUSE) *Explanation*: this factor attempts to capture the additional effort associated with the development of a reusable ontology				
Documentation match to lifecycle needs (DOCU) *Explanation*: this factor states the additional costs caused by detailed documentation requirements				
Ontology Complexity (OCPLX) *Explanation*: this factor is intended to account for those ontology features which increase the complexity of the engineering outcome (e.g. large domain, large requirement list with many conflicting ones, existence of instances)				

Table 27: Product Factors

Personnel Factors				
	Not relevant	**Relevant**	**Very relevant**	**No opinion**
Ontologist Expert Capability (OCAP) *Explanation*: this factor is related to the average level of technical of the ontology engineer (ontologist).				
Domain Expert Capability (DECAP) *Explanation*: this factor is related to the average level of domain knowledge of domain expert.				
Experience of the ontologist with ontologies and their development (OEXP) *Explanation*: this factor takes into account the experience of the team of ontologists w.r.t. the ontology engineering process.				
Experience of the domain expert with ontologies and their development (DEEXP) *Explanation*: this factor takes into account the previous experience of the domain experts' team w.r.t. the ontology engineering process.				
Experience of the team with ontology representation languages (LEXP) *Explanation*: this factor measures the level experience of the project team constructing the ontology w.r.t. the ontology representation language.				
Experience of the team with ontology engineering tools (TEXP) *Explanation*: this factor measures the level experience of the project team constructing the ontology w.r.t ontology management tools (e.g. editors, validators).				
Continuity of the personnel (PCON) *Explanation*: this factor expresses the additional costs caused by frequent changes in the project team.				

Table 28: Personnel Factors

Project Factors				
	Not relevant	Relevant	Very relevant	No opinion
Availability of the support tools for ontology engineering (TOOL) *Explanation*: this factor takes into account the influence the usage of ontology management tools (editors, validators, tools for ontology population or evaluation, etc.) has on the overall costs.				
Multi-site development(SITE) *Explanation*: this factor involves the additional costs arisen in a project in which the engineering team is located at different sites.				
Required development schedule with deadlines (SCED) *Explanation*: this factor takes into account the cost implied by particular time schedule constraints. Schedule constraints influence the effort invested in single phases of the engineering process.)				

Table 29: Project Factors

- Which of the above listed cost factors (Tables 27, 28, 29) are in your opinion overlapping?
- Which of the above listed cost factors (Tables 27, 28, 29) should be renamed and how?
- Which other relevant cost factors do you miss in the lists above (Tables 27, 28, 29)?
- How do you judge the importance of the following factors with respect to the costs (time and resources) arisen during the process of reusing existing ontologies? Please mark the right answer (Table 30).

Reuse Factors				
	Not relevant	Relevant	Very relevant	No opinion
Ontology Understanding (OU) *Explanation*: this factor expresses the ability of the ontologists and domain experts to understand the ontology (how well a given ontology can be understood by the engineering team re-using it). It is influenced by two categories of factors: the complexity of the conceptual model (which depends on existence of imports, size of the ontology, expressivity of used representation language) and the self-descriptiveness or the clarity of the conceptual model (which depends on readability/meaningfulness of ontological primitives, availability of metadata, etc.))				
Ontologist/Domain Expert Unfamiliarity (UNFM) *Explanation*: this factor increments the impact of the OU cost driver and it expresses the level of unfamiliarity of the engineering team w.r.t. the ontology to be reused (did the team build themselves? does the team work with it every day? does the team have only little experience with this ontology?				
Ontology Evaluation (OE) *Explanation*: this factor expresses the real effort needed to evaluate the ontology for reuse purposes including different level of documentation and different level of testing required to assess its relevance in a certain context.				
Ontology Modification (OM) *Explanation*: this factor accounts for the complexity of the modifications required by the reuse process after the evaluation phase has been completed (e.g. simple modification, excessive modification)				
Ontology Translation (OT) *Explanation*: this factor expresses the effort invested in the translation between knowledge representation languages of the source and target ontology.				

Table 30: Reuse Factors

- Which of the above listed cost factors (Table 30) are, in your opinion, overlapping?
- Which of the above listed cost factors (Table 30) should be renamed and how?
- In your opinion, which other relevant cost factors (Table 30) have not been listed above?
- How do you judge the importance of the following factors with respect to the complexity of an ontology? (Table 31)?

Factors	**Not relevant**	**Relevant**	**Very relevant**	**No opinion**
Size of the ontology/ Number of ontological primitives				
Expressivity of the used representation language				
Structure of the import graph of an ontology (i.e. the graph consisting of external ontologies, which are imported in the first one)				

Table 31: Importance of factors

- In your opinion, which other relevant factors have not been listed in Table 31?

Cost Driver	Rating (Value)
DATA	High (1,30)
REUSE	Nominal (1)
DOCU	Low (0,85)
OCPLX	High/Very High (1,40)
OCAP	High (0,85)
DECAP	Low (1,15)
OEXP	High (0,85)
DEEXP	Very Low (1,30)
LEXP	Nominal (1)
TEXP	Nominal (1)
PCON	Very High (0,70)
TOOL	Very Low (1,60)
SITE	Nominal (1)
SCED	Nominal (1)

C Using ONTOCOM: An Example

In this section we give a brief example on the usage of the ontology cost model described in this report. Starting from a typical ontology building scenario, in which a domain ontology is created from scratch by the engineering team, we simulate the cost estimation process according to the parametric method underlying ONTOCOM. Given the top-down nature of our approach this estimation can be realized in the early phases of a project, in particular after the domain analysis has been accomplished and an initial prediction of the size of the target ontology is available.

The first step of the cost estimation is the specification of the size of the ontology to be build, expressed in thousands of ontological primitives. Ontological primitives are concepts, relations (including is-a), axioms and instances. Note that we do not consider the size of the data set which will be used to populate the ontology, but only the instances which are to be conceptualized manually during the conceptualization phase. For example, if we consider an ontology with 1000 concepts, 200 relations and 100 axioms, the size parameter of the estimation formula will be 1,3

The next step is the specification of the cost driver ratings, corresponding to the information available at this point to the engineering team and to the decision criteria assigned to each rating (Section 4). Assuming that the ratings of the cost drivers are those depicted in Table C these ratings are replaced by numerical values (as illustrated in Table 22 in Appendix A).

According to formula 10 the estimated effort in person months would be 2,43PMs:

$$PM = 1,3 * 1,3 * 1 * 0,85^2 * 1,15 * 0,85 * 1,3 * 1 * 1 * 0,7 * 1,6 * 1^2 (15)$$

The value of the OCPLX cost driver was computed as an equally weighted, averaged sum of a high-valued rating for the domain analysis (1,3) , a very high rating for the conceptualization (1,6) and a high effort multiplier for the instantiation phase, respectively:

$$OCPLX = 1 * 1,3 + 1 * 1,6 + 1 * 1,31 + 1 + 1 \quad (16)$$

References

[1] A. Advani, S. Tu, and M. Musen. Domain Modeling with Integrated Ontologies: Principles for Reconciliation and Reuse. Technical Report SMI-97-0681, Stanford Medical Informatics, 1997.

[2] M. Annamalai and L. Sterling. Guidelines for Constructing Reusable Domain Ontologies. In *Proceedings of the OAS*, pages 71–74, 2003.

[3] B. W. Boehm, C. Abts, B. Clark and S. Devnani-Chulani. COCOMO II Model Definition Manual, 1997.

[4] A. Bernaras, I. Laresgoiti, and J. Corera. Building and Reusing Ontologies for Electrical Network Applications. In *Proceedings of the European Conference on Artificial Intelligence ECAI96*, 1996.

[5] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[6] W. N. Borst. Construction of Engineering Ontologies. Technical report, University of Tweenty, Eschede, 1997.

[7] P. Buitelaar, D. Olejnik, and M. Sintek. A Protege Plug-In for Ontology Extraction from Text Based on Linguisitc Analysis. In *Proceedings of the European Semantic Web Symposium ESWS04*, 2004.

[8] S. Chulani. *Incorporating Bayesian Analysis to Improve the Accuracy of COCOMO II and Its Quality Model Extension*. PhD thesis, University of Southern California, 1998.

[9] S. Cranefield. UML and the Semantic Web. In *Proceedings of the 1st Semantic Web Working Symposium SWWS01*, pages 113–130, 2001.

[10] M. Dittenbach, H. Berger, and D. Merll. Improving domain ontologies by mining semantics from text. In *Proceedings of the 1st Asian-Pacific Conference on Conceptual Modelling*, pages 91–100, 2004.

[11] M. J. Eppler. The Concept of Information Quality: An Interdisciplinary Evaluation of Recent Information Quality Frameworks. *Studies in Communication Sciences*, 1:167–182, 2001.

[12] D. Faure and Poibeau T. First experiments of using semantic knowledge learned by ASIUM for information extraction task using INTEX. In *Proceedings of the Ontology Learning Workshop at the ECAI00*, 2000.

[13] M. Fernandez, A. Gomez-Perez, and N. Juristo. Methontology: From ontological art towards ontological engineering. In *Proceedings of the AAAI'97 Spring Symposium on Ontological Engineering*, 1997.

[14] A. Fernndez-Lpez, M.and Gmez-Prez, A. Pazos-Sierra, and J. Pazos-Sierra. Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment. *IEEE Intelligent Systems and their Applications*, pages 37–46, January/February 1999.

[15] M. Fernndez-Lpez and A. Gmez-Prez. Overview and analysis of methodologies for building ontologies. *Knowledge Engineering Review*, 17(2):129–156, 2002.

[16] M. Grüninger and M. Fox. Methodology for the Design and Evaluation of Ontologies. In *Proceedings of the IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[17] N. Guarino. Formal Ontology and Information Systems. In *Proceedings of the International Conference on Formal Ontology and Information Systems FOIS98*, pages 3–15, 1998.

[18] U. Hahn, M. Romacker, and K. Schnattinger. Automatic knowledge acquisition from medical text. In *Proceedings of the 1996 AMIA Annual Symposium*, pages 383 – 387, 1996.

[19] K. T. Huang, Y. W. Lee, and R. Y. Wang. *Quality Information and Knowledge*. Prentice Hall, 1999.

[20] M. Sabou K. Falkovych and H. Stuckenschmidt. *UML for the Semantic Web: Transformation-Based Approaches*, volume Knowledge Transformation for the Semantic Web. IOS Press, 2003.

[21] C. F. Kemerer. An Empirical Validation of Software Cost Estimation Models. *C-ACM*, 30(5), 1987.

[22] J. Krogstie, O. I. Lindland, and G. Sindre. Defining Quality Aspects for Conceptual Models. In *Proceedings of the IFIP8.1 working conference on Information Systems Concepts ISCO03: Towards a Consolidation of Views*, 1995.

[23] D. B. Lenat and R. V. Guha. *Building large knowledge-based systems*. Addison-Wesley Publising Company, 1990.

[24] A. Maedche and S. Staab. Measuring Similarity between Ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management EKAW02*, 2002.

[25] D. L. Moody, G. Sindre, T. Brasethvik, and A. Solvberg. Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In *Proceedings of the 25th International Conference on Software Engineering ICSE03*, 2003.

[26] National Aeronautics and Space Administration (NASA). NASA Cost Estimating Handbook 2004, 2004.

[27] E. Paslaru Bontas, M. Mochol, and R. Tolksdorf. Case Studies in Ontology Reuse. In *Proceedings of the 5th International Conference on Knowledge Management IKNOW05*, 2005.

[28] E. Paslaru Bontas, M. Mochol, and R. Tolksdorf. Towards a methodology for ontology reuse. In *Proceedings of the International Conference on Terminology and Knowledge Engineering TKE05(to be published)*, 2005.

[29] B. J. Peterson, W. A. Andersen, and J. Engel. Knowledge Bus: Generating Application-focused Databases from Large Ontologies. In *Proceedings of the KRDB*, 1998.

[30] H. S. Pinto and J. P. Martins. A methodology for ontology integration. In *Proceedings of the International Conference on Knowledge Capture K-CAP2001*, pages 131–138. ACM Press, 2001.

[31] R. Price and G. Shanks. A Semiotic Information Quality Framework. In *Proceedings of the International Conference on Decision Support Systems DSS04*, 2004.

[32] R.D. Stewart (Editor) and R.M. Wyskida (Editor) and J.D. Johannes (Editor). *Cost Estimator's Reference Manual*. Wiley, 2 edition, 1995.

[33] T. Russ, A. Valente, R. MacGregor, and W. Swartout. Practical Experiences in Trading Off Ontology Usability and Reusability. In *Proceedings of the Knowledge Acquisition Workshop KAW99*, 1999.

[34] D. Schlangen, M. Stede, and E. Paslaru Bontas. Feeding OWL: Extracting and Representing the Content of Pathology Reports. In *Proceedings of the NLPXML04*, 2004.

[35] F. Sowa, A. Bremen, and S. Apke. Entwicklung der Kompetenz-Ontologie fr die Deutsche Montan Technologie GmbH. http://www.kowien.uni-essen.de/workshop/DMT_01102003.pdf, 2003.

[36] Y. Sure, S. Staab, and R. Studer. On-To-Knowledge Methodology (OTKM). In *Handbook on Ontologies*, pages 117–132. 2004.

[37] B. Swartout, R. Patil, K. Knight, and T. Russ. Toward Distributed Use of Large-Scale Ontologies. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.

[38] M. Uschold, P. Clark, M. Healy, K. Williamson, and S. Woods. An Experiment in Ontology Reuse. In *Proceedings of the 11th Knowledge Acquisition Workshop KAW98*, 1998.

[39] M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods. Ontology Reuse and Application. In *Proceedings of the International Conference on Formal Ontology and Information Systems FOIS98*, pages 179–192, 1998.

[40] M. Uschold and M. King. Towards a Methodology for Building Ontologies. In *Proceedings of the IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[41] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The Enterprise Ontology. *The Knowledge Engineering Review*, 13, 1998.