NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 – "Semantic-based knowledge and content systems"

D5.3.1 NeOn Development Process and Ontology Life Cycle

Deliverable Co-ordinator: Mari Carmen Suárez-Figueroa

Deliverable Co-ordinating Institution: UPM

Other Authors: Guadalupe Aguado de Cea (UPM), Carlos Buil (iSOCO), Caterina Caracciolo (FAO), Martin Dzbor (OU), Asunción Gómez-Pérez (UPM), German Herrrero (ATOS), Holger Lewen (UKARL), Elena Montiel-Ponsoda (UPM), Valentina Presutti (CNR)

This deliverable presents ontology network development process, ontology network life cycle models and ontology network life cycle. The main contributions included in this document are: the NeOn Glossary of Activities, different ontology network life cycle models, guidelines for obtaining the ontology network life cycle for a particular ontology network, and preliminary evaluation results of the notions presented in this document within the three NeOn use cases.

Document Identifier:	NEON/2007/D5.3.1/v1.0	Date due:	August 31, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	August 31, 2007
Project start date:	March 1, 2006	Version:	V1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is a part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta} @open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 28 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 655 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Martino della Battaglia, 44 - 00185 Roma-Lazio, Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 1 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

NeOn

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

ATOS

CNR

iSOCO

FAO

OU

UKARL

UPM

Change Log

Version	Date	Amended by	Changes
0.1	12-06-2006	Mari Carmen Suárez-Figueroa	First draft, including first list of activities and definitions in the NeOn Glossary of Activities
0.2	21-05-2007	Mari Carmen Suárez-Figueroa	Update of the table of content First version of the introduction in the state of the art First version of the process to reach consensus in the NeOn Glossary of Activities
0.3	8-06-2007	Mari Carmen Suárez-Figueroa	First version of the State of the Art
		Mari Carmen Suárez-Figueroa, German Herrero, Caterina Caracciolo	First version of the examples describing NeOn use cases
0.4	18-06-2007	Mari Carmen Suárez-Figueroa	Second version of the State of the Art Second version of the process to reach consensus in the NeOn Glossary of Activities
0.5	22-06-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	First version of the NeOn Ontology Development Process and NeOn Life Cycle
		Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	First version of the about Scenarios for Building Ontology Networks
0.6	3-07-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Second version of the NeOn Ontology Development Process and NeOn Life Cycle
0.7	16-07-2007	Mari Carmen Suárez-Figueroa	Third version of the process to reach consensus in the NeOn Glossary of Activities
0.8	17-07-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Third version of the NeOn Life Cycle
0.9	18-07-2007	Mari Carmen Suárez-Figueroa, Guadalupe Aguado de Cea	Revision of section about the process to reach consensus in the NeOn Glossary of Activities

		Mari Carmen Suárez-Figueroa, Carlos Buil	First version of scenario for the invoice use case
0.10	24-07-2007	Mari Carmen Suárez-Figueroa	Inclusion of the final version of the NeOn Glossary of Activities
0.11	24-07-2007	Mari Carmen Suárez-Figueroa, Rosario Plaza	Revision of the NeOn Glossary of Activities (from a linguistic point of view) Revision of the State of the Art (from a linguistic point of view) Revision of the NeOn Ontology Development Process section (from a linguistic point of view)
0.12	25-07-2007	Mari Carmen Suárez-Figueroa, Rosario Plaza	Revision of the section about Scenarios for Building Ontology Networks (from a linguistic point of view)
0.13	25-07-2007	Mari Carmen Suárez-Figueroa	Introduction section and executive summary. Revision of the NeOn Glossary of Activities
0.14	26-07-2007	Mari Carmen Suárez-Figueroa	Update of the NeOn Life Cycle section
0.15	27-07-2007	Mari Carmen Suárez-Figueroa, Caterina Caracciolo	Second version of scenario for the fishery use case
0.16	13-08-2007	Mari Carmen Suárez-Figueroa, German Herrero	Second version of scenario for the nomenclature use case
0.17	13-08-2007	Mari Carmen Suárez-Figueroa, Holger Lewen	Revision of NeOn Ontology Development Process
0.18	17-08-2007	Mari Carmen Suárez-Figueroa	Third version of the State of the Art
0.19	20-08-2007	Mari Carmen Suárez-Figueroa	Revision of NeOn Life Cycle
0.20	21-08-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of the Introduction and the State of the Art
0.21	22-08-2007	Mari Carmen Suárez-Figueroa, Carlos Buil	Second version of scenario for the invoice use case
		Mari Carmen Suárez-Figueroa, Caterina Caracciolo	Review of scenario for the fishery use case
		Mari Carmen Suárez-Figueroa, German Herrero	Review of scenario for the nomenclature use case
0.22	22-08-2007	Mari Carmen Suárez-Figueroa, Valentina Presutti	Extreme Programming section in the State of the Art
0.23	24-08-2007	Mari Carmen Suárez-Figueroa	Review of the State of the Art, general scenarios for building ontology networks, and invoice and nomenclature ontology network life cycle
0.24	27-08-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of the NeOn Ontology Development Process
0.25	28-08-2007	Mari Carmen Suárez-Figueroa, Martin Dzbor	Software Engineering: Development Process section in the State of the Art
0.26	28-08-2007	Mari Carmen Suárez-Figueroa	On-To-Knowledge Development Process in the State of the Art
0.27	30-08-2007	Mari Carmen Suárez-Figueroa	Diligent Development Process and Life Cycle in the State of the Art On-To-Knowledge Development Process (revision) and Life Cycle in the State of the Art

0.28	3-09-2007	Mari Carmen Suárez-Figueroa	Revision of NeOn Life Cycle
0.29	4-09-2007	Mari Carmen Suárez-Figueroa, Valentina Presutti	Second version of Extreme Programming section in the State of the Art
0.30	5-09-2007	Mari Carmen Suárez-Figueroa	Revision of NeOn Life Cycle
0.31	5-09-2007	Mari Carmen Suárez-Figueroa	Revision of Required or If Applicable Activities
0.32	6-09-2007	Mari Carmen Suárez-Figueroa	Revision of NeOn Life Cycle
0.33	7-09-2007	Mari Carmen Suárez-Figueroa, Elena Montiel-Ponsoda	Revision of NeOn Life Cycle (Natural Language Questions)
0.34	10-09-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of NeOn use cases life cycles
		Mari Carmen Suárez-Figueroa, Martin Dzbor (Q.A. reviewer)	Inclusion of general Q.A. reviewer's comments
0.35	11-09-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of the scenarios for building ontology networks
		Mari Carmen Suárez-Figueroa	Revision of NeOn Life Cycle
0.36	12-09-2007	Mari Carmen Suárez-Figueroa	Revision of NeOn Life Cycle
0.37	13-09-2007	Mari Carmen Suárez-Figueroa	Revision of Introduction Revision of NeOn Life Cycle
0.38	14-09-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of Introduction
0.39	16-09-2007	Mari Carmen Suárez-Figueroa, Elena Montiel-Ponsoda	Revision of NeOn Life Cycle (Natural Language Questions)
0.40	19-09-2007	Mari Carmen Suárez-Figueroa, German Herrero	Review of scenario for the nomenclature use case
		Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of the conclusions in the State of the Art
		Mari Carmen Suárez-Figueroa, Carlos Buil	Revision of scenario for the invoice use case
0.41	20-09-2007	Mari Carmen Suárez-Figueroa, Martin Dzbor (Q.A. reviewer)	Revision and restructuring of the scenarios for building ontology networks (based on Q.A. reviewer's comments)
0.42	21-09-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of Introduction and first version of Conclusions
0.43	24-09-2007	Mari Carmen Suárez-Figueroa, German Herrero	Review of scenario for the nomenclature use case
		Mari Carmen Suárez-Figueroa, Carlos Buil	Revision of scenario for the invoice use case
0.44	25-09-2007	Mari Carmen Suárez-Figueroa, Martin Dzbor (Q.A. reviewer)	Inclusion of Q.A. reviewer's comments in the ontology development process section
0.45	25-09-2007	Mari Carmen Suárez-Figueroa	Revision of the scenarios for building ontology networks Revision of NeOn Life Cycle Revision of scenarios for the NeOn use cases
0.46	26-09-2007	Mari Carmen Suárez-Figueroa, Caterina Caracciolo	Review of scenario for the fishery use case

0.47	26-09-2007	Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez	Revision of Conclusions and Future Work
0.48	26-09-2007	Mari Carmen Suárez-Figueroa	Revision of the whole deliverable
		Mari Carmen Suárez-Figueroa, Martin Dzbor (Q.A. reviewer)	Inclusion of general Q.A. reviewer's comments
0.49	28-09-2007	Mari Carmen Suárez-Figueroa	Revision of the whole deliverable and references
0.50	1-10-2007	Mari Carmen Suárez-Figueroa, Rosario Plaza	Revision of Chapter 1 (from a linguistic point of view)
0.60	3-10-2007	Mari Carmen Suárez-Figueroa, Rosario Plaza	Revision of Chapters 2 and 3 (from a linguistic point of view)
0.70	5-10-2007	Mari Carmen Suárez-Figueroa, Rosario Plaza	Revision of Chapters 4 and 5 (from a linguistic point of view)
0.80	8-10-2007	Mari Carmen Suárez-Figueroa, Rosario Plaza	Revision of Chapters 6 and 7 (from a linguistic point of view)
0.90	9-10-2007	Mari Carmen Suárez-Figueroa	Revision of the whole deliverable and references
0.95	13-10-2007	Mari Carmen Suárez-Figueroa, Martin Dzbor (Q.A. reviewer)	Inclusion of Q.A. reviewer's comments on Chapter 5
1.0	14-10-2007	Mari Carmen Suárez-Figueroa	Final version

Executive Summary

This deliverable deals with ontology network development process, ontology network life cycle models and ontology network life cycle, that are necessary to provide methodological guidelines for the development of ontology networks. The deliverable also provides the following new definitions:

❑ The ***ontology network development process*** is defined in NeOn as the process by which user needs are translated into an ontology network. The main goal of the ontology network development process is to identify and define which activities are carried out when an ontology network is developed.

❑ An ***ontology network life cycle model*** is defined as the framework, selected by each organization, on which to map the activities identified in the ontology network development process to produce the ontology network life cycle.

❑ The **ontology network life cycle** is defined as the project-specific sequence of activities created by mapping the activities identified in the ontology network development process onto a selected ontology network life cycle model.

The deliverable contains five main chapters, which deal with critical methodological issues (development process, models and life cycle).

The first chapter presents the state of the art of the aforementioned issues (development process, models and life cycle) in three different fields: Software Engineering, Knowledge Engineering and Ontology Engineering.

The second chapter describes the ontology network development process in NeOn by:

▪ defining the NeOn Glossary of Activities; and

▪ proposing required and dispensable activities.

NeOn

The third chapter of the deliverable identifies and describes 8 scenarios for collaboratively building network of ontologies with special emphasis in reusing and reengineering knowledge resources (ontological and non ontological) and in merging ontologies.

The fourth chapter defines and describes several ontology network life cycle models, according to those models defined in the Software Engineering field; it also provides guidelines for obtaining the concrete life cycle for an ontology network.

Finally, the fifth chapter describes the ontology network life cycle for the three NeOn use cases (fisheries, invoicing and nomenclator).

Other essential methodological points such as methods, techniques and tools will be described in D5.4.1 (*NeOn methodology for building contextualized networked ontologies*) at M24.

Table of Contents

List of Tables

List of Figures

NeOn

1. Introduction

The Semantic Web of the future will be characterized by using a very large number of ontologies embedded in ontology networks built collaboratively by distributed teams, where an ontology network or a network of ontologies is defined as a collection of ontologies (called networked ontologies) related together through a variety of different relationships such as mapping, modularization, version, and dependency relationships [35]. So, future Semantic Web applications will be based on networks of contextualized ontologies, which will be in continuous evolution. Such networks could include ontologies that already exist or they could be developed by reusing either other ontologies or non ontological but knowledge aware resources (thesauri, lexicons, text corpora, DBs, UML diagrams, etc.) built by others [9].

1.1. WP5 Objectives and Main Tasks

In this context, the main objectives of WP5 are:

1   To create the NeOn methodology that will support the collaborative aspects of ontology development and the reuse and the dynamic evolution of networked ontologies in distributed environments where contextual information is introduced by developers (domain experts, ontology practitioners) at different stages of the ontology development process.

2   To create a rigorous, sound NeOn methodology for the development of large scale Semantic Web applications that supports the reference architecture and the service oriented infrastructure developed in WP6.

3   To provide qualitative and quantitative experimental evidence of how by following the NeOn methodologies the system development improves.

These objectives will be achieved through investigating the following tasks:

- Task 5.3. Identification and definition of the *development process* and *life cycle for networks of ontologies.* Results of these researches are included in this deliverable.

- Task 5.4. The NeOn methodology for building collaboratively ontology networks will include methods, techniques and tools for carrying out the activities identified and defined in the ontology network development process.

- Task 5.5. The NeOn methodology for development of large-scale Semantic Web applications from the initial phases (requirement analysis) of the development process until the stage prior to the implementation. To ease the use of the NeOn reference architecture and of the NeOn software components, a set of developer-oriented reference specifications will be defined. These specifications will serve as a skeleton for adapting the selected components and for developing new complex semantic-based software components and semantic applications.

- Task 5.6. Experimentation with the NeOn methodologies. In this task experiments, methods, and metrics are proposed for evaluating the main outcomes produced in this WP. The goal is to provide qualitative and quantitative evidence that with the NeOn methodologies ontologies and systems are built faster and better.

1.2. Our Approach

With this new vision of the ontologies and the Semantic Web, it is important to provide strong methodological support for the collaborative and context-sensitive development of ontology networks.

Yet, we do not start from scratch. Our work is based on Software Engineering methodological work (terminology, models, ideas, etc.), on existing methodologies for ontology construction, particularly METHONTOLOGY [32] (from the UPM team), On-To-Knowledge [53] and DILIGENT [47] (from UKARL team), and on the experience of people involved in building the ontologies for the fishery and pharmaceutical domains in the NeOn use cases. It is also based on the experience gained by some of the consortium members from participating in other international and national projects. Whenever it is possible, we present the WP5 results through a Software Engineering approach with different levels of complexity to facilitate a promptly assimilation by software developers and ontology practitioners.

Methodological frameworks are widely accepted in different mature fields, like in Software Engineering. Taking into account that single ontologies and ontology networks could be seen as "software artifacts" and as part of software products, current software methodologies could be relevant to the Ontology Engineering field; at least, to take them as inspiration.

In order to provide the methodological support for collaborative and context-sensitive development of ontology networks, we have taken IEEE standard definitions related to development process, life cycle models, and life cycle in Software Engineering as a starting point and we have adapted these definitions to the specific features of the development of single ontologies and ontology networks. Thus, the ontology network development process can be seen as a specific case of software development process, and an ontology network life cycle, as a specific case of software life cycle.

1.3. Main Contributions

The new definitions provided in this deliverable are the following:

❑ The ***ontology network development process*** is defined in NeOn as the process by which user needs are translated into an ontology network. The main goal of the ontology network development process is to identify and define which activities are carried out when an ontology network is developed.

❑ An ***ontology network life cycle model*** is defined as the framework, selected by each organization, on which to map the activities identified in the ontology network development process to produce the ontology network life cycle.

❑ The **ontology network life cycle** is defined as the project-specific sequence of activities created by mapping the activities identified in the ontology network development process onto a selected ontology network life cycle model.

Therefore, in the NeOn project we argue that it is not enough to support the ontology network development process; the ontology network life cycle needs also be supported. Consequently, in addition to the previous definitions, we have included other important results in this deliverable such as the following ones:

❑ The *NeOn Glossary of Activities*, which identifies and defines the activities carried out when ontology networks are collaboratively built. In order to facilitate the identification of which activities are important in the ontology network life cycle, in a first draft we identified which

activities from those included in the NeOn Glossary of Activities are mandatory and which are optional.

❑ A proposal that includes different *ontology network life cycle models*.

❑ Guidelines for obtaining the ontology network life cycle for a concrete ontology network; the ontology network life cycle is the specific sequence of activities needed for developing a concrete ontology network following a concrete ontology network life cycle model.

❑ Preliminary evaluation results of the notions presented in this document within the three NeOn use cases.

1.4. Deliverable Structure

The deliverable is structured as follows:

❑ Chapter 2 deals with the state of the art of the development process and life cycle models in different fields (Software Engineering, Knowledge Engineering and Ontology Engineering).

❑ Chapter 3 presents the definition of the ontology network development process in NeOn. This chapter also includes the following results:

  ➢ The NeOn Glossary of Activities, which identifies and defines the activities involved in ontology network construction. Definitions in the glossary have been collaboratively built and consensuated by all NeOn partners; the process followed to reach consensus for building the NeOn Glossary of Activities is also included.

  ➢ A classification of the activities into those that are required for the development of ontology networks and those that are applicable (depending on the cases), but not required, and, therefore, they are non-essential or dispensable.

❑ Chapter 4 deals with the identification and definition of 8 scenarios for building ontology networks collaboratively with special emphasis in reuse, reengineering and merging.

❑ Chapter 5 provides the definition of several ontology network life cycle models according to those defined in the Software Engineering field; it also provides guidelines for obtaining the concrete life cycle for an ontology network.

❑ Chapter 6 describes the ontology network life cycle for the three NeOn use cases (fisheries, invoicing and nomenclator).

❑ Chapter 7 presents the conclusions and future work.

Defining the NeOn methodology (which includes methods, techniques and tools for the activities identified and defined in the NeOn Glossary of Activities) is out of the scope of this document, it will be presented in deliverable D5.4.1 (*NeOn methodology for building contextualized networked ontologies*) at M24.

Thorough evaluations of the different ontology network life cycle models proposed in this deliverable will be carried out within task T5.6 (*Evaluation of the NeOn methodologies*) and included in deliverable D5.6.1 (*Experimentation with NeOn methodologies*).

1.5. Relation with the Rest of WPs within Project

The relation between this deliverable and the other parts of WPs in the NeOn project is briefly described below:

❑ The NeOn Glossary of Activities includes activities related to the research being carried out in WP1 (e.g., ontology evolution), WP2 (e.g., ontology localization), WP3 (e.g., ontology aligning), and WP4 (e.g., ontology customization).

❑ FAO, iSOCO, and ATOS as leaders of the use case WPs (WP7 and WP8, respectively) reviewed the NeOn Glossary of Activities from their practical point of view.

❑ The work carried out in WP7 and WP8 and their initial ideas for developing ontologies provide a useful feedback for methodological guidelines.

❑ Methodological guidelines for building ontology networks were provided to the two use cases in WP8.

2. State of the Art

Research on methodologies for the Ontology Engineering field is in its "adolescence" since this field is only 10 years old. The 1990s and the first years of this new millennium have witnessed the growing interest of many practitioners in approaches that support the creation and management as well as the population of single ontologies built from scratch. There are some methodological approaches (e.g., METHONTOLOGY [32], On-To-Knowledge [53], and DILIGENT [47]) that help develop ontologies. All these approaches have provoked a step forward by having transformed the art of constructing single ontologies into an engineering activity. However, such methodological approaches lack guidelines for building ontologies embedded in ontology networks in continuous evolution and they do not follow a user-oriented approach.

One important difference between a technical field that is in its "adolescence" and another that has reached "adulthood" is that the mature field has widely accepted methodologies, whereas the young discipline usually has none or a few. It we look at the Software Engineering field, for example, we can say it has reached "adulthood" because it has widely accepted methodologies for developing software; in fact, although different development methodologies are used, the methodologies share the same principles and viewpoints and provide similar activities and techniques [23].

The IEEE standard glossary of Software Engineering terminology [1] defines software as "computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system", Ontologies are part (sometimes only potentially) of software products. Therefore, ontologies should be developed according to the standards generally proposed for software, which should be adapted to the special characteristics of ontologies [23].

For such reasons, we will use and adapt Software Engineering terminology and methodological principles (taken from different IEEE standards) as basis for the current methodological research in the Ontology Engineering field.

Methodologies involve the following issues: development process, life cycle models and life cycle, methods, techniques and tools to be used during the ontology building. As mentioned in Chapter 1, this deliverable deals only with the three first issues: development process, life cycle models and life cycle. So in this chapter, some notions related to development process, life cycle model and life cycle taken from the Software Engineering field are included as background knowledge to better understand the complete deliverable. Furthermore, this chapter presents different works on development process and life cycle models carried out in the following fields: Software Engineering (Sections 2.2 and 2.3), Knowledge Engineering (Sections 2.4 and 2.5), and Ontology Engineering (Sections 2.6 and 2.7). Finally, conclusions about development process and life cycle models are included.

2.1. Basic Terminology from Software Engineering

In this section we include several terms defined in Software Engineering, which are crucial for this deliverable. The terminology included have been extracted from the following IEEE documents:

➢ IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990) [1].

➢ IEEE Guide for Developing Software Life Cycle Processes (IEEE Std 1074.1-1995) [2].

➢ IEEE Standard for Developing Software Life Cycle Processes (IEEE Std 1074-1997) [3].

The terminology is presented in an incremental way, from the most simple term to the most complex one.

A **task** can be defined as the smallest unit of work subject to management accountability. Another definition of **task** [2] can be a well-defined work assignment for one or more project members. Related tasks are usually grouped to form activities.

An **activity** [3] is a defined body of work that is to be performed, including its required input and output information.

A **process** [1] is a sequence of steps performed for a given purpose. A process is composed of activities.

The **software development process** [1] is the process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use.

A **software life cycle model** [3] is the framework, selected by each organization, on which to map the activities identified in the software development process to produce the software life cycle. For example, waterfall model [48], evolving prototyping model [19], etc.

The **software life cycle** [1] is defined as the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. Furthermore, the software life cycle [3] is the project-specific sequence of activities that is created by mapping the activities identified in the software development process onto a selected software life cycle model.

2.2. Software Engineering: Development Process

A **software development process** can be seen as a process of solving problems. That is, such process transforms an user necessity into a computable product, that satisfies the user need.

This section briefly summarizes three well-known Software Engineering development process: IEEE development process (Section 2.2.1), ISO development process (Section 2.2.2) and RUP development process (Section 2.2.3).

2.2.1. IEEE development process

According to IEEE [4], the 69 activities identified in the software project development process are grouped administratively into five main activity groups. The full set of activities covers the entire software life cycle from concept exploration through the eventual retirement of the software system. The main activity groups [4] are the following:

1. Project Management Activity Group [4]. This group includes activities that initiate, monitor, and control a software project throughout its life cycle. For example, Project Monitoring and Control activities are used to track and manage the project. During these activities, actual project performance is tracked, reported, and managed against the planned performance. Special consideration is given to the management of risk. This group is divided into three subgroups: Project Initiation, Project Planning, and Project Monitoring and Control.

2. Pre-Development Activity Group [4]. This group includes the activities that explore and allocate system requirements before the software development can begin. For example, the

feasibility study is used to analyse the idea or need, the potential approaches, and all life cycle constraints and benefits. Justification for each recommendation shall be fully documented and formally approved by all concerned organizations (including the user and the developer). This group is divided into three activity subgroups: Concept Exploration, System Allocation, and Software Importation.

3. Development Activity Group [4]. This group includes the activities performed during the development and enhancement of a software project. For example, Design activity has as main goal to develop a coherent, well-organized representation of the software system that meets the software requirements. This group is divided into three subgroups: Software Requirements, Design, and Implementation.

4. Post-Development Activity Group [4]. This group includes the activities that install, operate, support, maintain, and retire a software product. For example, the Installation activity consists of the transportation and installation of a software system from the development environment to the target environment(s); and the Operation and Support activities involve user operation of the system and ongoing support. Support includes providing technical assistance, consulting with the user, and recording user support requests by maintaining a Support Request Log. This group is divided into four subgroups: Installation, Operation and Support, Maintenance, and Retirement.

5. Support Activity Group [4]. This group includes activities that are necessary to assure the successful completion of a project, but that are considered as supporting activities rather than as activities directly oriented to the development effort. In other words, the Support Activity Group includes the set of activities necessary to ensure that a system fulfils its original requirements and any subsequent modifications to those requirements. For example, the Documentation Development activity for software development and usage has as purpose to plan, design, implement, edit, produce, distribute, and maintain the documents that are needed by developers and users. This group is divided into four subgroups: Evaluation, Software Configuration Management, Documentation Development, and Training.

2.2.2. ISO development process

ISO 12207 [7] establishes a software development process including processes, activities and tasks applied during the acquisition and configuration of the services of the system. Each process within the life cycle has a set of outcomes associated with it. The structure of the standard reflects a flexible, modular way that is adaptable to the needs of whoever uses it. The standard emphasises two basic principles: *modularity* and *responsibility*. Modularity means processes with minimum coupling and maximum cohesion. Responsibility means establishing a responsibility for each process, facilitating the application of the standard to projects in which many people can be legally involved.

This International Standard [7] groups the activities that may be performed during the life cycle of a software system into seven process groups. Each of the processes within those groups is described in terms of its purpose and desired outcomes and lists activities and tasks that need to be performed to achieve those outcomes. The purposes and outcomes of these processes constitute a process reference model.

The seven process groups, shown in Figure 1, are the following:

1. **Agreement Processes** define the activities necessary to establish an agreement between two organizations. These processes are the acquisition process and the supply process.

2. **Project-Enabling Processes** manage the organization's capability to acquire and supply products or services through the initiation, support and control of projects.

3. **Project Processes**. There are two categories of Project Processes: project management processes and project support processes. The Project Management Processes are used to

plan, execute assess and control the progress of a project. The Project Support Processes support specialized management objectives.

4. **Technical Processes**, which include eleven processes, are used to define the requirements for a system, to transform the requirements into an effective product, to permit consistent reproduction of the product where necessary, to use the product, to provide the required services, to sustain the provision of those services and to dispose of the product when it is retired from service.

5. **Software Implementation Processes**, which include seven processes, are used to produce a specified system element (software item) implemented in software. Those processes transform specified behaviour, interfaces and implementation constraints into implementation actions resulting in a system element that satisfies the requirements derived from the system requirements.

6. **Software Support Processes**, which includes eight processes, provide a specific focused set of activities for performing a specialized software process. A supporting process assists the Software Implementation Process as an integral part with a distinct purpose, contributing to the success and quality of the software project.

7. **Software Reuse Processes**, which includes three processes (domain engineering process, reuse program management process, and reuse asset management process), support an organization's ability to reuse software items across project boundaries.

System Life Cycle Processes

Agreement Processes

Acquisition Process
(Clause 6.1.1)

Supply Process
(Clause 6.1.2)

Project-Enabling Processes

Life Cycle Model
Management Process
(Clause 6.2.1)

Infrastructure
Management Process
(Clause 6.2.2)

Project Portfolio
Management Process
(Clause 6.2.3)

Human Resource
Management Process
(Clause 6.2.4)

Quality Management
Process
(Clause 6.2.5)

Project Processes

Project Planning Process
(Clause 6.3.1)

Project Assessment and
Control Process
(Clause 6.3.2)

Decision Management
Process
(Clause 6.3.3)

Risk Management
Process (Clause 6.3.4)

Configuration
Management Process
(Clause 6.3.5)

Information Management
Process
(Clause 6.3.6)

Measurement Process
(Clause 6.3.7)

Technical Processes

Stakeholder
Requirements Definition
Process (Clause 6.4.1)

System Requirements
Analysis Process
(Clause 6.4.2)

System Architectural
Design Process
(Clause 6.4.3)

Implementation Process
(Clause 6.4.4)

System Integration
Process
(Clause 6.4.5)

System Qualification
Testing Process
(Clause 6.4.6)

Software Installation
Process
(Clause 6.4.7)

Software Acceptance
Support Process
(Clause 6.4.8)

Software Operation
Process
(Clause 6.4.9)

Software Maintenance
Process
(Clause 6.4.10)

Software Disposal
Process
(Clause 6.4.11)

Software Life Cycle Processes

SW Implementation Processes

Software Implementation
Process
(Clause 7.1.1)

Software Requirements
Analysis Process
(Clause 7.1.2)

Software Architectural
Design Process
(Clause 7.1.3)

Software Detailed Design
Process
(Clause 7.1.4)

Software Construction
Process
(Clause 7.1.5)

Software Integration
Process
(Clause 7.1.6)

Software Qualification
Testing Process
(Clause 7.1.7)

SW Support Processes

Software Documentation
Management Process
(Clause 7.2.1)

Software Configuration
Management Process
(Clause 7.2.2)

Software Quality
Assurance Process
(Clause 7.2.3)

Software Verification
Process
(Clause 7.2.4)

Software Validation
Process
(Clause 7.2.5)

Software Review Process
(Clause 7.2.6)

Software Audit Process
(Clause 7.2.7)

Software Problem
Resolution Process
(Clause 7.2.8)

Software Reuse Processes

Domain Engineering
Process
(Clause 7.3.1)

Reuse Program
Management Process
(Clause 7.3.3)

Reuse Asset
Management Process
(Clause 7.3.2)

Figure 1. Process Groups in the Software Development Process [7]

2.2.3. RUP development process

One of the best-known methodologies is the Rational Unified Process (or RUP). This methodology is an iterative software development framework originally created by the Rational Software Corporation in late 1980-s, which became a division of IBM in 2002. Since then the methodology may be sometimes referred to as IBM RUP process.

RUP is not a single concrete process that would prescribe definite steps for the software development, but rather it is an adaptable framework. It is intended to be tailored and customized by the development organizations and project teams that are expected to select the elements of the process appropriate for their needs. RUP is also a fully-fledged software product in its own right (it includes a hyperlinked knowledge base with sample artefacts and detailed descriptions for many different types of activities).

RUP is based on six key principles for so-called business-driven development:

1. *Adapting the process*. There is not a single 'typical' software development process; organizations must have options to tailor the process to their needs. To this extent, RUP provides pre-configured process templates for small, medium and large projects, which can be used for easier adoption.

2. *Balancing stakeholder priorities*. The key principle of this tenet is to address the traditional requirements gathering and analysis, and see these activities as a part of a broader relationship to business goals and stakeholder interests, which are often contradictory.

3. *Collaborating across teams*. Since RUP is built upon an assumption of iterative, incremental development, the direct effect of this assumption is that a range of developers will be involved.

This tenet addresses an often-seen feature of larger, longer software projects, which is by some referred to as 'over the wall' design – limited interaction of parties involved in different stages of the development process, which decreases efficiency and often quality.

4. *Demonstrating the added value iteratively*. As above, software projects are assumed to be delivered in increments and in an iterative fashion. The increment, which includes the value of the past iteration, is also used as a measure of the project progress. That increment is also used to encourage feedback from stakeholders about the direction of the project. All this allows the provision of a timely feedback and rectification of identified shortcomings.

5. *Elevating the level of abstraction*. This is the key innovation championed by RUP. RUP motivates the use of reusable assets such as software patterns. Its importance is in preventing the engineers from 'doing hacks'; i.e. going directly from the requirements to custom-made software code. A higher level of abstraction also allows discussions on different architectural levels, which are in practice often accompanied by visual representations, for example using UML.

6. *Focusing continuously on quality*. Quality checks are not only at the end of each iteration but form a continuous activity, often performed daily and supported by the entire team. Automating test scenarios (scripts) are increasingly popular to cope with the increasing amount of tests due to iterative development.

The RUP development process recognizes the following four broad phases:

❑ *Inception phase*. In this initial phase, the business case (including the context of the product and its success factors) is established. At least two aspects are crucial for this phase – an initial use case and an initial project description comprising the key requirements, constraints and features. The purpose of this phase is to appreciate the scope of the definition, the understanding of the requirements and their satisfaction in the proposed use case(s), and the overall credibility of the project.

This is also the phase where one usually defines base milestones related to satisfying the key aspects discussed in this phase.

❑ *Elaboration phase*. This is where the project starts taking its shape; the key activity in this step is the problem domain analysis and the key outcome is usually the initial architecture. The purpose of this phase is to extend and elaborate the initial use case(s) and use case descriptions that start driving the formulation of key architectural modules.

❑ *Construction phase*. In this phase, the main focus goes away from the analysis and to the synthesis and development of components and other features of the system being designed. This is the phase when the bulk of the coding, modelling and also testing take place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments. Thus, two key outcomes come from this phase – a realistic project plan (e.g. the familiar Gantt chart is a good example of this) and a suite of demonstrable prototypes. It is also in this phase when the first software releases are produced; hence, this phase is often associated with the operational capability milestones.

❑ *Transition phase*. Here the project is moving from the development organization to the end user(s). The activities of this phase include training of the end users and maintainers and beta testing of the system to validate it against the end users' expectations. The product is also checked against the quality level set in the inception phase. If it does not meet this level or the standards of the end users, the entire cycle may begin again or may be returned to the appropriate earlier phase.

2.3. Software Engineering: Life Cycle Models

The life of a software product can be modelled by a life cycle model consisting of stages [7].

Software life cycle models define how to develop and maintain a software system, i.e., they describe different ways of organizing the activities to be carried out [46]. Software life cycle models are abstractions of the phases or states a software product passes through along its life. These models could also determine the order of the phases and establish the transition criteria between phases.

It is known that there is no a life cycle model valid for all the software projects. Each software life cycle model is appropriate for a concrete project, depending on several characteristics:

- Software organization culture: the organization is enterprising or conservative (that is, it likes to assume or not risks in projects).

- Previous experience of the software development team.

- Software application area.

- Comprehension and volatility of the software requirements.

Once a particular software life cycle model has been chosen for the software project, then the concrete software life cycle is obtained. The **software life cycle** describes the software product life from its conception until its implementation, deliver, use, and maintenance. That is, the software life cycle describes the states a software product passes through from its conception until its dead.

This section briefly summarizes the most commonly used and well-known software life cycle models. Such models are based on the existing literature, but it was difficult to summarize them because different authors explain some models in a different way. For these reason, we include in this section our own understanding of the existing literature.

- ➢ The sequential models, such as the waterfall (Section 2.3.1) and the 'V' model (Section 2.3.2).

- ➢ The gradual models, such as the incremental development (Section 2.3.3) and the iterative development (Section 2.3.4).

- ➢ The prototype-based models, such as evolutionary prototyping model (Section 2.3.5) and rapid throwaway prototyping approach (Section 2.3.6).

- ➢ The spiral model (Section 2.3.7).

- ➢ The extreme programming model (Section 2.3.8).

- ➢ The reuse components model (Section 2.3.9).

2.3.1. Waterfall life cycle model

The pure waterfall life cycle model shown in Figure 3 was defined by Royce [48] to help cope with the growing complexity of the development of software projects. The use of such a model encourages the developer to specify what the system is supposed to do (i.e., to define the requirements) before building the system and to plan how components are going to interact (i.e., designing) before building the components (i.e., implementing). This model enables project

managers to track progress more accurately and demands that the development process generate a series of documents that can later be used to test and maintain the system. The use of this model reduces development and maintenance costs due to all of the above reasons.
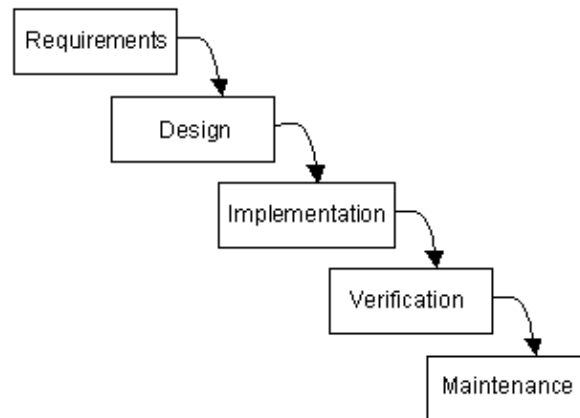
Figure 2. Pure Waterfall Life Cycle Model

The pure waterfall life cycle model [39, 46] is divided into sequential phases (analysis, design, implementation and testing) and represents the stages as a waterfall, from a particular stage through the following one. This model represents the stages of a project, starting with the requirements plan, such as the problem which the user has or thinks he has, through the analysis and definition of requirements, design, coding, testing, and operation to maintenance. The result of each phase is one or more documents that are approved ('signed-off') [52]. The following phase should not start until the previous one has finished. For example, when all the requirements have been identified, analyzed to check their integrity and consistency, and documented, then the development team can begin with the activities related to the system design.

In practice, the waterfall life cycle model has a lineal order in the transitions between phases, with a review for each phase. It is not possible to pass to the following phase until the current one has been not completed. This model permits to backtrack to previous phases if errors or mistakes are detected [52]. Figure 3 shows this usual waterfall life cycle model.
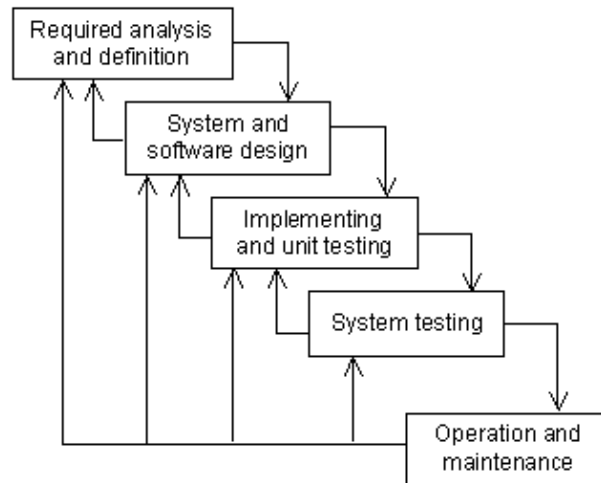
Figure 3. Waterfall Life Cycle Model [52]

The waterfall life cycle model assumes that the requirements are completely known and without ambiguities at the beginning of the development process. Such requirements do not change along the project. Therefore, this model should only be used when the requirements are well understood and unlikely to change radically during system development. This model is still used particularly when the software project is part of a larger systems engineering project [52].

The advantages of the waterfall model are that documentation is produced at each phase, that it is easy to understand, and that it is easy to plan a project according to this sequential approach.

The main problem with this type of model is that it does not reflect clearly how the software code is developed [46] because real projects rarely follow the sequential flow that the model proposes. Other problem is its inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements [52]. Another inconvenience of this model is that the requirements can be obsolete at the end of the project. And there is no software product to show to the user until the end of the project.

2.3.2. 'V' life cycle model

The most commonly known variant of the waterfall model, the 'V' life cycle model [46, 8], is a sequential path of execution of processes, where each phase must be completed before the next phase begins. This model explicitly includes the observation that the result of each development task must be verified in a corresponding test task. This life cycle model gives emphasis to the validation of the products in each phase.

Figure 4 shows the graphical representation for the 'V' life cycle model. The symmetry between the left and right sides of the model reflects the relationship between the steps on the left and the steps on the right. The system definition that is generated on the left is ultimately used to verify the system on the right. The connections between the left and right are indicated by the arrows that cross the "V", showing how plans developed on the left drive the process on the right. These connections provide continuity between the beginning and end of project development and ensure that the engineers are focused on the completion of the project from the beginning [10].

The relation between the right and left sides of the V-model, shown in Figure 4, implies that if problems are found during the verification and validation, then the left part of the 'V' can be

performed again in order to solve the problem and improve the requirements, the design and the code before redoing the tests.
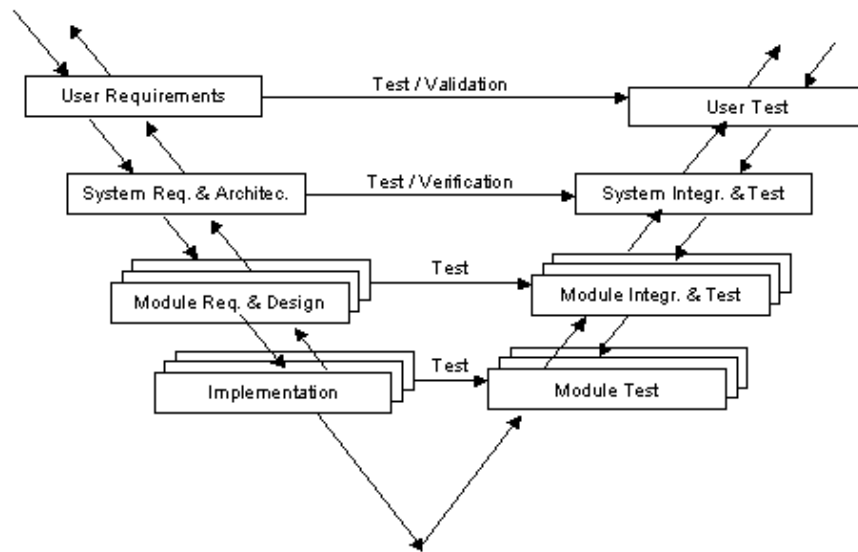
Figure 4. 'V' Life Cycle Model

2.3.3. Incremental development life cycle model

Incremental development model [52] involves producing and delivering the software in increments rather than in a single package.

In the incremental development model [46], the system, specified in the requirement documents, is divided into functional subsystems, which are completely developed in different cycles. Each cycle follows the waterfall model. The system development starts with a small functional subsystem which is fully developed. In each cycle a functional subsystem is developed and a new version is delivered to the user at the end of each cycle. Following this model, the software development process begins with the most essential functionalities and continues and ends with the less important ones.

In this model, the complete system is built gradually through different iterations. This model can be seen as a repetition of the waterfall model for each functional subsystem. So, in each iteration each subsystem passes through the requirements, design, implementation and testing phases.

The two main advantages of this approach are: accelerated delivery of customer services and user engagement with the system [52].

The main advantage of the incremental development model [19] is that it produces an operational system more rapidly, thus reducing the possibility that the user needs change during the development process. Incremental development implies that we understand up front most of our requirements and simply choose to implement them in subsets of increasing capability. With the incremental development life cycle model, risk of developing the wrong thing is reduced by breaking the project into a series of small subprojects (increments).

In the incremental development model, as in the waterfall one, the problem and the overall requirements of the final product are well-known at the start of the development. In incremental models however a limited set of requirements is allocated to each increment and with each

successive release more requirements are addressed until the final release satisfies all requirements.

One risk with the incremental approach is that the first releases address such a limited set of requirements that the customer could be dissatisfied; one opportunity on the other hand is that wrong or missing requirements can be corrected in time.

There are some types of systems where incremental development and delivery is not the best approach. These are very large systems where development may involve teams working in different locations, some embedded systems where the software depends on hardware development and some critical systems where all the requirements must be analysed to check for interactions that may compromise the safety or security of the system [52].

2.3.4. Iterative development life cycle model

The iterative development [46] also divides the system into small parts, as the incremental development. However, instead of building and delivering a new and complete subsystem in each cycle, the iterative model proposes to build and deliver in each cycle a system with of all the subsystems partially developed, being the functionalities of each subsystem improved in each new cycle. At the end of the cycle, an improved new version of the complete system is delivered to the user.

This life cycle model has as purpose to reduce the risk between the user needs and the final product; it consists in the iteration of several waterfall life cycles. At the end of each iteration, an improved or refined version of the software product is delivered to the user.

2.3.5. Evolutionary prototyping life cycle model

The evolutionary prototyping model [19] is based on the assumption that it is often difficult to know all the system requirements at the beginning of a project and that the requirements can change during the project life. Following an evolutionary prototyping life cycle model [19], the developers construct a partial implementation of the system which meets the known requirements. The prototype is rigorously developed by means of incorporating the best understood requirements. The prototype is then evaluated and used by its intended users and the requirements are refined based on such evaluation.

Evolutionary prototyping implies that software developers do not know all of the requirements up front, and so they need to experiment with an operational system in order to learn them; however, in the case of evolutionary prototypes software developers are more likely to start with those system aspects that are best understood and thus they build upon their strengths.

This kind of model is used when the user does not know exactly what he wants, or when what the user wants is not understood by the development team, or when the viability of the solution is not clear. Furthermore, in this model a high degree of uncertainty is assumed.

2.3.6. Rapid throwaway prototyping life cycle model

The rapid throwaway prototyping approach [19] addresses the issue of ensuring that the software product being proposed really meets the users' needs. The approach is to construct a "quick and dirty" partial implementation of the system prior to (or during) the requirements stage. The potential users utilize and evaluate this prototype for a period of time and supply the developers with feedback concerning its strengths and weaknesses. This feedback is then used to modify the software requirements specification to reflect the real user needs. At this point, the developers can

proceed with the actual system design and implementation with the confidence that they are building the "right" system (except in those cases where the user needs evolve). Note that in the case of throwaway prototypes, software developers are likely to implement only those aspects of the system that are poorly understood or doubtful.

The objective of throwaway prototyping is to validate or derive the software requirements. The development should start with requirements that are not well understood in order to find out more about them. Requirements that are straightforward may never need to be prototyped [52].

Throwaway prototypes have a very short lifetime. It must be possible to change them rapidly during development, but long term maintainability is not required. Poor performance and reliability may be acceptable in a throwaway prototype so long as it helps everyone understand the requirements [52].

This kind of model is used when the user does not know exactly what he wants, or when what the user wants is not understood by the development team, or when the viability of the solution is not clear.

2.3.7. Spiral life cycle model

The spiral model [46] was designed to include the best features from the waterfall and prototyping models, and it introduces a new component: the concept of risk, which appears because of the uncertainties in the requirements, and its assessment. This life cycle model [39] is based on the idea of showing the historical cycles of the Earth's geology.

The spiral life cycle model [46, 16, 15], shown in Figure 5, proposes to combine the development activities with the risk management activities to minimize and control the risk. This model divides the software engineering space into four quadrants: management planning, formal risk analysis, engineering, and customer assessment. The development of the system is divided into n-cycles. Each cycle consists of waterfall phases. The result of each cycle is a prototype, and the development cycle continues until the final version of the product is reached. With each iteration around the spiral (beginning at the centre and working outward), more complete versions of the system are progressively built. Furthermore, user feedback is considered at the output of each cycle, and new additional features are included into the next cycle.
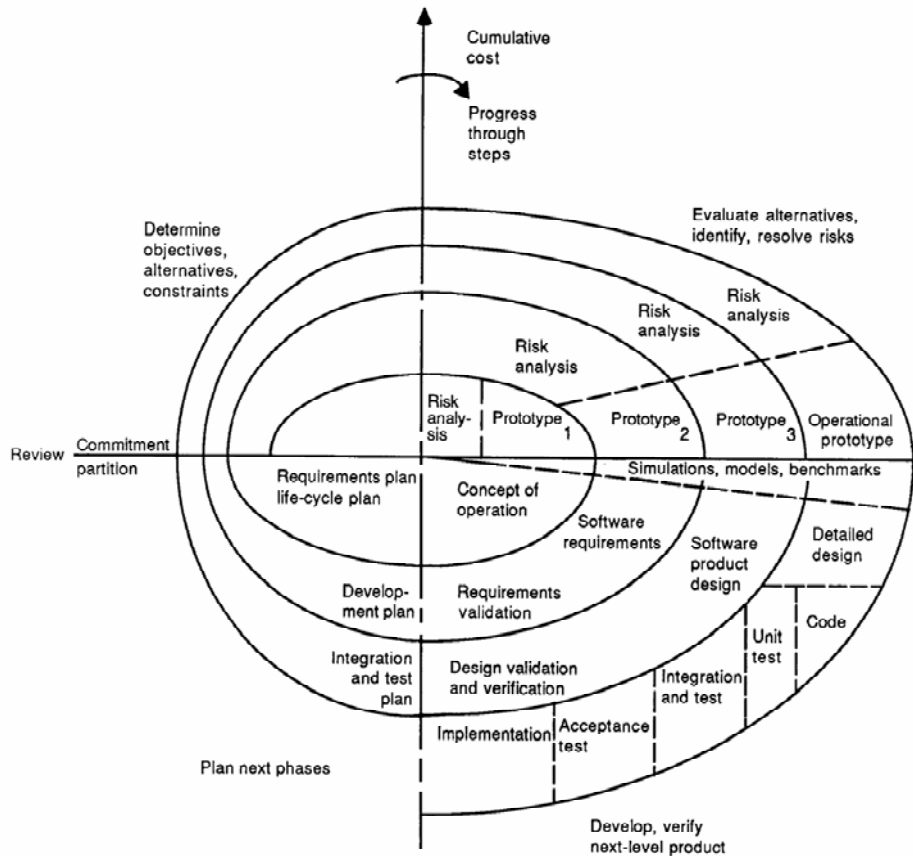
Figure 5. Spiral Life Cycle Model [15]

This life cycle model consists of the following four repetitive phases:

❑ Planning: it is based on the requirements.

❑ Risk analysis: the decision of continuing with the development is taken according to the requirements.

Risk analysis is included as a step in the development process. This analysis serves as a way of evaluating each version of the system to determine whether development should continue. If the customer decides that any identified risks are too great, the project may be halted. For example, if a substantial increase in cost or project completion time is identified during one phase of risk analysis, the customer or the developer may decide that it does not make sense to continue with the project, since the increased cost or lengthened timeframe may make continuation of the project impractical or unfeasible.

Risk management [52] is increasingly seen as one of the main jobs of project managers. It involves anticipating risks that might affect the project schedule or the quality of the software being developed and taking action to avoid these risks. The results of the risk analysis should be documented in the project plan along with an analysis of the consequences of a risk occurring.

There are three related categories of risk:

➢ Project risks are risks that affect the project schedule or resources (e.g., the loss of an experienced designer).

➢ Product risks are risks that affect the quality or performance of the software being developed (e.g., the failure of a purchased component to perform as expected).

➢ Business risks are risks that affect the organization developing or procuring the software (e.g., when a competitor introduces a new product in the market).

❑ Implementation: a prototype based on the requirements is developed.

❑ Evaluation: the user evaluates the prototype. If the user is satisfied, the project ends; if not, new requirements are included in the following iteration.

In a nutshell, the main difference between the spiral model and other life cycle models is the explicit recognition of risk. Informally, risk simply means something that can go wrong [52]. This model has an approach driven by the risk. In each cycle, a risk analysis is carried out to identify possible situations causing the project failure and to focus first on the software aspects with more risk.

2.3.8. Extreme programming life cycle model

The Extreme Programming[1] (XP) [13] is a lightweight methodology for software development, which emphasizes customer involvement and team work, and targets small to medium sized teams building risky software projects with dynamic requirements.

Key aspects of XP are: customer satisfaction, daily releases and tests. The software project is managed as a small pieces puzzle, a way to approach software development that mainly differentiates XP from other traditional software development methodologies.

Software requirements are collected by means of *user stories* that are written by the customers as "things that the systems has to do". User stories consist of two or three sentences and drive the creation of acceptance tests.

The four basic activities in the XP life cycle model are *coding*, *testing*, *listening*, and *designing.* There is no special order for them, all happen in parallel.

❑ Coding means to constantly create the simplest possible solutions for tasks based on single user stories, or possibly a few related stories at once. Coding is performed by pairs of programmers working together.

❑ Testing means creating executable test scripts even before coding starts. They are run daily, and the number of them that succeed measure the velocity of the project.

❑ Listening means meeting users regularly, and taking down user stories from them. These form the basis of both coding and testing.

❑ Designing means to apply design patterns and going over the code and 'refactoring' it in order to remove any duplications or inelegant constructs.

XP methodology is an iterative process. It consists in applying 12 basic rules, which are also called XP practices.

❑ Planning game[2]: is related to the project planning activity and is aimed at defining a release plan (user stories to be implemented for each release, and their dates). The planning activity is treated as a game with a goal, playing pieces, players, and rules for allowable moves. The playing pieces are the user stories. They are estimated, and based on such estimates a number of them are put into production. The goal of the planning game is to put the larger number of stories in production over the game time. The Planning game is an iterative process itself, and has three phases: *exploration*, *commitment*, and *steering*. The release plan provides the set of stories from which the customer will choose during the iteration planning meeting whose aim is to plan releases for the next iteration.

---

[1] http://www.extremeprogramming.org/

[2] http://c2.com/cgi/wiki?PlanningGame

- ❑ Small releases: the approach is to release new versions of the system quickly and continuously. The system is split into small pieces that are planned to be released every two weeks.

- ❑ Metaphor: the project development is guided by a shared metaphor for the system. This approach simplifies the definition of naming/coding conventions and allows developers to share the intuition of how the overall system works.

- ❑ Simple design: the design should be simple and intuitive; the use of design pattern is key.

- ❑ Testing: the development is *test-driven.* Developers write and run test first, and then code. Each user stories has its set of acceptance tests to be satisfied. Until this happens, the user story is not considered complete.

- ❑ Refactoring: it has to be performed whenever needed and possible, in order to keep the code clean and to improve system design.

- ❑ Pair programming: means that two programmers develop software side by side at one computer. Costs and benefits of pair programming are discussed and shown in [18]. Specifically, the authors show through a project experience that many mistakes are caught on the fly, the end result contains less defects, design is better and code is shorter, people learn more about software development and share knowledge about all pieces of the system, people learn team working and enjoy their work more. What is more, these benefit cost approximately the 15% that is repaid in shorter and less expensive testing, and quality assurance.

- ❑ Collective ownership: in theory, all programmers work on all pieces of the system.

- ❑ Continuous integration: integration is performed every few hours as well as testing and building.

- ❑ 40-hours weeks: never work overtime.

- ❑ Onsite customer: direct interaction with and involvement of the customer.

- ❑ Coding standards: the use of coding patterns is key, as well as the use of design patterns. They help the entire team to read and refactor code.

Figure 6 (based on that one appearing in the XP web site[3]) shows the XP methodology. As a first step, the team develop a *Spike Architecture* which is a disposable prototype. It is a very simple program used in order to explore potential solutions and evaluate risks. Based on the *Release Planning* the needed iterations are performed. Iterations are meant to address a number of user stories that are implemented by following the four basic life cycle model activities, i.e. by following a test-driven approach. Iterations include an *Iteration Plan* activities for the next iteration to perform. The software that is developed during an iteration is presented for judgment. The evaluation is performed by means of the *Acceptance Tests* that have been previously defined by customers for the user stories. If the software passes the acceptance test, it is released.

---

[3] http://www.extremeprogramming.org/

![NeOn]
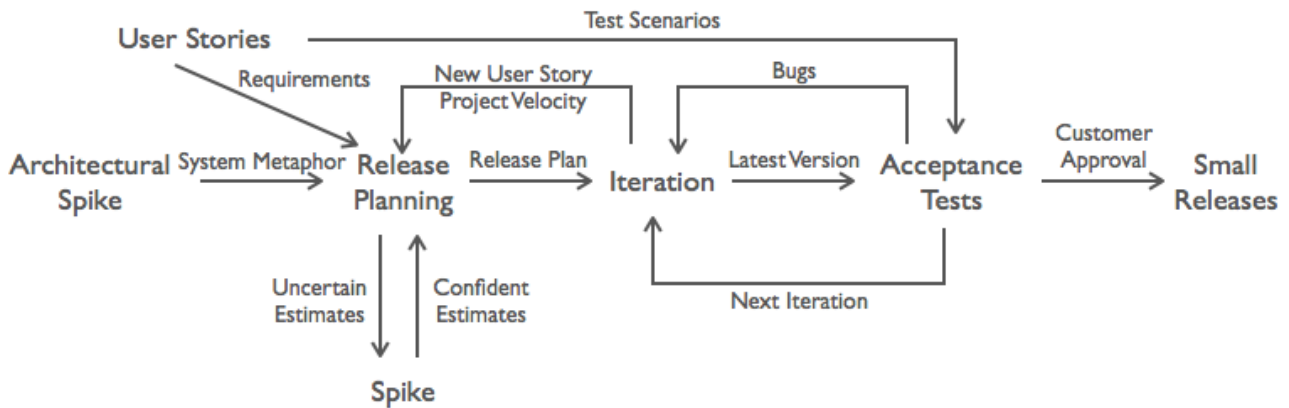
Figure 6. Extreme Programming Methodology

XP methodology is particularly targeted to small to medium sized teams building software with vague and/or rapidly changing requirements. XP is an agile and very light methodology, it emphasizes customer satisfaction but does not stress the production of detailed design documentation. For this reason it is not particularly adequate for use by organizations dealing with life-critical or high-reliability systems.

2.3.9. Reuse components life cycle model

Reuse components life cycle model [19, 38] is the discipline that attempts to reduce development costs by incorporating previously proven designs and code in new software products. The basic premise behind this model is that systems should be built using existing components, as opposed to custom-building new components. The net effect of reusing components would be shorter development schedules and more reliable software since the developer is using components that have been previously "shaken down".

The reuse-oriented approach relies on a large base of reusable software components and some integrating framework for these components. While the initial requirements specification stage and the validation stage are comparable with other processes in other life cycle models, the intermediate stages in a reuse-oriented approach are different. Such intermediate stages includes: component analysis, requirements modification, system design with reuse, and development and integration.

The advantages of reusing software components are obvious [52]. If it is possible to find and use a software component that fulfils the requirements at hand, it will in general cheaper and include more functionality than an in-house component would. Furthermore, its quality is known and it is immediately available. With this approach the amount of software to be developed is reduced and so cost and risks are also reduced. It usually also leads to faster delivery of the software.

However, requirements compromises are inevitable and this may lead to a system that does not meet the real needs of users. Furthermore, some control over the system evolution is lost as new versions of the reusable components are not under the control of the organisation using them [52].

Clearly, what is needed are techniques to create reusable components, techniques and tools to store and retrieve reusable components, and component specification techniques to help catalog and locate relevant components.

2.4. Knowledge Engineering: Development Process

The major difference between Knowledge Engineering (the science of developing knowledge-based software systems) and Software Engineering is the requirement for knowledge engineers to capture, represent, analyse and exploit knowledge in order to produce a successful knowledge-based system [41].

Both disciplines (Software Engineering and Knowledge Engineering) has the similar main goal, that is to turn the process of developing systems (classical and knowledge-based, respectively) from an art into an engineering discipline. This requires the analysis of the building and maintenance process itself and the development of appropriate methods, languages and tools for developing systems [56].

A methodological development of a project requires the definition and standarization of development and maintenance processes (from the requirements specification to the maintenance of the final product) and of the life cycle models to be within the project.

Based on Software Engineering field, some methodologies for building knowledge-based systems (KBSs) have been proposed, taking into account the features of this kind of systems. Such methodologies provides the complete life cycle for the development process, including guidelines to be followed in the different activities of the process.

In this deliverable we include briefly the development processes proposed by two Knowledge Engineering methodologies: CommonKADS and IDEAL.

In the **CommonKADS methodology** [50], KBS development entails constructing a set of six engineering models of problem solving behaviour in its concrete organization and application context. This modeling concerns not only expert knowledge, but also the various characteristics of how that knowledge is embedded and used in the organizational environment. A KBS, then, is a computational realization associated with a collection of these models. The models to be developed are:

- The *organization model*, which models the main features of the organization developing the KBS.

- The *task model*, which models the different tasks to be supported by the application being developed.

- The *expertise model*, which models the problem solving behaviour of an agent in terms of the knowledge that is applied to perform a certain task.

- The *agent model*, which models the features and capabilities of the agents (people, information systems, etc.) carrying out tasks.

- The *communication model*, which models the communication between the agents involved in the resolution of a task.

- The *design model*, which models the technical aspects of the KBS.

The aforementioned models are considered not as "steps along the way", but as independent products in their own right that play an important role during the development process of the KBS [50].

In the **IDEAL methodology** [34], KBS development and maintenance processes consists of the following five main phases, which includes several stages.

- *Phase I: Identification of the tasks to be developed*. In this phase the KBS objectives and problem characteristics are defined and the systems requirements are specified. A viability study is also carried out in this phase.

- *Phase II: Development of the different prototypes for the different subproblems.* This phase includes the following main stages for each prototype: (a) knowledge acquisition, (b) conceptualization and formalization, (c) implementation and (d) evaluation and validation.

- *Phase III: Development of the complete and integrated KBS.*

- *Phase IV: Perfective maintenance of the KBS.*

- *Phase V: Proper technology transfer.* In this phase, the technology transfer is organized and the documentation of the KBS is completed.

2.5. Knowledge Engineering: Life Cycle Models

When building a knowledge-based system (KBS) or an expert system (ES), the aim is to model the behaviour of the experts working in their domain of expertise. In this type of systems it is very difficult to establish the requirements to design a system that models the subjective expert behaviour [39]. This implies that it is difficult to establish a priori the requirements and less so a definition of specifications. So, it is necessary to use a process of progressive improvement to define the requirements gradually. In addition, the operative product will never be completely terminated since its use by the experts should improve their services which should in turn lead to a new version of the system, thus producing a feedback cycle which, at least to begin with, although toned down, is positive and therefore unending.

In this deliverable we include briefly the KBS life cycle models proposed by two Knowledge Engineering methodologies: CommonKADS and IDEAL.

In CommonKADS methodology [50], the life cycle model proposed is a **cyclic, risk-driven model similar to Boehm's life cycle model** (presented in Section 2.3.7). For each project a specialized life cycle, based on the life cycle model proposed, is configured depending on specific project objectives and risks.

In a CommonKADS project, the KBS development [50] usually consists of several cycles, depending on the identification of new objectives and risks. Steps within a cycle can be repeated many times. The CommonKADS model set (described in Section 2.4) provides a comprehensive and organized collection of aspects that can be relevant in a KBS project. However, this does not mean that in an actual project all models have to be fully developed; only those model components and states that bear on the project objectives and risks are selected. Whenever possible, parallel development of models is encouraged. Models must be maintained over the life cycle of the KBS. Control of quality and progress is integrated through regular checking of model states that must be reached in each cycle.

Figure 7 gives a stylized representation of how project management and development works are connected through model states. At the start of a management cycle, objectives for the cycle are defined, and associated risks are identified. From these objectives and risks, a set of model states is derived that must be realized within the cycle. These target model states are projected onto development activities that should result in "filling" elements of the CommonKADS models.
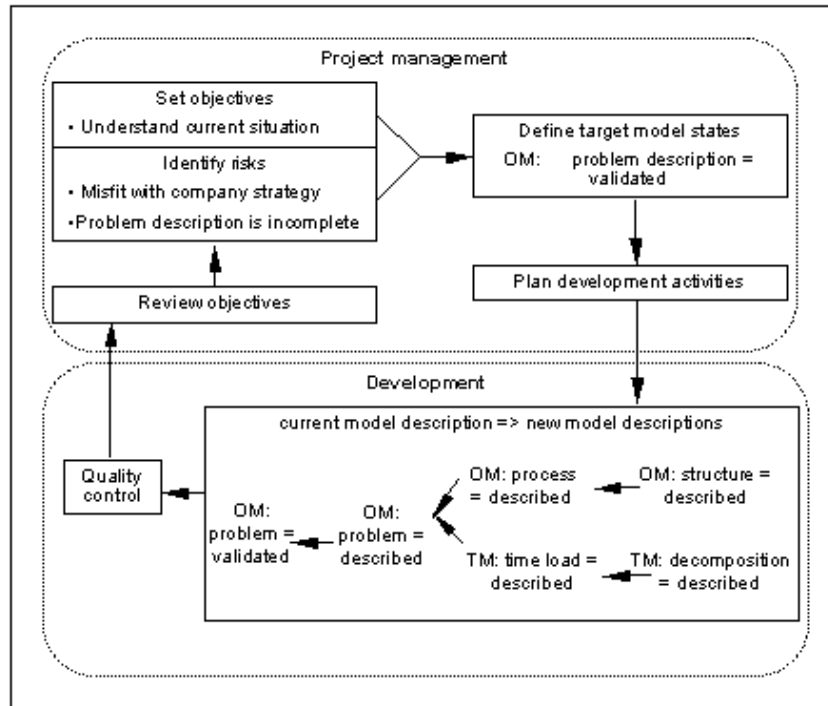
Figure 7. Example of CommonKADS Cycle [50]

The IDEAL methodology [34] proposes a **conical-spiral life cycle model** (shown in Figure 8), based on Boehm's spiral model but in three dimensions, for the case of KBS [39]. The authors [39] claim that a KBS life cycle model cannot be defined by means of the two axis of the classical software (cost and time), but by the addition of a new axe, which reflects the intrinsic characteristic of knowledge. This new axe could be the quality of knowledge which, like cost and time, increases during the life of a KBS. The third dimension of the conical-spiral life cycle model would correspond to *adaptative or perfective maintenance*, i.e., to the incorporation of new knowledge that the expert acquires in time into the system. Since knowledge generates always new knowledge, a kind of adaptative maintenance, where new knowledge is added to the system as experts obtain new experience, is always necessary. Perfective or adaptatitve maintenance means to build in new knowledge acquired by the expert at a later stage. This would call for another spiral life-cycle with shorter stages and smaller prototypes, located on another plane, to produce the spiral cone [11].

(a) Top view.   (b) Side view.
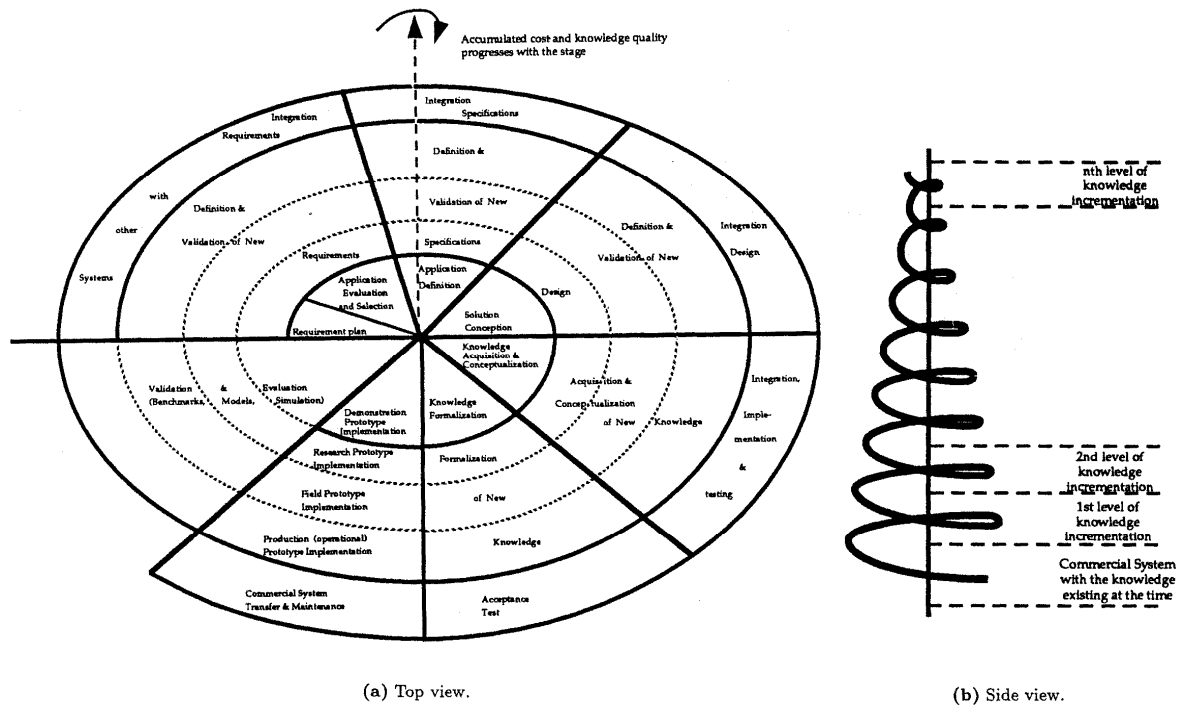
Figure 8. Conical-Spiral Life Cycle Model [11]

The conical-spiral life cycle model would have the development of the ES on one plane, based on a specific level of expert knowledge. Its development would involve the use of prototypes to define the initial system and to refine the requirements and specifications, obtaining a review and operational prototype system [11]. The number of prototypes to be developed will depend on cost analysis and problem complexity. Later, the ES would be integrated into a general purpose system to make a commercial system [11].

In short, according to the authors [12], a life cycle model for an ES with a spiral structure and conical prototype, based on a modular design with small well-defined modules and on modules which are easily related to the problem and independent, would make it possible to obtain a highly maintainable, adaptable, and flexible product, i.e., a minimum-cost system which is the aim of all methodologies.

2.6. Ontology Engineering: Development Process

2.6.1. Ontology development process in METHONTOLOGY

The ontology development process [26, 14] was identified on the framework of the **METHONTOLOGY methodology** for ontology construction. Such a proposal was based on the IEEE standard for software development [3]. The ontology development process was defined to refer to *which* activities are performed when building ontologies. It is crucial to identify these activities if agreement is to be reached on ontologies built by geographically distant cooperative teams. Such activities are organized into the three following categories, as Figure 9 shows: management, development and support.
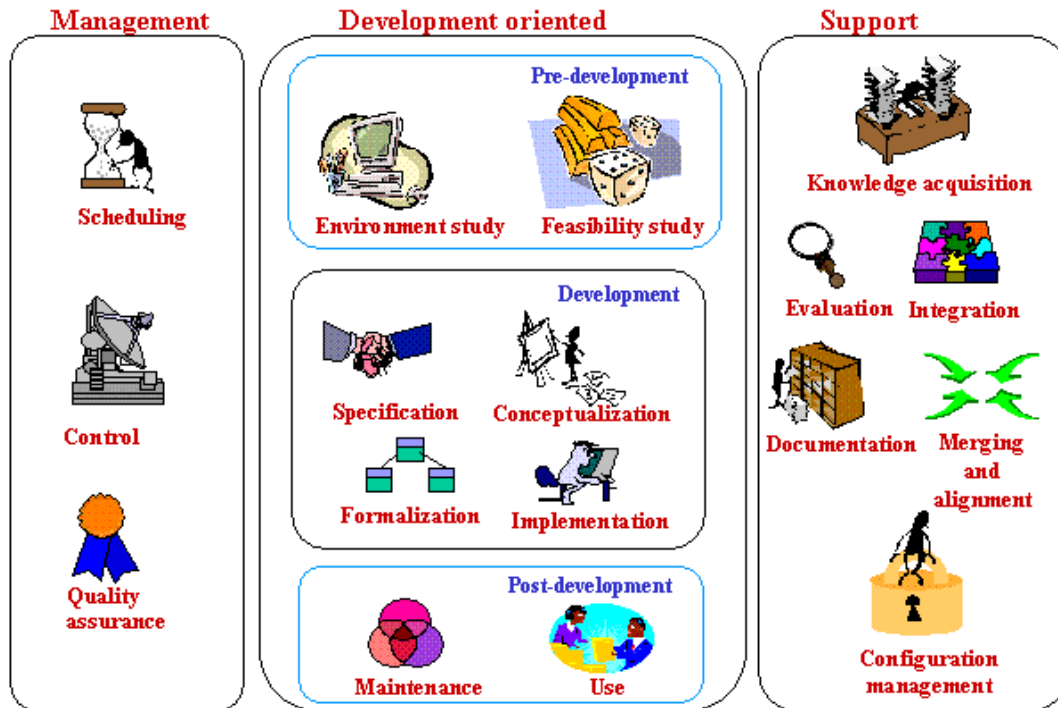
Figure 9. METHONTOLOGY Ontology Development Process

1. **Ontology management activities** include activities that initiate, monitor, and control an ontology project throughout its life cycle. These activities are:

   ❑ The *scheduling* activity identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion. This activity is essential for ontologies that use ontologies stored in ontology libraries or for ontologies that require a high level of abstraction and generality.

   ❑ The *control* activity guarantees that scheduled tasks are completed in the manner intended to be performed.

   ❑ The *quality assurance* activity assures that the quality of each and every product output (ontology, software and documentation) is satisfactory.

2. **Ontology development oriented activities** are grouped, as presented in Figure 9, into pre-development, development and post-development activities.

   The ***pre-development activities*** include activities that explore and allocate requirements before ontology development can begin. These activities are:

   ❑ An *environment study* is carried out to know the platforms where the ontology will be used, the applications where the ontology will be integrated, etc.

   ❑ A *feasibility study* answers questions such as: yours it possible to build the ontology? Is it suitable to build the ontology?, etc.

   The ***development activities*** include activities performed during the development and enhancement of an ontology project. Such activities are:

   ❑ The *specification* activity states why the ontology is being built, what its intended uses are and who the end-users are.

   ❑ The *conceptualization* activity structures the domain knowledge as meaningful models at the knowledge level [45].

> ❑ The *formalization* activity transforms the conceptual model into a formal or semi-computable model.
>
> ❑ The *implementation* activity builds computable models in an ontology language.

The ***post-development activities*** include the *maintenance* activity, that updates and corrects the ontology if needed, and the *(re)use* activity, that refers to the (re)use of the ontology by other ontologies or applications.

3. **Ontology support activities** include activities that are necessary to assure the successful completion of an ontology project. This group includes a series of activities performed at the same time as the development-oriented activities, without which the ontology could not be built. Such activities are:

❑ The *knowledge acquisition* activity whose goal is to acquire knowledge from experts of a given domain or through some kind of (semi)automatic process, which is called ontology learning.

❑ The *evaluation* activity makes a technical judgment of the ontologies, of their associated software environments, and of the documentation. This judgment is made with respect to a frame of reference during each stage and between stages of the ontology's life cycle.

❑ The *integration* activity is required when building a new ontology by reusing other ontologies already available.

❑ The *merging* activity consists in obtaining a new ontology starting from several ontologies on the same domain. The resulting ontology is able to unify concepts, terminology, definitions, constraints, etc., from all the source ontologies. The merge of two or more ontologies can be carried out either in run-time or design time.

❑ The *alignment* activity establishes different kinds of mappings (or links) between the ontologies involved. Hence this option preserves the original ontologies and does not merge them.

❑ The *documentation* activity details, clearly and exhaustively, each and every one of the completed stages and products generated.

❑ The *configuration management* activity records all the versions of the documentation and of the ontology code to control the changes.

From this analysis we can conclude that the ontology development process identifies which activities are to be performed. However, it does not identify the order in which the activities should be performed [26, 3].

2.6.2. Ontology development process in On-To-Knowledge

The **On-To-Knowledge methodology** [53] for building ontologies proposes to build the ontology taking into account how the ontology will be used in further applications. Consequently, ontologies developed with this methodology are highly dependent of the application. The processes proposed by this methodology can be summarized as follows:

1. **Feasibility study**. On-To-Knowledge adopts the kind of feasibility study described in the CommonKADS methodology [51]. According to On-To-Knowledge, the feasibility study is applied to the complete application and, therefore, should be carried out before developing the ontologies. In fact, the feasibility study serves as a basis for the kickoff process.

2. **Kickoff**. The result of this process is the ontology requirements specification document that describes the following: the domain and goal of the ontology; the design guidelines (for instance, naming conventions); available knowledge sources (books, magazines, interviews, etc.); potential users and use cases as well as applications supported by the ontology.

Another outcome of this process is a semi-formal description of the ontology.

In the kickoff process the developers should look for potentially reusable ontologies already developed.

3. **Refinement**. The goal here is to produce a mature and application oriented "target ontology" according to the specification given in the kickoff process. This refinement process is divided into two activities:

- ❑ *Activity 1: Knowledge elicitation process with domain experts*. The baseline ontology, that is, the first draft of the ontology obtained in process 2, is refined by means of interaction with experts in the domain. When this activity is performed, axioms are identified and modeled. During the elicitation, the concepts are gathered on one side and the terms to label the concepts on the other. Then, terms and concepts are mapped.

- ❑ *Activity 2: Formalization*. The ontology is implemented using an ontology language. Such language is selected according to the specific requirements of the envisaged application.

4. **Evaluation**. The evaluation process serves as a proof of the usefulness of the developed ontologies and their associated software environment. The product obtained is called ontology based application. During this process two activities are carried out:

- ❑ *Activity 1: Checking the requirements and competency questions*. The developers check whether the ontology satisfies the requirements and "can answer" the competency questions.

- ❑ *Activity 2: Testing the ontology in the target application environment*. Further refinement of the ontology can arise in this activity. This evaluation process is closely linked to the refinement process. In fact, several cycles are needed until the target ontology reaches the envisaged level.

2.6.3. Ontology development process in DILIGENT

The **DILIGENT methodology** [47] is intended to support domain experts in a distributed setting to engineer and evolve ontologies. The ontology development process proposed by this methodology includes the following five main activities:

1. **Build.** The build phase aims at creating an initial version of the ontology quickly, so that the stakeholder can start using the ontology soon [21]. Having *domain experts*, *users*, *knowledge engineers* and *ontology engineers* building an initial ontology. The team involved in building the initial ontology should be relatively small, in order to more easily find a small and consensual first version of the shared ontology. Moreover, we do not require completeness of the initial shared ontology with respect to the domain.

2. **Local adaptation.** During the next phase users locally adapt the ontology according to their own needs, while the ontology is in use e.g. to organise knowledge [21]. Once the product is made available, users can start using it and locally adapting it for their own purposes. Typically, due to new business requirements, or user and organization changes, their local ontologies evolve in a similar way as folder hierarchies in a file system. In their local environment they are free to change the reused shared ontology. However, they are not allowed to directly change the ontology shared by all users. Furthermore, the control board collects change requests to the shared ontology.

3. **Analysis.** The analysis phase requires the ontology control board to evaluate the changes suggested by the stakeholders [21]. The board analyzes the local ontologies and the requests and tries to identify similarities in users' ontologies. Since not all of the changes introduced or requested by the users will be introduced, a crucial activity of the board is deciding which changes are going to be introduced in the next version of the shared ontology. The input from users provides the necessary arguments to underline change requests. A balanced decision that takes

into account the different needs of the users and meets user's evolving requirements has to be found.

4. **Revision.** Then, the board revises the ontology by deciding which changes are applied to the ontology [21]. The board should regularly revise the shared ontology, so that local ontologies do not diverge too far from the shared ontology. Therefore, the board should have a well-balanced and representative participation of the different kinds of participants involved in the process.

5. **Local update.** In the last step the stakeholders update their local ontologies based upon the revised version of the ontology [21]. Once a new version of the shared ontology is released, users can update their own local ontologies to better use the knowledge represented in the new version. Even if the differences are small, users may rather reuse e.g. the new concepts instead of using their previously locally defined concepts that correspond to the new concepts represented in the new version.

2.7. Ontology Engineering: Life Cycle Models

2.7.1. Ontology life cycle model in METHONTOLOGY

METHONTOLOGY methodology [26, 14] was developed within the Ontology Engineering Group at Universidad Politécnica de Madrid. This methodology enables the construction of ontologies at the knowledge level. METHONTOLOGY is based on the main activities identified by the software development process [3] and in Knowledge Engineering methodologies [58, 34]. This methodology includes: the identification of the ontology development process (already presented in Section 2.6), a life cycle based on evolving prototypes (to be presented in this section), and techniques to carry out each activity in the management, development-oriented, and support activities (to be presented in the state of the art in D5.4.1).

As mentioned in Section 2.6, the ontology development process does not identify the order in which the activities should be performed. This is the role of the ontology life cycle. The ontology life cycle identifies *when* the activities should be carried out, that is, it identifies the *set of stages* through which the ontology moves during its life time, describes what activities are to be performed in each stage and how the stages are related (relation of precedence, return, etc.). METHONTOLOGY proposes an ontology building life cycle based on *evolving prototypes* because it allows adding, changing, and removing terms in each new version (prototype). For each prototype, METHONTOLOGY proposes to begin with the schedule activity that identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion. After that, the ontology specification activity starts and at the same time several activities begin inside the management (control and quality assurance) and support processes (knowledge acquisition, integration, evaluation, documentation, and configuration management). All these management and support activities are performed in parallel with the development activities (specification, conceptualization, formalization, implementation and maintenance) during the whole life cycle of the ontology.

Once the first prototype has been specified, the conceptual model is built within the ontology conceptualization activity. This is like assembling a jigsaw puzzle with the pieces supplied by the knowledge acquisition activity, which is completed during the conceptualization. Then the formalization and implementation activities are carried out. If some lack is detected after any of these activities, we can return to any of the previous activities to make modifications or refinements.

Figure 10 shows the ontology life cycle proposed in METHONTOLOGY, and summarizes the previous description. Note that the activities inside the management and support processes are carried out simultaneously with the activities inside the development process.
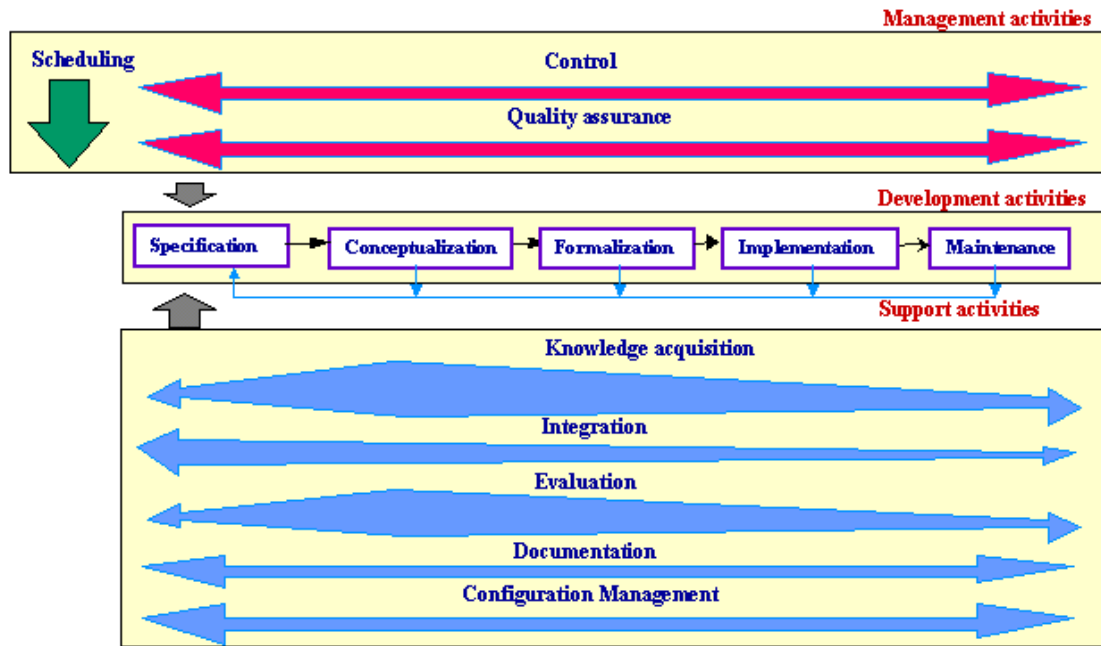
Figure 10. METHONTOLOGY Ontology Life Cycle

Related to the support activities, Figure 10 also shows that the knowledge acquisition, integration and evaluation is greater during the ontology conceptualization, and that it decreases during formalization and implementation. The reasons for this greater effort are:

❑ Most of the knowledge is acquired at the beginning of the ontology construction.

❑ The integration of other ontologies into the one we are building is not postponed to the implementation activity. Before the integration at the implementation level, the integration at the knowledge level should be carried out.

❑ The ontology conceptualization must be evaluated accurately to avoid propagating errors in further stages of the ontology life cycle.

The relationships between the activities carried out during ontology development are called *intra-dependencies*, or what is the same, they define the ontology life cycle.

METHONTOLOGY also considers that the activities performed during the development of an ontology may involve performing other activities in other ontologies already built or under construction [25]. Therefore, METHONTOLOGY considers not only intra-dependencies, but also inter-dependencies. *Inter-dependencies* are defined as the relationships between activities carried out when building different ontologies. Instead of talking about the life cycle of an ontology, we should talk about crossed life cycles of ontologies. The reason is that, most of the times and before integrating an ontology in a new one, the ontology to be reused is modified or merged with other ontologies of the same domain.

2.7.2. Ontology life cycle model in On-To-Knowledge

The **On-To-Knowledge methodology** proposes an incremental and cyclic ontology life cycle, based on evolving prototypes [32]. This ontology life cycle is shown in Figure 11.
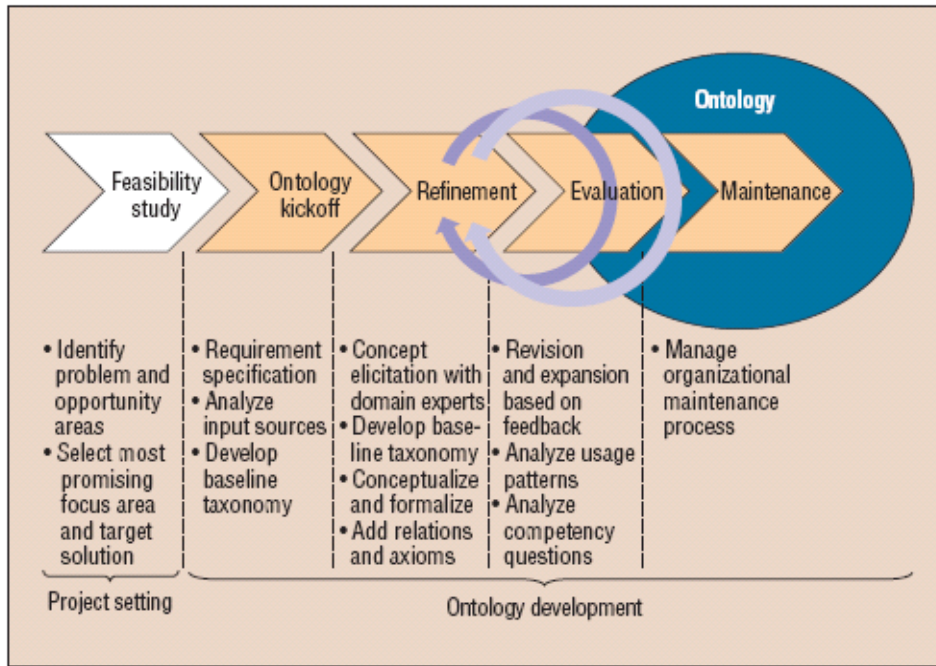
Figure 11. On-To-Knowledge Ontology Life Cycle [53]

2.7.3. Ontology life cycle model in DILIGENT

The **DILIGENT methodology** propose an ontology life cycle model based on evolving prototypes, which is shown in Figure 12.
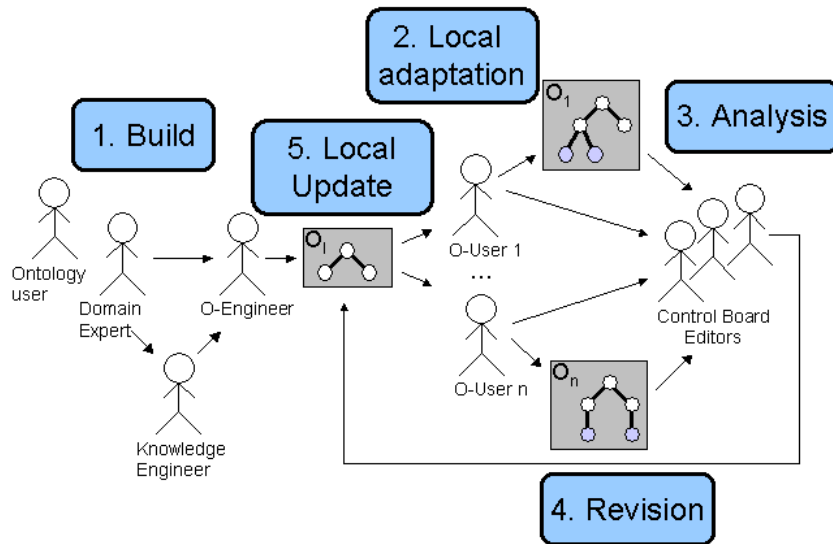
Figure 12. DILIGENT Ontology Life Cycle [47]

2.8. Conclusions

After analysing the state of the art, we can say that the degree of maturity of the ontological engineering field is very low if we compare it with the Knowledge Engineering field and, specially,

with the Software Engineering field. The long term goal of the Ontology Engineering field will be to reach a similar degree of maturity to the degree of Software Engineering field today.

Unlike what happens in the Software Engineering field, in the Ontology Engineering field:

1. **It does not exist an activity glossary that identifies and defines the activities that potentially could be carried out when single ontologies and ontology networks are developed**. Neither of the most well known methodologies (METHONTOLOGY, On-To-Knowledge and DILIGENT) includes a definition for the activities they propose. It is also remarkable that during the last years, new activities were identified when building ontologies, though they have no concrete and precise definition for new and old activities. This situation results from a lack of standardization on the ontology engineering field terminology, in contrast with the *IEEE Standard Glossary of Software Engineering Terminology* [1] in the Software Engineering field.

2. **Little research has been carried out in development process and life cycle of single ontologies and ontology networks**. METHONTOLOGY proposes explicitly the ontology development process that identifies a set of activities performed during ontology development, but such set includes also the maintenance and the use of the ontology, which is not compliant with software development process that only refers to development issues [1] and not to maintenance and use ones. Furthermore, On-To-Knowledge and DILIGENT does not include an explicit definition.

3. **Life cycle models defined in SE have not been seriously analyzed and no new life cycle models has been proposed yet to treat the special features of network of ontologies**. METHONTOLOGY proposes a unique type of life cycle model based on evolving prototypes for building single ontologies. On-To-Knowledge proposes an incremental and cyclic ontology life cycle model based on evolving prototypes, and DILIGENT proposes an ontology life cycle model based on evolving prototypes. The ontology engineering field lacks of a set of ontology life cycle models, in contrast with the software life cycle models.

4. **There are no guidelines that help software developers and ontology practitioners to select a specific life cycle model to create a particular ontology life cycle**.

5. **Existing methodologies do not cover more complex scenarios in with reuse and reengineering of ontological and non-ontological resources are needed**.

   a. METHONTOLOGY and On-To-Knowledge are up to now the most complete methodologies for building ontologies from scratch. They mainly include guidelines for single ontology construction from the ontology specification to the implementation. However, neither methodologies consider distributed ontology engineering among heterogeneous and geographically distributed groups of domain experts and ontology practitioners. DILIGENT does it, but only provides a rich argumentation framework in order to quickly proceed with building a single ontology and tracking all relevant discussions about the conceptualization activity [21].

   b. METHONTOLOGY and DILIGENT do not consider the reuse of existing information and knowledge resources for speeding up the ontology building process. In On-To-Knowledge, ontology learning methods from textual resources are used for reducing the efforts made to develop the ontology.

   c. Only METHONTOLOGY provides a definition and initial guidelines for reengineering ontologies [25, 33].

Taking into account the previous limitations and the NeOn project vision, the Ontology Engineering field needs to systematize, extend and improve the methodological work to reach a similar level of maturity than in the Software Engineering field by means of:

➢ Defining adequately the activities involved in the development process of networks of ontologies.

- ➢ Identifying a collection of life cycles models for building ontologies and ontology networks.
- ➢ Providing guidelines for establishing the ontology network life cycle.
- ➢ Supporting the collaborative construction of ontology networks by distributed teams, the reuse of existing resources (e.g., corpora, lexicons, thesauri, data bases, ontologies, etc.) for building them, and their reengineering.

3. NeOn Ontology Development Process

Ontologies are artefacts that are designed for a specific purpose to satisfy certain requirements and needs emerging in the real world.

A network of ontologies or an ontology network is defined as a collection of ontologies (called networked ontologies) related together through a variety of different relationships such as mapping, modularization, version, and dependency relationships [35].

The ontology development process can be defined (according to the Software Engineering terminology) as the process by which user needs are translated into an ontology.

So, the **ontology network development process** is defined as the process by which user needs are translated into an ontology network.

There is at least two key aspects of ontology network development process:

- ❑ The identification of when an ontology network is better than a single ontology.
- ❑ The impact of the evolution of components in a network, that is greater than in a single ontology.

Hence, the development of ontology networks is a more complex process, which has some specific features different from those of building single ontologies.

The main goal of the ontology network development process is to identify and define which activities are carried out when ontology networks are collaboratively built.

During one of the first NeOn plenary meetings, it was noticed that researchers, technology developers, and users used different terminology to name the activities involved in the ontology development process; that is, no consensus had been reached yet on many of the definitions for ontology engineering activities. For instance, it was not clear enough the difference between ontology modification [54] and ontology update [55]; and other activities had multiple definitions in natural language (e.g., ontology merging [26, 40, 42]). Additionally, we can now observe that new activities related to the Semantic Web of the future (characterized by the use of a very large number of ontologies embedded in ontology networks built by distributed teams) are emerging without a concrete and precise definition (e.g. ontology modularization). This situation is the result of a lack of standardization in the ontology engineering terminology, which clearly contrasts with the Software Engineering field that boasts the *IEEE Standard Glossary of Software Engineering Terminology* [1], a consensuated glossary.

Furthermore, to provide the methodological framework for NeOn, we need to have unambiguous understanding of the methodological components, i.e., the activities involved in the ontology network development process.

Within the NeOn consortium[4], it has been decided to build the **NeOn Glossary of Activities** for unifying the terminology used by the NeOn partners. The idea was to achieve consensus on the identification and definition of the activities involved in developing ontology networks.

The *NeOn Glossary of Activities* is exclusively focused on the activities involved in the ontology network development process. Out of the scope of this glossary are definitions of resources (data, metadata, etc.), which will be included in a Resource Glossary. The NeOn Glossary will include the NeOn Glossary of Activities, as an important and independent subset, and the Resource Glossary.

To reach a consensus in the activity terminology, we decided to use the wiki technology [44], which supports a higher level of consensus building by community members because a user who

---

[4] http://www.neon-project.org/

NeOn

disagrees with a statement can very easily modifiy it, delete it, comment it, etc [57]. We created a non public space in the NeOn wiki for discussing the ontology engineering terminology, expressing and exchanging different opinions among different partners involved in NeOn, and reaching a final agreement. Meetings and mailing lists were also employed for agreeing on the activity definitions at final stages.

For the time being, the NeOn Glossary of Activities is addressed to the Ontology Engineering community, but during the project we will include comments, examples, etc., to make the glossary more user-friendly and complete for software developers and users.

Summarizing, in this chapter we present:

1. The consensus reaching process for the identification and definition of the activities involved in the ontology network development process (Section 3.1).

2. The NeOn Glossary of Activities Version 1 (Section 3.2), which identifies and defines the activities involved in ontology network construction. Definitions in the glossary have been collaboratively built and consensuated by all NeOn partners, by means of the consensus reaching process explained in Section 3.1.

3. A classification of the activities, included in the NeOn Glossary of Activities, into those that are required for the development of ontology networks and those that are applicable (depending on the cases), but not required, and, therefore, they are non-essential or dispensable. This classification is summarized in a "required-if Applicable" table version 1 (Section 3.3.).

3.1. Consensus Reaching Process

In this section we sketch the roles and the overall process gone through by the NeOn consortium to reach a consensus on the activities for developing ontology networks. During the process we have tried to achieve a consensus on the list of activities and on the activity definitions.

Roles in the process: a varied number of skilled people, geographically dispersed (called 'NeOn Glossary' team), participated collaboratively in our consensus reaching process. The 'NeOn Glossary' team has a well-balanced and representative participation of the different people involved in the process: ontology engineers, ontology editors, and users within the NeOn project. The 'NeOn Glossary' team members had the following concrete roles (or functions):

- ❑ UPM was in charged of creating the NeOn wiki page[5] dedicated to the consensus reaching process. UPM also created and publicated in the wiki a template for gathering information about activities. UPM was in charged of creating the initial list of activities with initial definitions based on the study of the state of the art in ontology engineering.

- ❑ CNR, FAO, INRIA, iSOCO, JSI, OU, UKARL, UPM, and USFD introduced several comments in the activity definitions.

- ❑ CNR, FAO, JSI, OU, UKARL, UPM, and USFD participated in some of the ad-hoc meetings carried out to reach consensus.

Process Stages: Before beginning with the consensus reaching process, the meanings of consensus and consensus reaching process were explained. The proposed process of achieving consensus on the activities was also explained and reviewed by the 'NeOn Glossary' team. After that, the team agreed on a targeted time period (one year) to reach a consensus. And finally, the team followed the general process, shown in Figure 13, to achieve consensus on the activities involved in the development of ontology networks. We achieved consensus after the third round of the process.

---

[5] http://www.neon-project.org/wiki/index.php?title=WP5D5.3.1

Consensus is defined as a state of mutual agreement among members of a group where all opinions have been heard and addressed to the satisfaction of the group [49].

Consensus reaching process is defined as a dynamic and iterative process composed of several rounds, where the experts express and discuss about their opinions in order to reach the maximum agreement about a set of alternatives before making a decision [36].

The general process followed to achieve consensus in the activity terminology and grouping, shown in Figure 13, can be summarized as follows:

1. To create a NeOn wiki page dedicated to the consensus reaching process, within the NeOn consortium.

2. To create a template for gathering general information about activities (definition, classification, inputs, outputs, etc.) and to publish this template in the wiki.

3. To create an initial list of activities with definitions.

4. To identify and define the activities collaboratively in the wiki according to the initial list.

5. To reach a consensus on the activity definitions.

6. To publish the final results in the NeOn website[6] and in a public wiki page and to establish the procedure for getting feedback from the Ontology Engineergin (OE) community, using the argumentation tool "Cicero" (which is already a wiki).

7. To propose the standardization of the NeOn Glossary of Activities.

---

[6] http://www.neon-project.org/

```
                    ┌──────────────────────────┐
                    │    To create a wiki page  │
                    └──────────────────────────┘
                                 │
                    ┌──────────────────────────┐
                    │ To create and publish a template │
                    └──────────────────────────┘
                                 │
          ┌───────────────────────────────────────────┐
          │ To create an initial identification and definiton of activities │
          └───────────────────────────────────────────┘
                                 │
          ┌───────────────────────────────────────────┐
          │ To identify and define collaboratively the activities │
          └───────────────────────────────────────────┘
                                 │
          ┌───────────────────────────────────────────┐
          │         To reach a consensus on the definitions        │
          │   ┌──────────────────────────────┐                     │
          │   │ To include comments in the wiki │                  │
          │   └──────────────────────────────┘                     │
          │              │                                         │
          │   ┌──────────────────────────┐                        │
          │   │   To reach a consensus    │                       │
          │   │   in an ad-hoc meeting     │                      │
          │   └──────────────────────────┘                        │
          │              │                                         │
          │   ┌──────────────────────────┐                        │
          │   │   To exchange e-mails      │                      │
          │   └──────────────────────────┘                        │
          └───────────────────────────────────────────┘
                                 │
         No        ◇ Consensus achieved? ◇
                                 │ Yes
          ┌───────────────────────────────────────────┐
          │ To publish the final results in the NeOn website and │
          │ in a public wiki page; and to establish the procedure │
          │ for getting feedback from the ontology community │
          └───────────────────────────────────────────┘
                                 │
                    ┌──────────────────────────┐
                    │ To propose the standardization │
                    └──────────────────────────┘
```
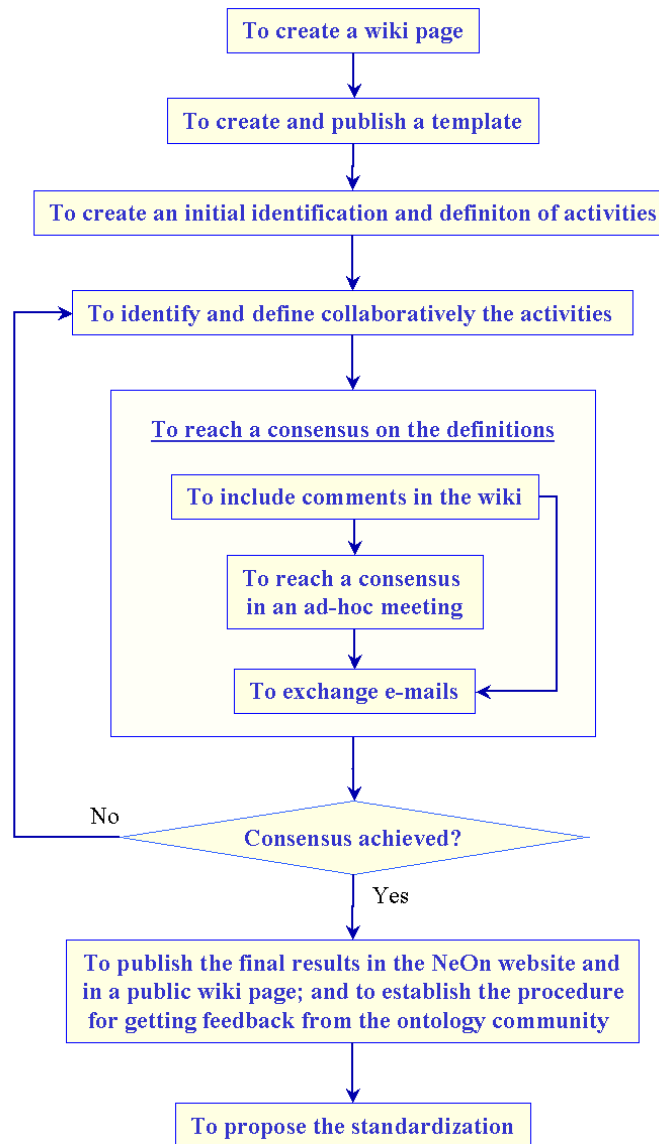
Figure 13. Consensus Reaching Process

The detailed stages of the consensus reaching process for the NeOn Glossary of Activities are the following:

1. *To create a NeOn wiki page dedicated to the consensus reaching process, within the NeOn consortium.* We used this wiki page within the NeOn consortium to build collaboratively and by consensus the NeOn Glossary of Activities. This technology was chosen because wiki web sites facilitates to reach a consensus by community members [57]. The non public wiki page was created in mid-June 2006. At such moment, the argumentation tool "Cicero" (to be included in D2.3.1 due in M20) was not available for using in the consensus reaching process.

2. *To create a template for gathering information about activities and to publish this template in the dedicated wiki page.* The template shown in Table 1 is presented to gather not only possible activity definitions but also, other general information useful for future methodological work within WP5, like the input and ouput for the activity. The template was built and published in mid-June 2006.

Table 1. Template for the Activities in the NeOn Glossary

Template Slot	Description
Activity Name	Name of the activity
Definition	One or several natural language (NL) definitions (with the corresponding references)
Type of Activity according to IEEE	In the Software Engineering field, activities are grouped administratively into five main activity groups [4]. Based on that, the following groups are proposed: (a) Ontology Management Activity (b) Ontology Pre-Development Activity (c) Ontology Development Activity (d) Ontology Post-Development Activity (e) Ontology Support Activity The type for a concrete activity should be unique
Input	A list of the required information to be input of the activity
Output	A list of the information that is required to be output of the activity
References	References for the NL definitions
Comments	Other comments about the activity

3. *To create an initial list of activities with definitions*. An initial NeOn Glossary of Activities (initial identification and definition of the main activities to be included in the ontology network development process) was made available in the wiki following the template presented in Table 1. The initial glossary was created using as starting point the ontology activities found in the following sources: METHONTOLOGY [32], On-To-Knowledge [53], DILIGENT [47], NeOn use cases [37, 30], the Semantic Web Framework (SWF) from Knowledge Web Network of Excellence[7] [28], and inputs from papers and ontology experts, as can be seen in Figure 14. The initial list of activities, definitions and grouping contained 39 activities, and they were uploaded on the wiki in mid-June 2006.
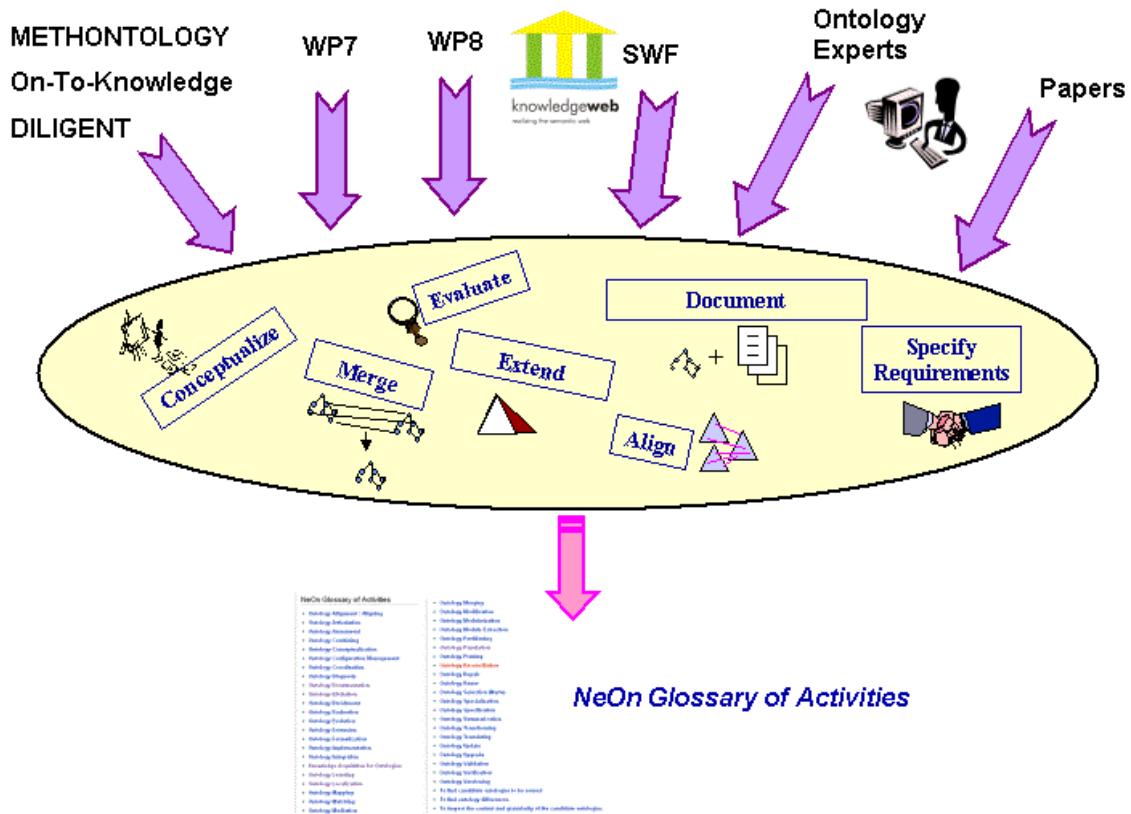
---

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

NeOn

Figure 14. Approach for Creating initially the NeOn Glossary of Activities

4. *To identify and define collaboratively the activities in the wiki, according to the initial list.* People in the 'NeOn Glossary' team were totally free to incorporate more activities and/or definitions in the initial glossary; and to include more general information about the activity (such as, input and output, classification of the activity following the groups based on IEEE [4], etc.).

5. *To reach a consensus on the activity definitions.* For this stage, we used the wiki, ad-hoc meetings and e-mails and we adopted the following process:

   a. *To include comments in the wiki*. The 'NeOn Glossary' team introduced their comments for each activity in a table in which the institutions participating in the 'NeOn Glossary' team appeared. This table was uploaded on the wiki after the Madrid meeting (mid-October 2006). In such table participants could include comments about activity definitions, i.e., the definition they prefer, in the case there were more than one; they could also include a proposal for a new definition. Figure 15 shows an example of a table of preferences for the 'knowledge acquisition' activity.

Preference Table

Partner Name	Partner Preferences
CNR	Definition: agreement with OU's definition -- two remarks: 1) does knowledge acquisition yield "formalized knowledge"? The orginal definition aimed at a pre-development activity, while OU's definition shifts the balance on development and/or use (population being use rather than development) 2) in d2.1.1 OU's definition seems to correspond to a number of functionalities (learning, upgrading database content, data annotation). Activity Group: Development and Maintanance and Use (by OU's definition). Type of Activity according to IEEE: Ontology Management and Development (by OU's defintion). Input: Knowledge Resource or Ontology Element (as in C-ODO). Output: Ontology Element (as in C-ODO).
OU	For me knowledge acquisition is a rather broad task that encompases a couple of specialised tasks which seam to be mixed up in the current definition. On one hand "knowledge elicitation" is often used when knowledge is derived from a domain expert. Then, ontology learning (acquiring TBOX) and ontology population (acquiring ABOX) are seen as two aspects of knowledge acquisition that involve extraction of ontologies (resp. instances) from structured, semi-structured and unstructured data sources. Therefore, I would rephrase the definition as follows: "Knowledge acquisition stands for a family of tasks that deal with deriving formalized knowledge from a variaty of sources. In particular, we distinguish knowledge elicitation when knowledge is acquired from a domain expert. Then, ontology learning and ontology population are knowledge acquisition tasks that rely on (semi)automatic methods to transform unstructured, semi-structured and structured data sources into ontologies and their associated instance data respectively." Development Development Input: some source of knowledge (expert, documents, etc) Output: formalized knowledge

Figure 15. Knowledge Acquisition Activity: Table of Preferences

b. *To reach a consensus in ad-hoc meetings and via e-mail*. Two ad-hoc meetings to review and agree on the definitions of activities of the NeOn Glossary took place at the Bled plenary meeting (January 23rd 2007) and at the Dubrovnik plenary meeting (June 29th 2007), respectively.

During such meetings, the following informal rules for accepting or not a concrete activity definition were taken into account:

❑ If the team's comments were generally positive and no major objections were raised, then the definition was considered as final.

❑ If general comments were positive, but someone had a major objection to the definition, the definition was modified until no major objection was encountered.

❑ If the team's comments were generally negative, the definition was ruled out.

❑ If the team's comments were mixed, there were three possibilities:

▪ discussions continued until positive or negative results were achieved;

▪ discussions were postponed until the next meeting; and

▪ the issue was postponed until more information was available in the wiki.

❑ If discussions seemed to be going on forever without the possibility of reaching an agreement, the team could:

▪ decide to drop the definition, or the activity; or

▪ move onto approval by voting the definition. The selected voting procedure was based on absolute majority.

During the **first round** of the process, at the meeting in Bled (January 23rd 2007) we reached consensus on 25 activity definitions out of 46 identified activities. The following partners participated in this meeting: JSI, OU, UKARL, and UPM. Figure 16 shows the consensuated activity definitions during the first round. Some activities have been classified following a "sub-type" approach, such as: ontology validation and ontology verification that can be seen as ontology evaluation activities. Such classification is also shown in Figure 16.
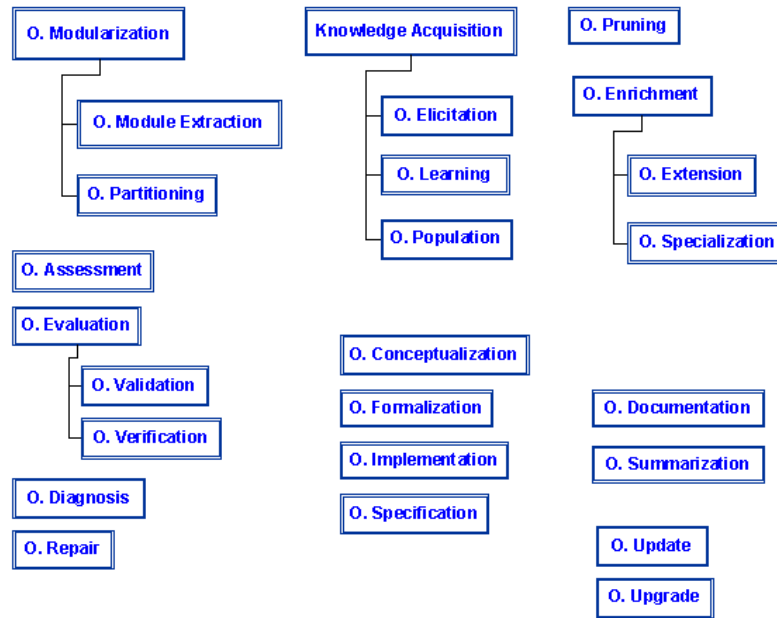

Figure 16. Activities Consensuated in the First Round on 23rd January at Bled

During the **second round** of the process, at the meeting held in Dubrovnik (29th June 2007) there were 61 identified activities, 36 activities of them without consensus. At this meeting we reached a consensus on 4 activity definitions; we also agreed on deleting 5 activities, and almost achieved a consensus on 8 activity definitions (waiting for concrete feedback on minor details and to be consensuated in the next round). The following partners participated in this meeting: CNR, FAO, JSI, UKARL, UPM, and USFD. Figure 17 shows the consensuated activity definitions during the second round. The 5 deleted activities are surrounded by a dotted line and the synonymous activities are linked by a bi-directional arrow in Figure 17.
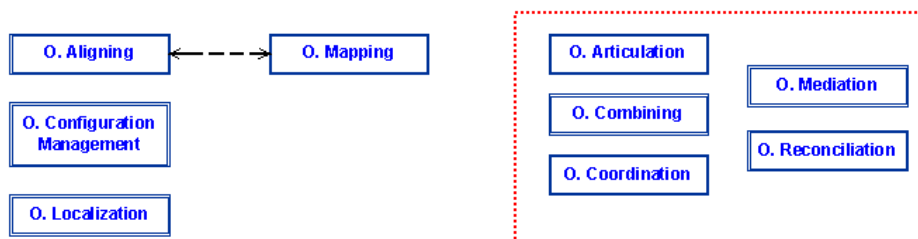

Figure 17. Activities Consensuated in the Second Round on 29th June at Dubrovnik

Furthermore, in the **third round**, it was decided to finish the consensus process on the remaining 27 activity definitions via email. Therefore, during the third round of the process (July 2007), several expert people provided feedback on the aforementioned 8 almost consensuated activities and UKARL and UPM exchanged several emails and included several comments in the wiki with the following results: 22 activity definitions consensuated

and 5 activities deleted. Figure 18 shows the activity consensuated during the third round. The 5 deleted activities are surrounded by a dotted line in Figure 18.
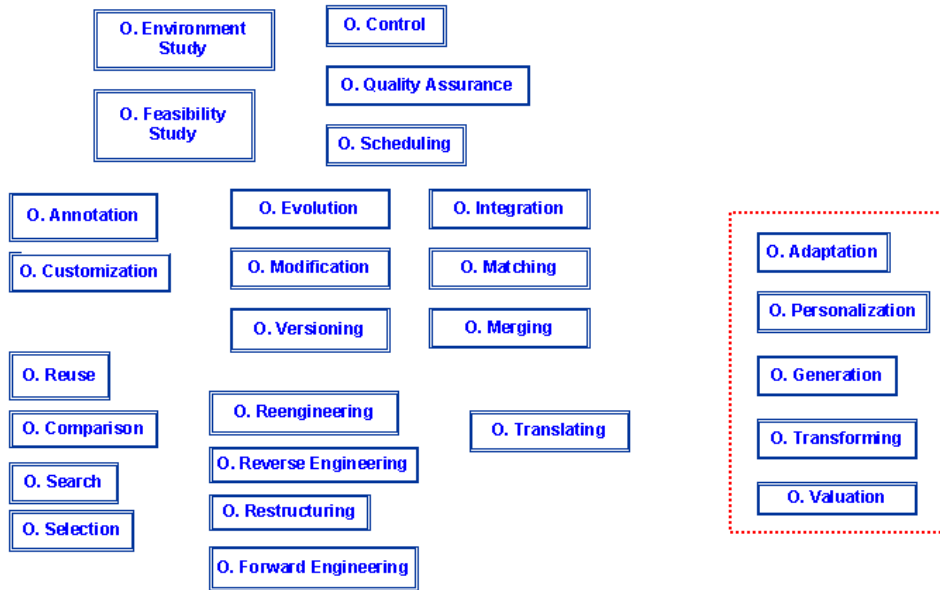

Figure 18. Activities Consensuated in the Third Round during July 2007

c. *To exchange e-mails.* Several e-mails were sent to the NeOn consortium to encourage other WPs to put forward their ideas.

Once the two ad-hoc meetings were held, WP5 partners decided that the existing definitions should be reviewed by other WPs within the NeOn consortium. Then, we identified which activities are related to which WPs, taking into account the WP main objectives, and we created and sent a table for obtaining feedback from WPs about different activities. Table 2 presents the activities reviewed and commented by WP1 to WP4. Several activities were reviewed by more than one WP. This table will be useful for the future methodological work within WP5, by means of informing about which WP is working with each activity.

The received comments were focused on differenciating or making equal some activity definitions, on including missing aspects in some activity definitions, and on deleting some activities. All the received comments were taking into account during the second and third round of the process.

Table 2. Activities Reviewed and Commented by NeOn WPs

	WP1	WP2	WP3	WP4
Ontology Alignment	*Reviewed* *Commented*		*Reviewed* *Commented*	
Ontology Articulation			*Reviewed* *Commented*	
Ontology Assessment		*Reviewed*		
Ontology Combining			*Reviewed* *Commented*	
Ontology Comparing	*Reviewed*		*Reviewed* *Commented*	*Reviewed* *Commented*

	WP1	WP2	WP3	WP4
Ontology Conceptualization		*Reviewed*		
Ontology Configuration Management	*Reviewed*			
Ontology Coordination			*Reviewed* *Commented*	
Ontology Diagnosis	*Reviewed*	*Reviewed*	*Reviewed*	
Ontology Documentation		*Reviewed*		*Reviewed*
Ontology Elicitation		*Reviewed*		
Ontology Enrichment		*Reviewed*		
Ontology Evaluation		*Reviewed*		
Ontology Evolution	*Reviewed*			
Ontology Extension		*Reviewed*		
Ontology Formalization		*Reviewed*		
Ontology Implementation		*Reviewed*		
Ontology Integration	*Reviewed*		*Reviewed* *Commented*	
Knowledge Acquisition for Ontologies		*Reviewed*		
Ontology Learning		*Reviewed*		
Ontology Localization		*Reviewed* *Commented*		*Reviewed* *Commented*
Ontology Mapping	*Reviewed* *Commented*		*Reviewed* *Commented*	
Ontology Matching			*Reviewed* *Commented*	
Ontology Mediation			*Reviewed* *Commented*	
Ontology Merging			*Reviewed* *Commented*	
Ontology Modification		*Reviewed*		
Ontology Modularization	*Reviewed*	*Reviewed*		
Ontology Module Extraction	*Reviewed*	*Reviewed*		
Ontology Partitioning	*Reviewed*	*Reviewed*		
Ontology Population		*Reviewed*		
Ontology Pruning		*Reviewed*		
Ontology Reconciliation				
Ontology Repair	*Reviewed*	*Reviewed*	*Reviewed*	
Ontology Reuse		*Reviewed*		
Ontology Searching		*Reviewed*		*Reviewed* *Commented*
Ontology Selection		*Reviewed*		

	WP1	WP2	WP3	WP4
Ontology Specialization		*Reviewed*		
Ontology Specification		*Reviewed*		
Ontology Summarization				*Reviewed*
Ontology Transforming		*Reviewed*		*Reviewed* *Commented*
Ontology Translating		*Reviewed*		
Ontology Update	*Reviewed*	*Reviewed*		
Ontology Upgrade	*Reviewed*			
Ontology Validation	*Reviewed* *Commented*	*Reviewed*		
Ontology Valuation		*Reviewed*		*Reviewed* *Commented*
Ontology Verification		*Reviewed*		
Ontology Versioning	*Reviewed*			

6. *To publish the final results in the NeOn website[8] and in a public wiki page and to establish the procedure for getting feedback from the ontology community, using the argumentation tool "Cicero".*

   After finishing the NeOn Glossary of Activities and delivering the present deliverable, we will publish the NeOn Glossary of Activities in the NeOn website and delve into the idea of obtaining feed-back from the ontology engineering community (outside NeOn). We plan to create a public wiki page with all the activities in the glossary to obtain comments from other people. We will to use the argumentation tool "Cicero" (to be included in D2.3.1 due in M20), which is already a wiki, so that the OE community could comment on the activity definitions for about a year. The long term goal is to have a more complete and consensuated glossary, which could become the terminological reference in the field.

7. *To propose the standardization of the NeOn Glossary of Activities.* If the OE community supports the activity of providing feedback on the NeOn Glossary, then we could think of approaching IEEE or W3C for the standardization of the NeOn Glossary of Activities.

Figure 19 shows the history of the consensus reaching process, including the main results in each time point.

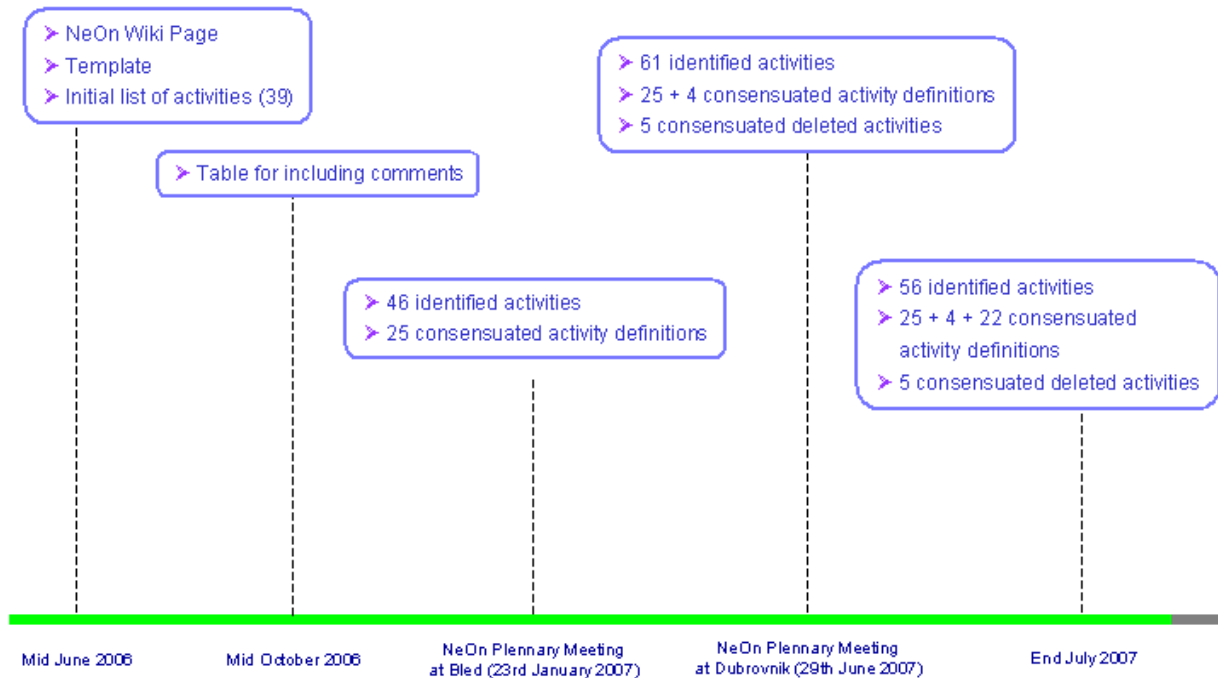---

[8] http://www.neon-project.org/

Figure 19. History of the Consensus Reaching Process

3.2. NeOn Glossary of Activities

According to ISO 12200:1999 [5] and ISO 1087-1:2000 [6], a **glossary** can be defined as:

a. A terminological dictionary containing the terminology of a specific subject field or of related fields and based on terminology work [5].

b. A terminological dictionary that contains designations and definitions from one or more specific subject fields. The vocabulary may be monolingual, bilingual or multilingual [6].

According to [43] there are three basic principles of **defining**: "avoid circularity, define every word in a definition, and make sure that every word's definition says what the word means". However, applying these basic rules does not necessarily produce good definitions.

The following principles are also common in lexicographic practice:

❑ Conciseness, i.e. every definition should say the most in the least number of words.

❑ Clarity in that it avoids ambiguity, i.e. words should be used unambiguously.

❑ Appropriateness, i.e. the definition should be appropriate to the target reader.

❑ Priority of essential traits, i.e. a definition should highlight the essential features of meaning.

Following the aforementioned definitions and principles, we have collected in the *Neon Glossary of Activities* the main activities involved in the ontology network development process (in the specific field of ontology engineering) and provided agreed natural language definitions and explanations of them. The vocabulary included in the glossary is monolingual (English).

As mentioned in the introduction, it is crucial to identify and define which main activities should be performed in the networked ontology development process. 51 activities have been identified and defined as part of the NeOn Glossary of Activities, by means of the consensus reaching process explained in Section 3.1. Apart from the 51 activities resulting of the consensus reaching process, in the NeOn Glossary of Activities two new activities (*Non Ontological Resource Reengineering*

and *Non Ontological Resource Reuse*) and their corresponding definitions have been included as result of the identification of different scenarios for building ontology networks (presented in Chapter 4). Furthermore, based on different linguistic revision and on Q.A. comments some agreed definitions have been slightly modified.

So, in this section we present the **NeOn Glossary of Activities (version 1)**, ordered alphabetically, which includes 53 activities.

Note that some of the activities have maintained their definition taken from the literature (as it is shown in the list bellow). This is the case of, for example: ontology configuration management, ontology evolution, and ontology formalization.

For most of the activities the definition have been changed and adapted based on past definitions existing in the literature and based on NeOn partners comments and discussions. This is the case of, for example, ontology conceptualization, ontology documentation, ontology matching, and ontology selection.

New activities and definitions have been created during the consensus building process, either because the activity did not yet exist in the literature or because the activity had no definition in the literature. This is the case of the following activities: ontology annotation, ontology comparison, ontology customization, ontology diagnosis, ontology elicitation, ontology enrichment, ontology extension, ontology learning, ontology localization, ontology module extraction, ontology partitioning, non ontological resource reengineering, ontology repair, non ontological resource reuse, ontology search, and ontology upgrade.

- ❑ **Ontology Aligning** refers to the activity of finding the correspondences between two or more ontologies and storing/exploiting them. A synonym for this activity is Ontology Mapping.

- ❑ **Ontology Annotation** refers to the activity of enriching the ontology with additional information, e.g. metadata or comments.

- ❑ **Ontology Assessment** refers to the activity of checking an ontology against user requirements, such as usability, usefulness, abstraction, quality, etc.

- ❑ **Ontology Comparison** refers to the activity of finding differences between two or more ontologies or between two or more ontology modules.

- ❑ **Ontology Conceptualization** refers to the activity of organizing and structuring the information (data, knowledge, etc.), obtained during the acquisition process, into meaningful models at the knowledge level according to the ontology specification document. This activity is independent of the way in which the ontology implementation will be carried out.

- ❑ **Ontology Configuration Management** [32] refers to the activity of recording all the versions of the documentation, software and ontology code, and of controling the changes.

- ❑ **Control** [32] refers to the activity of guaranteeing that scheduled activities in the ontology development process are completed in the manner intended to be performed.

- ❑ **Ontology Customization** refers to the activity of adapting an ontology to a specific user's needs.

- ❑ **Ontology Diagnosis** refers to the activity of identifying parts of the ontology directly responsible for incorrectness and incompleteness. Ontology diagnosis is triggered by ontology validation.

- ❑ **Ontology Documentation** refers to the collection of documents and explanatory comments generated during the entire ontology building process.

  Note: Examples of documents external to the implemented ontology include ontology specification documents, sources used for acquiring knowledge, ontology conceptualization document, design and decision criteria, ontological commitments, etc.

Information inside the implemented ontology includes natural language comments, ontology metadata, and implementation code.

In summary: anything that could be useful to help users, who did not build the ontology, to understand and learn how the ontology was built. Note that the level of granularity of descriptions (might) help or hinder the understanding of the ontology.

❑ **Ontology Elicitation** is a knowledge acquisition activity in which conceptual structures (e.g. T-Box) and their instances (e.g. A-Box) are acquired from domain experts.

❑ **Ontology Enrichment** refers to the activity of extending an ontology with new conceptual structures (e.g., concepts, roles, axioms, etc.).

❑ **Ontology Environment Study** refers to the activity of analyzing the environment in which the ontology is going to be developed.

❑ **Ontology Evaluation** refers to the activity of checking the technical quality of an ontology against a frame of reference.

❑ **Ontology Evolution** [54] refers to the activity of facilitating the modification of an ontology by preserving its consistency.

Note: Ontology Evolution can be seen as a consequence of different activities during the development of the ontology.

❑ **Ontology Extension** is an ontology enrichment activity for stretching the ontology in width.

❑ **Ontology Feasibility Study** [32] refers to the activity of answering questions like: is it possible to build the ontology? is it suitable to build the ontology?, etc.

❑ **Ontology Formalization** [32] refers to the transformation of a conceptual model into a formal or semi-computable model according to a knowledge representation paradigm (e.g., description logics, frames, rules, etc.).

❑ **Ontology Forward Engineering** [33] refers to the activity of outputting a new implementation of the ontology on the basis of the new conceptual model.

❑ **Ontology Implementation** refers to the activity of generating computable models according to the syntax of a formal representation language (e.g., RDF(S), OWL, FLogic, etc.).

❑ **Ontology Integration** refers to the activity of including one ontology in another ontology.

❑ **Knowledge Acquisition for Ontologies** comprises activities for capturing knowledge (e.g., T-Box and A-Box) from a variety of sources (e.g., documents, experts, data bases, etc.). We can distinguish between: Ontology Elicitation, Ontology Learning and Ontology Population.

❑ **Ontology Learning** is a knowledge acquisition activity that relies on (semi-) automatic methods to transform unstructured (e.g. corpora), semi-structured (e.g. folksonomies, html pages, etc.) and structured data sources (e.g. data bases) into conceptual structures (e.g. T-Box).

❑ **Ontology Localization** refers to the adaptation of an ontology to a particular language and culture.

❑ **Ontology Mapping** refers to the activity of finding the correspondences between two or more ontologies and storing/exploiting them. A synonym for this activity is Ontology Aligning.

❑ **Ontology Matching** refers to the activity of finding or discovering relationships or correspondences between entities of different ontologies or ontology modules.

Note: Ontology Matching can be seen as the first stage of Ontology Aligning.

❑ **Ontology Merging** refers to the activity of creating a new ontology or ontology module from two or more, possibly overlapping, source ontologies or ontology modules.

- **Ontology Modification** [54] refers to the activity of changing the ontology, without considering the consistency.

- **Ontology Modularization** refers to the activity of identifying one or more modules in an ontology with the purpose of supporting reuse or maintenance.

  Note: We can make distinctions between: Ontology Module Extraction and Ontology Partitioning.

- **Ontology Module Extraction** refers to the activity of obtaining from an ontology concrete modules to be used for a particular purpose (e.g. to contain a particular sub-vocabulary of the original ontology).

- **Ontology Partitioning** refers to the activity of dividing an ontology into a set of (not necessary disjoint) modules that together form an ontology and that can be treated separately.

- **Ontology Population** is a knowledge acquisition activity that relies on (semi-) automatic methods to transform unstructured (e.g. corpora), semi-structured (e.g. folksonomies, html pages, etc.) and structured data sources (e.g. data bases) into instance data (e.g. A-Box).

- **Ontology Pruning** refers to the activity of discarding conceptual structures (e.g., part of T-Box) of a given ontology that are not or no longer relevant.

  Note: Pruning is mostly used in combination with ontology learning methods to discard potentially irrelevant learned concepts/relations.

- **Ontology Quality Assurance** [32] refers to the activity of assuring that the quality of each and every process carried out and product built (ontology, software and documentation) is satisfactory.

- **Non Ontological Resource Reengineering** refers to the process of retrieving and transforming an existing non ontological resource (data bases, controlled vocabularies, etc.) into an ontology.

- **Ontology Reengineering** [33] refers to the process of retrieving and transforming a conceptual model of an existing and implemented ontology into a new, more correct and more complete conceptual model which is reimplemented.

- **Ontology Restructuring** [33] refers to the activity of correcting and reorganizing the knowledge contained in an initial conceptual model, and detecting missing knowledge.

  Note: This process contains two phases: analysis and synthesis. The "analysis phase goal" is to evaluate the ontology technically, that is, to check that the hierarchy of the ontology and its classes, instances, relations and functions are complete (contain all the definitions required for the domain of chemical substances), consistent (there are no contradictions in the ontology and with respect to the knowledge sources used), concise (there are no explicit and implicit redundancies) and syntactically correct. The "synthesis phase" seeks to correct the ontology after the analysis phase and document any changes made.

- **Ontology Repair** refers to the activity of resolving errors (incompleteness, incorrectness) in the ontology and it is triggered by ontology diagnosis.

- **Non Ontological Resource Reuse** refers to taking available non ontological resources (data bases, controlled vocabularies, etc.) for the development of ontologies.

- **Ontology Reuse** refers to using an ontology or an ontology module in the solution of different problems. Ontology reuse is the activity that allows employing an ontology or an ontology module in, for example, the development of new ontologies, the development of different ontology-based applications, the activity of ontology aligning (as background knowledge), etc.

- **Ontology Reverse Engineering** [33] refers to the activity of outputting a possible conceptual model on the basis of the code in which the ontology is implemented.

- ❑ **Scheduling** [32] refers to the activity of identifying the tasks to be performed during the ontology development, their arrangement, and the time and resources needed for their completion.

- ❑ **Ontology Search** refers to the activity of finding candidate ontologies or ontology modules to be reused.

- ❑ **Ontology Selection** refers to the activity of choosing the most suitable ontologies or ontology modules among those available in an ontology repository or library, for a concrete domain of interest and associated tasks.

- ❑ **Ontology Specialization** is an ontology enrichment activity for extending the ontology in depth.

- ❑ **Ontology Specification** is a collection of requirements that the ontology should fulfill, e.g. reasons to build the ontology, target group, intended uses, possibly reached through a consensus process.

- ❑ **Ontology Summarization** refers to the activity of providing an abstract or summary of the ontology content.

  Note: The summary can include, for example, a couple of top levels in the ontology class hierarchy (perhaps a graphical representation of these top-level concepts and links between them).

- ❑ **Ontology Translation** refers to the activity of changing the representation formalism or language of an ontology from one to another.

  Note: Ontology Translation can be part of an ontology reengineering process.

- ❑ **Ontology Update** refers to minor changes carried out in an ontology that could not be considered an upgrade.

- ❑ **Ontology Upgrade** refers to the activity of replacing an existing ontology with a new version.

- ❑ **Ontology Validation** is the ontology evaluation that compares the meaning of the ontology definitions against the intended model of the world aiming to conceptualize.

  Note: It answers the question: Are you producing the right ontology?.

- ❑ **Ontology Verification** is the ontology evaluation which compares the ontology against the ontology specification document (ontology requirements and competency questions), thus ensuring that the ontology is built correctly (in compliance with the ontology specification).

  Note: It answers the question: Are you producing the ontology right?.

- ❑ **Ontology Versioning** [54] refers to the activity of handling ontology changes by creating and managing different versions of the ontology.

The 53 definitions and their corresponding comments included in the NeOn Glossary of Activities have been checked and reviewed by an expert terminologist from the UPM team, from a linguistic point of view.

3.3. Required or If Applicable Activities

In order to facilitate the identification of which activities are important and neccessary for developing a software product, IEEE standards [3, 4] identify which activities are mandatory and optional when developing a particular software product.

Activities of the software development process are categorized as either mandatory or "if applicable" [3]. Mandatory activities must be always carried out, and "if applicable" ones contains an explanation of the cases to which it will apply. For example, most activities in the Project

Management activity group and some activities of the Support activity group are required, that is are mandatory for all development projects. Depending on the concrete development case and on the life cycle model chosen, activities in the Pre-Development, Development, and Post-Development activity groups may or may not be required, that is they are "If Applicable" [4].

In order to speed up the ontology network development by software developers and ontology practitioners, for each activity included in the NeOn Glossary of Activities (Section 3.2) we identify which activities are required and which activities are optional (or if-applicable) during the ontology network building process.

❑ *Required* activities refer to those activities that should be carried out when developing networks of ontologies.

❑ *If Applicable* or *Optional* activities refer to those activities that can be carried out or not, depending on the case, when developing ontology networks.

The table of 'Required-If Applicable' activities presented in this section is a first version. After delivering this document, such table will evolved and the final version of the table will be included in D5.3.2 (at month 36)

To carry out the identification of which activities are required and which ones are optionals during the ontology network building process, we invited in an open call to ontology developers from the ontology engineering community participating in projects (NeOn, KWeb, X-Media, etc.) and working in universities and companies (iSOCO, OEG group, DERI group, etc.) to participate in an on-line survey[9]. This survey/questionnaire began on July 27th 2007 and the results were collected on Agust 21st 2007. 35 people answered to this survey during the aforementioned period.

The analyzed results of this survey, taking into account also our own experience on developing ontologies, are shown in Table 3.

Table 3. Required-If Applicable Activities

	Required	If Applicable
Ontology Aligning		X
Ontology Annotation	X	
Ontology Assessment	X	
Ontology Comparison	X	
Ontology Conceptualization	X	
Ontology Configuration Management	X	
Control	X	
Ontology Customization		X
Ontology Diagnosis	X	
Ontology Documentation	X	
Ontology Elicitation	X	

---

[9] http://droz.dia.fi.upm.es/survey/index.jsp

	Required	*If Applicable*
Ontology Enrichment		X
Ontology Environment Study	X	
Ontology Evaluation	X	
Ontology Evolution	X	
Ontology Extension		X
Ontology Feasibility Study	X	
Ontology Formalization	X	
Ontology Forward Engineering		X
Ontology Implementation	X	
Ontology Integration	X	
Knowledge Acquisition for Ontologies	X	
Ontology Learning		X
Ontology Localization		X
Ontology Matching		X
Ontology Merging		X
Ontology Modification		X
Ontology Modularization		X
Ontology Module Extraction		X
Ontology Partitioning		X
Ontology Population		X
Ontology Pruning		X
Ontology Quality Assurance	X	
Non Ontological Resource Reengineering		X
Ontology Reengineering		X
Ontology Restructuring		X
Ontology Repair	X	
Non Ontological Resource Reuse		X
Ontology Reuse	X	
Ontology Reverse Engineering		X

	Required	If Applicable
Scheduling	X	
Ontology Search	X	
Ontology Selection	X	
Ontology Specialization		X
Ontology Specification	X	
Ontology Summarization		X
Ontology Translation		X
Ontology Update		X
Ontology Upgrade	X	
Ontology Validation	X	
Ontology Verification	X	
Ontology Versioning	X	

The activities identified as "required" can be considered as core for the ontology network development. Example of such activities are: ontology specification, ontology conceptualization, ontology reuse, and ontology implementation.

The other activities (the "if-applicable" ones, like ontology customization, ontology localization, and ontology aligning) are optional in the ontology development process, and they should be carried out depending on the concrete case. For example, if the ontology is developed in English, and as a requirement the ontology should be available in other natural languages (such as, French and Spanish), the ontology localization activity should be carried out.

4. Scenarios for Building Ontology Networks

Within the NeOn project, ontology networks need to be developed in three use cases:

❑ In WP7, FAO is building an ontology network for the Fishery use case by means of reusing proprietary knowledge resources, such as data bases and thesauri.

❑ In WP8, iSOCO is building an ontology network for the Invoicing Management use case from scratch, but they need to reengineer existing knowledge models into ontologies.

❑ In WP8, ATOS is building an ontology network for the Nomenclator use case by means of reusing non-proprietary ontologies and standards.

Based on the analysis of the NeOn use cases, on the different studies carried out to revise the state of the art of ontology development and on the building of ontologies in different international and national projects (Esperonto, Knowledge Web, SEEMP, SEKT, etc.), we have detected that there are different ways or paths to build ontologies and ontology networks.

In the most well-known ontology engineering methodologies (METHONTOLOGY, On-To-Knowledge and DILIGENT) the scenarios for building ontologies are very rigid. In this chapter, we provide flexibility to the ontology network development by means of presenting different alternative ways or possibilities for building networks of ontologies.

Figure 20 presents a set of 8 scenarios for building ontology networks. This set of scenarios is not exhaustive, although we think it covers the most plausible ways for building ontology networks. In this figure, activities to be carried out are represented by circles or by rounded boxes. Numbered directed arrows represent the different scenarios presented in this chapter. The figure also shows (as dotted boxes) existing resources and possible outputs (implemented ontology networks and alignments) following some of the presented scenarios.

This chapter includes, as independent sections, the most common scenarios (shown in Figure 20) that may arise during the ontology network development:

❑ Scenario 1: Building ontology networks from scratch without reusing existing resources.

❑ Scenario 2: Building ontology networks by reusing non ontological resources.

❑ Scenario 3: Building ontology networks by reusing ontologies or ontology modules.

❑ Scenario 4: Building ontology networks by reusing and reengineering ontologies or ontology modules.

❑ Scenario 5: Building ontology networks by reusing and merging ontology or ontology modules.

❑ Scenario 6: Building ontology networks by reusing, merging and reengineering ontologies or ontology modules.

❑ Scenario 7: Building ontology networks by restructuring ontologies or ontology modules.

❑ Scenario 8: Building ontology networks by localizing ontologies or ontology modules.

The activities of knowledge acquisition and elicitation, documentation, configuration management, evaluation and assessment (as shown at the bottom of Figure 20) should be carried out during the whole ontology network development.

It is worth mentioning that scenario 2 can be combined with scenarios 3-7; and that scenario 8 can be carried out or not with scenarios 1-7.
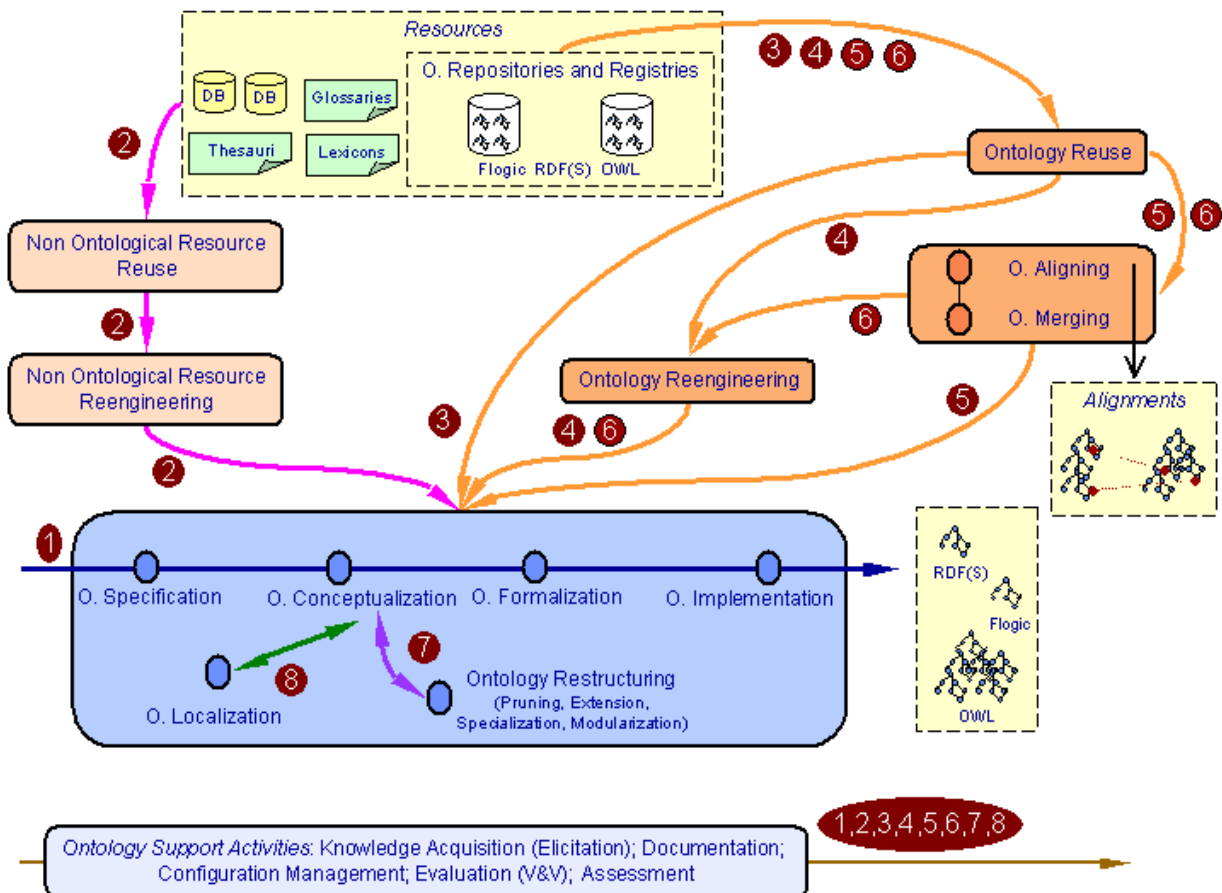
Figure 20. Scenarios for Building Ontology Networks

In this chapter each section, containing a particular scenario, is organized as follows:

❑ Assumptions for the scenario. For each of the presented scenarios, it is supposed that the scenario begins with the decision of whether a single ontology or an ontology network is going to be developed. In this chapter, it is assumed for all the scenarios that an ontology network will be built.

❑ Prerequisite resources.

❑ Sequence of activities to be carried out. The included activities are taken from the NeOn Glossary of Activities.

❑ Outcomes for the scenario.

4.1. Scenario 1: Building Ontology Networks from Scratch without Reusing Resources

❑ **Assumptions**: In this scenario it is supposed that software developers and ontology practitioners "want" to develop the ontology network from scratch (that is without reusing existing resources).

❑ **Prerequisite resources**: Knowledge about the domain (general or specific) of the ontology network to be developed should be available for the building of the ontology network.

❑ **Sequences of activities**: In this scenario (represented in Figure 20 by the arrow number 1), software developers and ontology practitioners should follow the same activities for building

ontology networks from scratch than those proposed by METHONTOLOGY [32] or On-To-Knowledge [53] for building ontologies.

The ontology network development should start with the *knowledge acquisition activity*, possibly including the *ontology elicitation activity*. This activity should be carried out during the whole development; however, software developers and ontology practitioners should acquire most of the knowledge at the beginning of the process. The level of knowledge acquisition decreases as development progresses.

Simultaneously with the knowledge acquisition activities, software developers and ontology practitioners should specify the requirements that the ontology network should fulfil, by means of the *ontology specification activity*. The purpose of the ontology specification activity is to output a document that includes the purpose, level of formality and scope of the ontology network, target group and intended uses of the ontology network.

Then, software developers and ontology practitioners should carry out the *ontology conceptualization activity*. In this activity, the domain knowledge, obtained during the knowledge acquisition activities, should be organized and structured in a conceptual model according to the ontology specification document. In this scenario reuse is not considered, as stated in the scenario assumptions.

Having completed the ontology network conceptualization, software developers and ontology practitioners should formalize the ontology network and implement it in an ontology language (e.g., OWL, RDF(S), or F-Logic), by means of the *ontology formalization* and the *ontology implementation activities*, respectively.

Ontology support activities, such as *knowledge acquisition* and *elicitation*, *documentation*, *configuration management*, *evaluation* and *assessment*, should be carried out during the whole ontology network development. The intensity of such activities depends on the concrete moment of the development progress.

In the case of the ontology evaluation activity, it can include different "sub-activities" (*ontology validation* and *ontology verification* activities) as shown in Figure 21. The ontology validation activity can be seen as a process composed of two activities (following the 'process' definition of Section 2.1): *ontology diagnosis* and *ontology repair* activities.
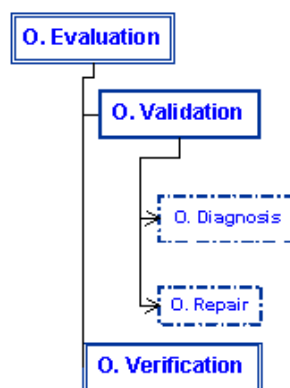

Figure 21. Ontology Evaluation Activity

❑ **Outcomes**: The principal output is a network of ontologies that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). In addition, a broad range of documents, such as the requirements specification, the ontology documentation, the ontology evaluation, etc., will be generated as output by the different activities.

4.2. Scenario 2: Building Ontology Networks by Reusing Non Ontological Resources

❑ **Assumptions**: In this scenario it is supposed that software developers and ontology practitioners "want" to develop the ontology network by means of reusing existing non ontological resources (i.e., thesauri, glossaries, etc).

❑ **Prerequisite resources**: Knowledge of the domain (general or specific) of the ontology network to be developed should be available for the building of the ontology network.

Additionally, non ontological resources, related to the domain that the ontology network must cover, should be made available.

❑ **Sequence of activities**: Currently, software developers and ontology practitioners are often considering reusing existing non ontological resources (e.g., thesauri, glossaries, data bases, etc.) for speeding up the ontology building process. The *non ontological resource reuse activity*, as defined in the NeOn Glossary of Activities (Section 3.2), refers to the activity that takes available non ontological resources (data bases, controlled vocabularies, etc.) for the development of ontologies. Being ontologies the key element in ontology engineering, we treat the reuse of ontologies as a different activity.

As Figure 20 shows (by arrows with number 2), in this scenario software developers and ontology practitioners should carry out the non ontological resource reuse activity; in this activity they should analyse whether existing resources (thesauri, glossaries, data bases, etc.) can be reused to build the ontology network or not. If software developers and ontology practitioners decide that one or more non ontological resources are useful for the ontology network development, then the *non ontological resource reengineering activity* should be carried out.

Non ontological resource reengineering is defined in the NeOn Glossary of Activities (Section 3.2) as the process of retrieving and transforming an existing non ontological resource (data bases, controlled vocabularies, etc.) into an ontology. In this scenario, the idea is to transform a non ontological resource into an ontology that could be reused in the ontology network being developed. Being ontologies the key element in ontology engineering, we treat the reengineering of ontologies as a different activity.

After carrying out the non ontological resource reengineering activity, software developers and ontology practitioners should use the output of such activity as input of some of the activities included in scenario 1 (explained in Section 4.1) as shown in Figure 20.

❑ **Outcomes**: The principal output is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Furthermore, a broad range of documents containing the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

Additionally, the non ontological resources selected to be reused have been "ontologized", by means of the non ontological resource reengineering activity.

4.3. Scenario 3: Building Ontology Networks by Reusing Ontologies or Ontology Modules

❑ **Assumptions**: In this scenario it is supposed that software developers and ontology practitioners "want" to use existing ontologies in the same or similar domain than that of the ontology network to be developed for building the ontology network. The existing ontologies may have been developed by themselves or by others.

❑ **Prerequisite resources**: Knowledge of the domain (general or specific) of the ontology network to be developed should be available for building the ontology network.

Additionally, ontologies or ontology modules, related to the domain that the ontology network must cover, should be made available.

❑ **Sequence of activities**: Existing ontologies are being collected in both ontology library systems, which offer various functions for managing, adapting and standardizing groups of ontologies [20], and ontology repositories that support knowledge access and reuse by humans and machines.

An ontology repository is a structured collection of ontologies, modules and additional meta knowledge. The software that manage an ontology repository is known as ontology repository management system; this system stores, organizes, modifies and extracts knowledge from an ontology repository. Related to this, the ontology registries should be also mentioned since such registries store descriptions of ontologies and not ontologies themselves.

As more ontologies are available in ontology library systems, in ontology repositories and on the Internet, software developers and ontology practitioners could also consider the reuse of existing ontologies or ontology modules. Domain and general ontologies can be reused to build others of more granularity and coverage.

Thus, the following activities should be carried out as part of the *ontology reuse activity*, which can be seen as a process composed of activities (following the 'process' definition of Section 2.1):

1. To search ontologies or ontology modules in repositories and registries like Knowledge Zone[10], ONTHOLOGY.org[11], Oyster[12], Swoogle[13], and WATSON[14], through the *ontology search activity*. Software developers and ontology practitioners search for candidate ontologies or modules that satisfy the requirements. These ontologies or modules could be implemented in different languages or could be available in different ontology tools.

2. To assess ontologies or ontology modules through the *ontology assessment activity*. After the ontology search, software developers and ontology practitioners must inspect the content and granularity of the ontologies or modules to find out if they satisfy their needs.

3. To compare ontologies or ontology modules through the *ontology comparison activity*. Software developers and ontology practitioners should compare the ontologies or modules found, taking into account a set of criteria identified by the software developers and ontology practitioners (e.g., ontology language, ontology coverage, existing ontology documentation, etc.) and their ontology network requirements.

4. To select ontologies or ontology modules through the *ontology selection activity*. Based on the comparisons, software developers and ontology practitioners should select the set of ontologies or modules that are the most appropriate for their ontology network requirements.

5. To define the reuse mode: at this point, software developers and ontology practitioners need to decide how they will reuse the selected ontologies or ontology modules. There are three modes:

   ▪ The selected ontologies or modules will be reused as they are.

---

[10] http://smiprotege.stanford.edu:8080/KnowledgeZone/

[11] http://www.onthology.org/

[12] http://oyster.ontoware.org

[13] http://swoogle.umbc.edu/

[14] http://watson.kmi.open.ac.uk/WatsonWUI/

> ▪ The ontology reengineering activity should be carried out with the ontologies or modules selected.

> ▪ Ontologies or modules will be merged to obtain a new ontology or module.

Before reusing the selected ontologies or modules by following any reuse mode, it is also convenient to evaluate the ontologies or modules through the *ontology evaluation activity*.

If the software developers and ontology practitioners decide to reuse the selected ontologies or modules as they are, then they integrate, by means of carrying out the *ontology integration activity*, the ontology or module codes in the ontology network code developed following the activities of scenario 1 (Section 4.1). In this case, Figure 22 shows the ontology reuse activity viewed as a process composed of the following activities: ontology search, ontology assessment, ontology comparison, ontology selection, and ontology integration.
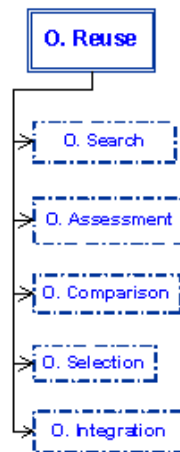

Figure 22. Ontology Reuse Activity

❑ **Outcomes**: The principal output is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

4.4. Scenario 4: Building Ontology Networks by Reusing and Reengineering Ontologies or Ontology Modules

❑ **Assumptions**: It is supposed that software developers and ontology practitioners "want" to use existing ontologies or ontology modules by reengineering them before reusing them in the ontology network building.

❑ **Prerequisite resources**: Knowledge of the domain (general or specific) of the ontology network to be developed should be available for building the ontology network.

Additionally, ontologies or ontology modules related to the domain to be covered by the ontology network to be developed should be available.

❑ **Sequence of activities**: The sequence of activities in this scenario is the same as the sequence presented in scenario 3 (Section 4.3); however, when software developers and ontology practitioners need to decide the reuse mode (that is, activity 5 in Section 4.3), they can argue that the selected ontologies or ontology modules are useful for the concrete use case. They do not want to reuse the ontologies or modules as they are but to transform them by means of the *ontology reengineering activity*, as shown in scenario 4 (Figure 20). In this

case, Figure 23 shows the ontology reuse activity viewed as a process composed of the following activities (according to the 'process' definition of Section 2.1): *ontology reverse engineering*, *ontology restructuring*, and *ontology forward engineering*.
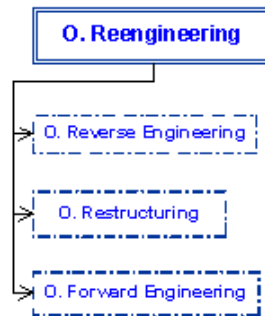

Figure 23. Ontology Reengineering Activity

Based on [17], we here propose the levels of abstraction that underlie the ontology reengineering activity shown in Figure 24.
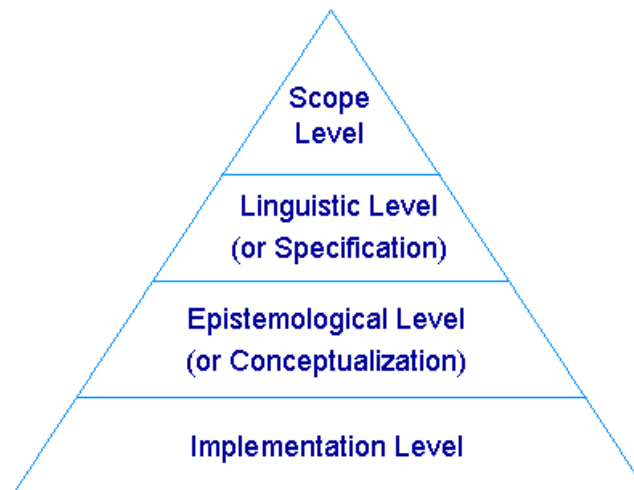

Figure 24. Levels of Abstraction for the Ontology Reengineering Activity

➢ The scope level is the highest level of abstraction, in which the general scope for an ontology is described.

➢ In the specification or linguistic level, the requirement characteristics for an ontology are described in detail.

➢ In the conceptualization or epistemological level, ontology characteristics such as the ontology structure and the ontology components are described. The knowledge that the ontology represents is organized following a set of knowledge representation primitives (concepts, relations, etc.).

➢ The implementation level is the lowest abstraction level. Here, an ontology description focuses on implementation characteristics and is represented in an ontology language understandable by computers and usable by automatic reasoners.

Software developers and ontology practitioners should decide at which level (at the implementation level, at the conceptualization level, at the specification level, or at the scope level) they want to carry out the ontology reengineering activity. Once software developers and ontology practitoners have decided the level, they should carry out the ontology reengineering

activity, and then they should integrate the result of such activity (code, conceptualization, specification, or scope) in the corresponding activity of scenario 1 presented in Section 4.1.

Taking into account the levels shown in Figure 24, we can propose a general ontology reengineering model, shown in Figure 25. This model suggests that, normally, the ontology reengineering activity begins with an ontology code and produces another ontology code. The model also suggests different possible paths from an existing ontology to the new ontology. The choice of a concrete path depends on the ontology characteristics that have to be changed. We distinguish the following types of changes: re-implementation, re-conceptualization, re-specification, and re-think. This general ontology reengineering model together with the different types of changes and ways of carrying out the ontology reengineering activity will be explained in detail in deliverable D5.4.1.
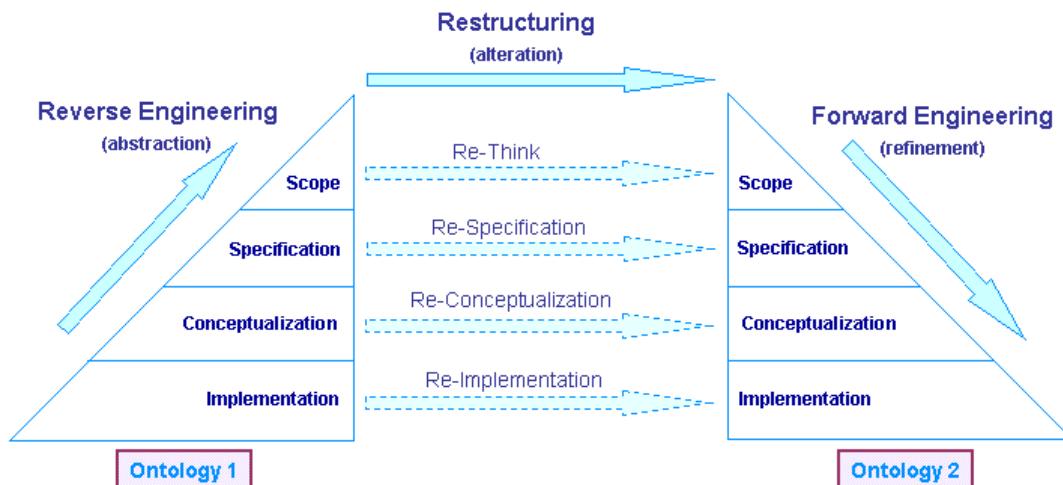

Figure 25. General Ontology Reengineering Model

❑ **Outcomes**: The principal outcome is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

Furthermore, new ontologies or ontology modules from those selected for reusing them are generated by means of the ontology reengineering activity.

4.5. Scenario 5: Building Ontology Networks by Reusing and Merging Ontology or Ontology Modules

❑ **Assumptions**: It is supposed that software developers and ontology practitioners "want" to use existing ontologies or ontology modules by merging them before their reuse in the development of the ontology network.

❑ **Prerequisite resources**: Knowledge of the domain (general or specific) of the ontology network to be developed should be available for building the ontology network.

Furthermore, ontologies or ontology modules related to the domain to be covered by the ontology network to be developed should be available.

❑ **Sequence of activities**: The sequence of activities in this scenario is the same as the sequence presented in scenario 3 (Section 4.3); however, when software developers and ontology practitioners need to decide the reuse mode (that is, activity 5 in Section 4.3), they think that the selected ontologies or modules are useful for the ontology network development,

but they decide to carry out the ontology aligning activity and the ontology merging activity in order to reuse the selected ontologies or ontology modules. This possibility is shown as scenario 5 in Figure 20.

In this scenario, software developers and ontology practitioners carry out the *ontology aligning activity* that results in a set of ontology alignments between the selected ontologies or ontology modules (which normally model the same or similar domains). Once the ontology alignments have been established, ontology developers can merge the ontologies or ontology modules using such alignments to obtain a new ontology or ontology module by means of the *ontology merging activity*.

In this case, Figure 26 shows the ontology aligning activity viewed as a process (following the 'process' definition in Section 2.1) composed of the *ontology matching activity*.

Figure 26. Ontology Aligning Activity

If software developers and ontology practitioners decide to use the merged ontology or ontology module as it is, then they should integrate the ontology or module codes in the ontology network code developed following the activities of scenario 1 (Section 4.1).

❑ **Outcomes**: The principal outcome is a network of ontologies that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

Furthermore, a merged ontology or ontology module created from those selected for reuse is generated. Alignments between the selected ontologies or ontology modules are also output of this scenario.

4.6. Scenario 6: Building Ontology Networks by Reusing, Merging and Reengineering Ontologies or Ontology Modules

❑ **Assumptions**: It is supposed that software developers and ontology practitioners "want" to use existing ontologies or ontology modules by means of merging and reengineering them before they are reused in the ontology network building.

❑ **Prerequisite resources**: Knowledge of the domain (general or specific) of the ontology network to be developed should be available for building the ontology network.

Furthermore, ontologies or ontology modules related to the domain to be covered by the ontology network to be developed should be available.

❑ **Sequence of activities**: This scenario has the same sequence of activities as scenario 5 (Section 4.5); however, in this case, software developers and ontology practitioners decide not to use the merged ontology or ontology module as it is, but to reengineer the merged ontologies or ontology modules. Once they have reengineered such merged ontologies or modules by means of the *ontology reengineering activity*, they should integrate the result of such activity (code, conceptualization, specification, or scope) in the corresponding activity of scenario 1 (Section 4.1).

❑ **Outcomes**: The principal output is an ontology network that represents the expected domain implemented in an ontology language (OWL, F-Logic, etc.). Additionally, a broad range of documents including the requirements specification, the ontology documentation, the ontology evaluation, etc. will be generated as output of different activities.

Furthermore, a merged ontology or ontology module, taken from those selected for reuse, and a reengineered merged ontology or ontology module are generated. Alignments between the selected ontologies or ontology modules are also outputs of this scenario.

4.7. Scenario 7: Building Ontology Networks by Restructuring Ontologies or Ontology Modules

❑ **Assumptions**: It is supposed that software developers and ontology practitioners "want" to restructure ontologies or ontology modules.

❑ **Prerequisite resources**: Knowledge of the domain (general or specific) of the ontology network to be developed should be available for building the ontology network.

❑ **Sequence of activities**: After the ontology conceptualization activity, software developers and ontology practitioners could carry out the *ontology restructuring activity* (scenario 7 in Figure 20). This activity can include different "sub-activities", as shown Figure 27.
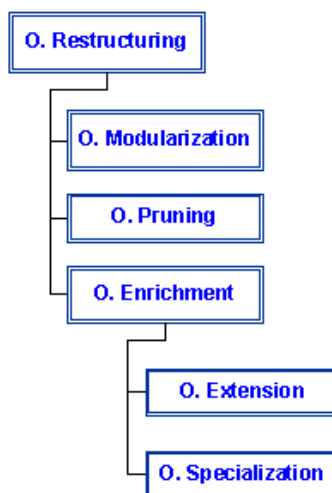

Figure 27. Ontology Restructuring Activity

The ontology restructuring activity can be performed in the following ways:

- Modularizing the ontology in different ontology modules to facilitate their reuse.
- Pruning the branches of the taxonomy not considered necessary.
- Extending the ontology including (in width) new concepts and relations.
- Specializing those branches that require more granularity and including more specialized domain concepts and relations.

After this restructuring activity, the resulting conceptual model should be integrated in the conceptualization activity in scenario 1 (Section 4.1).

❑ **Outcomes**: The principal output is a conceptual model of the ontology network that represents the expected domain.

4.8. Scenario 8: Building Ontology Networks by Localizing Ontologies or Ontology Modules

❑ **Assumptions**: It is supposed that software developers and ontology practitioners "want" to include multilinguality in an ontology or an ontology module.

❑ **Prerequisite resources**: Knowledge about the domain (general or specific) of the ontology network to be developed should be available for building the ontology network. Knowledge resources in different idioms (e.g., thesauri, dictionaries, etc.) should also be available.

❑ **Sequence of activities**: Once the ontology has been conceptualized and restructured, it can require the translation of all its terms into another natural language (Spanish, French, German, etc.) different from the language used in the conceptualization, using multilingual thesauri and electronic dictionaries (e.g. EuroWordNet). This possibility is shown in scenario 8 in Figure 20 and, in this case, ontology developers carry out the *ontology localization activity*.

After this localization activity, the resulting conceptual model should be integrated in the conceptualization activity in scenario 1 (Section 4.1).

❑ **Outcomes**: The principal outcome is a conceptual model of the ontology network in different natural languages (that is, a multilingual conceptual model) that represents the expected domain.

5. Life Cycle of Ontology Networks

Based on the definitions presented in Section 2.1, an ***ontology network life cycle model*** is defined as the framework, selected by each organization, on which to map the activities identified and defined in the NeOn Glossary of Activities, presented in Chapter 3, to produce the ontology network life cycle. On the other hand, the **ontology network life cycle** is defined as the project-specific sequence of activities created by mapping the activities identified in the NeOn Glossary of Activities onto a selected ontology network life cycle model.

Thus, the main objective of establishing the *life cycle* is to determine when the identified activities should be carried out and through which stages the ontology network moves during its life.

This chapter includes a collection of ontology network life cycle models (version 1) in Section 5.1, the classification of the activities identified (in Chapter 3) based on the IEEE groups in Section 5.2, and guidelines for obtaining the life cycle for a concrete ontology network in Section 5.3.

5.1. NeOn Life Cycle Models

Based on the Software Engineering field [46], we can say that ontology network life cycle models define how to develop an ontology network project (that is, how to develop and maintain an ontology network); in other words, they organize the activities of the NeOn Glossary of Activities into phases or stages.

In general, life cycle models can be seen as abstractions of the phases or stages through which a product passes along its life; in our case the product is an ontology network. Life cycle models are used to represent the entire life of a product from concept to disposal and they should determine the order of the phases and establish the transition criteria between different phases. Each phase or stage should be described with a statement of purpose and outcomes.

In the Software Engineering field it is acknowledged that there is no a unique life cycle model valid for all the software development projects and that each life cycle model is appropriate for a concrete project, depending on several features. For example, sometimes it is better a simple one (like waterfall), whereas other times it is most suitable a spiral one (if the analysis of the risk is needed within the project).

This claim is also valid for the Ontology Engineering field. Therefore, to propose a unique life cycle model for all the ontology network developments is not very realistic.

Taking into account the specific features of the ontology network development, we can define a collection of ontology network life cycle models (version 1) based on the life cycle models described and used in Software Engineering (Section 2.3).

In this section we have created and proposed several theoretical ontology network life cycle models, based on those proposed in the Software Engineering field. The ontology network life cycle models here presented vary from trivial and simple models to difficult and complex ones. In our proposal, we have put special emphasis in knowledge resources (ontological and non ontological) reuse and reengineering, and in ontology merging.

As a first stage we propose a collection of theoretical life cycle models for ontology network developments; and through the NeOn project, recommendations and guidelines about cases in which one model is more suitable than another (from the proposed collection) will be provided as well as information on updates on the collection based on experimentations in different use cases.

This section presents the following life cycle models for developing ontology networks: the waterfall life cycle model (Section 5.1.1), the incremental life cycle model (Section 5.1.2), the iterative life cycle model (Section 5.1.3), the evolving prototyping life cycle model (Section 5.1.4), and the spiral life cycle model (Section 5.1.5). For each model we present the requirements or outputs of each stage or phase.

5.1.1. Waterfall Ontology Network Life Cycle Model

The main characteristic of the proposed waterfall life cycle model for the ontology network development (as that of its homonymous in Section 2.3.1) is the representation of the stages of an ontology network as a sequential phases. This model represents the stages as a waterfall, from a particular stage through the following one. In this model a concrete stage must be completed before the following stage begins.

The main difference of this proposed model with respect to those presented in Section 2.3.1 is that in Software Engineering the waterfall model always includes a stage for testing the software system after its implementation, but in the case of ontology networks, the similar stage should be covered by the evaluation and the assessment activities (that should be carried out in all the phases (except in the initiation phase) and not by a different stage in the model.

As in the case of that model presented in Section 2.3.1, the main assumption for using the proposed waterfall ontology network life cycle model is that the requirements are completely known, without ambiguities and unchangeable at the beginning of the ontology network development.

Taking into account the importance of knowledge resources reuse and reengineering and ontology merging, we define and propose the following five significantly different versions of the waterfall ontology network life cycle model. All these five versions maintain the aforementioned difference with respect to the homonymous model in Software Engineering; that is, the five versions of the waterfall ontology network life cycle model include the evaluation (from technical perspective) and the assessment (from user and need perspectives) activities in all the phases of the model (except in the initiation phase) and not by a different stage in the model (as occurs in Software Engineering).

❑ *Five-phase waterfall ontology network life cycle model*.

This proposed model represents the stages of an ontology network, starting with the initiation phase and going through the requirements definition phase, the design or modelling phase, the implementation phase to the maintenance phase.

In this model a concrete phase must be completed before the following phase begins, and not backtracking is permitted except in the case of the maintenance phase.

The proposed model is shown in Figure 28, and the main requirements and outputs for each phase in the model are the following:

▪ *Initiation Phase*. In this phase the approval or rejection of the ontology network development should be obtained. This phase has as requisit to identify the development team and to establish the resources, responsibilities and timing once the development has been approved.

▪ *Requirements Definition Phase*. In this phase it is necessary to produce a requirement specification document, including the requisites that the ontology network should satisfy

and taking into account knowledge about the concrete domain. Such specification should be evaluated (from technical and user perspectives).

- *Design/Modelling Phase*. The output of this phase should be an informal model and a formal one that satisfy the requirements obtained in the previous phase. The formal model can not be used by computers, but it can be reused in other ontology networks. Both models should be documented, evaluated and assessed.

- *Implementation Phase*. In this phase, the formal model obtained in the previous one is implemented in an ontology language. As in previous phases, the result of this implementation should be documented, evaluated and assessed. The output of this phase can be used by semantic applications or by other ontology networks.

- *Maintenance Phase*. If, during the use of the ontology network, errors or missing knowledge are detected, then the development team should apply again the model starting from the requirements definition phase. Additionally, in this phase documentation, evaluation, and assessment should be also carried out as well as the generation of new versions for the ontology network.
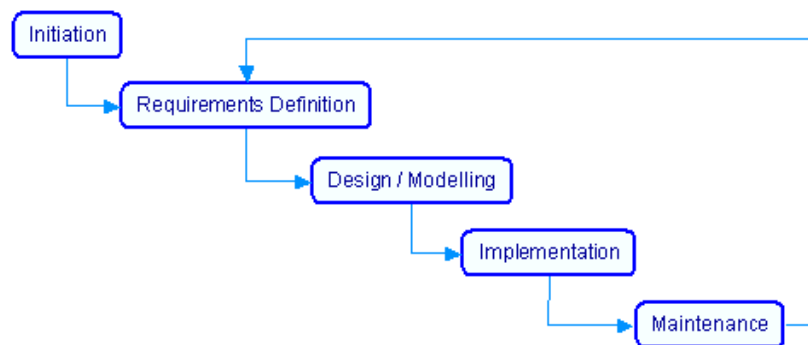

Figure 28. Five-phase waterfall ontology network life cycle model

Project management and configuration management issues should be carried out in all the phases of the model. The same applies to knowledge acquisition, documentation, evaluation and assessment.

- ❑ ***Six-phase waterfall ontology network life cycle model***.

This model, shown in Figure 29, extends the previous one with a new phase in which the reuse of already implemented ontologies or ontology modules is considered. The main requirement in the *Reuse Phase* is the obtention of one or more ontologies or ontology modules to be reused in the ontology network being developed. The output of this reuse phase could be an informal model or a formal one to be used in the 'design/modelling' phase; or an implemented model (in an ontology language) to be used in the 'implementation' phase.

For the other phases the requirements are the same as those presented in the previous model.
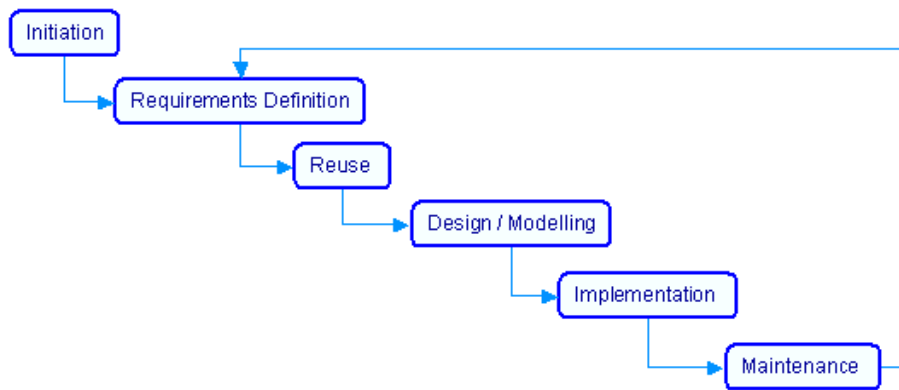
Figure 29. Six-phase waterfall ontology network life cycle model

❑ *Six-phase + merging phase waterfall ontology network life cycle model*.

Figure 30 shows the proposed model, which is a special case of the previous one. In the model, a new phase (the merging one) is added after the reuse one. This *Merging Phase* has as a requirement to obtain a new ontology from two or more ontologies selected in the reuse phase.

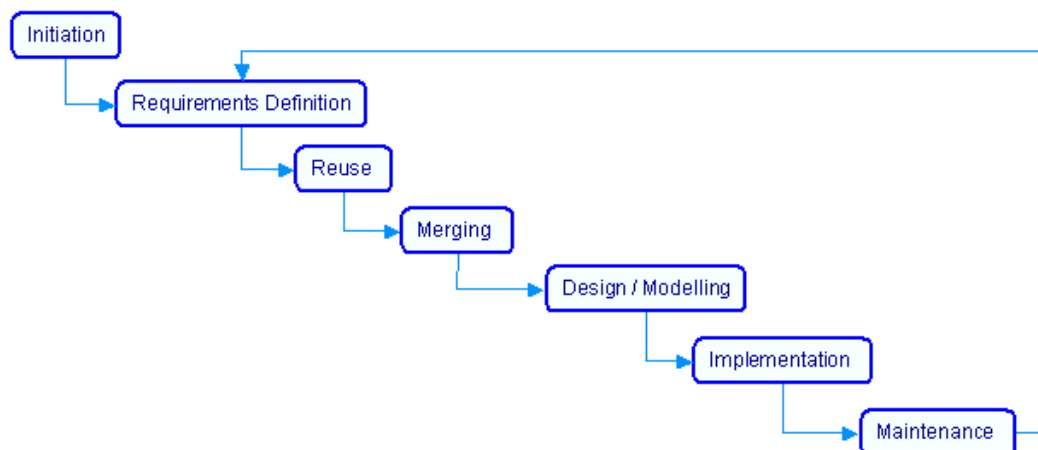For the other phases the requirements are the same as those presented in the previous model.

Figure 30. Six-phase + merging phase waterfall ontology network life cycle model

❑ *Seven-phase waterfall ontology network life cycle model*.

In this model, shown in Figure 31, the six-phase model is taken as general basis and a new phase (*Reengineering Phase*) is included after the reuse one. This model allows reusing knowledge resources (ontological and non ontological) and their later reengineering. In this model the reuse phase has as output one or more knowledge resources to be reused in the ontology network that is being developed. After this phase, the non ontological resources are reengineered in the reengineering phase; on the other hand, the ontological resources can or cannot be reengineered (this is decided by the development team).

For the other phases the requirements are the same as those presented in the previous model.
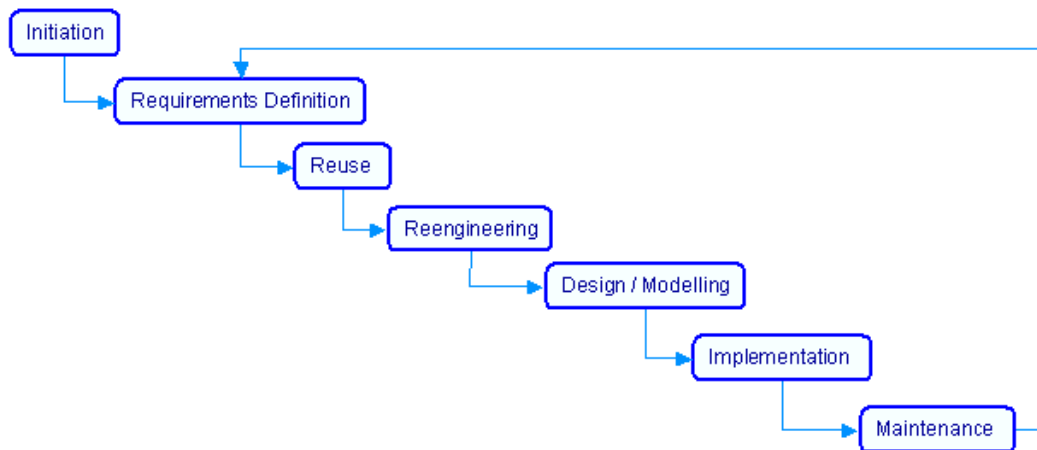
Figure 31. Seven-phase waterfall ontology network life cycle model

❑ *Seven-phase + merging phase waterfall ontology network life cycle model*.

This model, extended from the previous one and shown in Figure 32, includes the *Merging Phase* after the reuse if the development team is reusing ontologies or ontology modules. Additionally, this model allows going into the merging phase for merging the reengineered resources after reengineering knowledge resources (ontological and not ontological).

For the other phases the requirements are the same as those presented in the previous model.
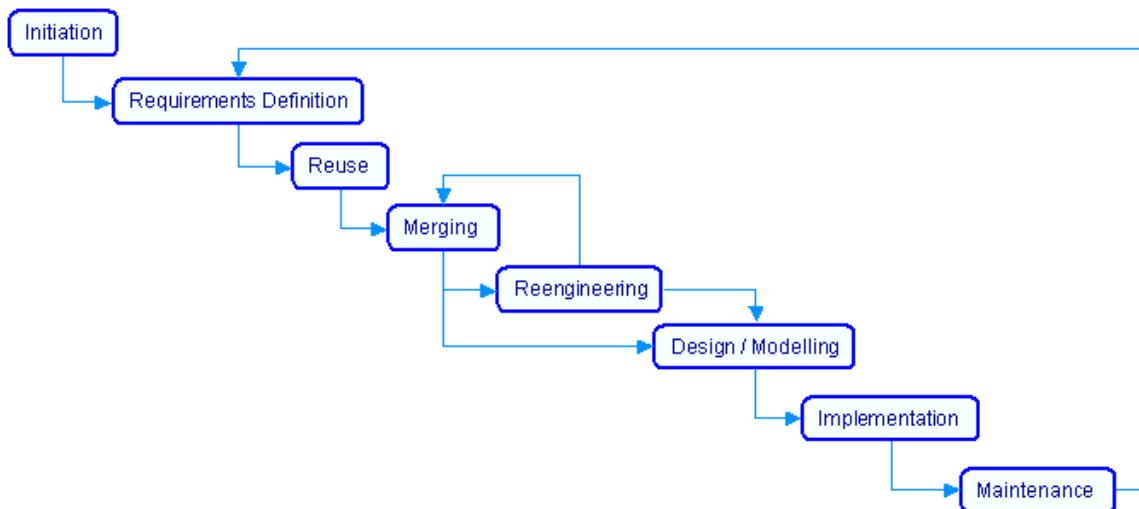

Figure 32. Seven-phase + merging phase waterfall ontology network life cycle model

5.1.2. Incremental Ontology Network Life Cycle Model

The main feature of this proposed model (as that of its homonymous in Section 2.3.3) is to divide the requirements in different parts and then develop each part in a different cycle. The idea is to "produce and deliver" the network of ontologies in increments (full developed and functional).

This model assumes that the overall requirements are well-known at the begin of the development. In incremental models, however, a limited set of requirements is allocated to each increment and with each successive release more requirements are addressed until the final release satisfies all requirements.

The main difference, and thus contribution, of this model is that in ontology engineering we do not deal with "subsystems" as in Software Engineering, but with ontologies or ontology modules (work on ontology modules is being carried out in WP1). That is, this model proposes to divide the ontology network requirements in different parts and to develop each part in a different cycle as an ontology (if we are developing an ontology network) or as an ontology module (if we are developing a networked ontology embedded in a network of ontologies). A mixed approach can be also applied. Each set of requirements is completely developed and maintained following any type of waterfall ontology network life cycle model of those presented in Section 5.1.1. So, as a summary we can say that the ontology network grows in layers (in a concentric way).

5.1.3. Iterative Ontology Network Life Cycle Model

The main characteristic of this model (as that of its homonymous in Section 2.3.4) is to divide all the requirements into small parts and to develop the ontology network including partially requirements from all the parts.

As in the model presented in Section 5.1.2, the requirements are divided into parts, but in this model ontologies and ontology modules satisfying each requirement part are developed in a non complete manner in each cycle; the idea here is to improve increasingly the ontologies or ontology modules in the next cycles. In each iteration different uncompleted parts of the network are improved, until all the requirements are included. For example, in the first iteration we deal with concepts and then in the next iteration we can improve the ontology with properties, etc.

Each cycle follows any type of waterfall ontology network life cycle model of those presented in Section 5.1.1.

5.1.4. Evolving Prototyping Ontology Network Life Cycle Model

The main feature of this model (as that of its homonymous described in Section 2.3.5) is the development of a partial product (in this case, partial ontology network) which meets the requirements best understood. But in the case of ontology network, this model includes the evaluation and assessment carried out during the development. The preliminary versions of the ontology network being developed (that is, the prototypes) permit the user to give feedback of unknown or unclear requirements.

5.1.5. Spiral Ontology Network Life Cycle Model

The main feature of the spiral ontology network life cycle model is that it follows the same philosophy as its homonymous model (Section 2.3.7) and proposes a set of repetitive cycles, based on waterfall and prototype models.

In Software Engineering the homonymous model divides the space into four quadrants: management planning, formal risk analysis, engineering, and customer assessment. However, in this model, taking into account the special characteristics of ontology networks, the space is divided into three section, as Figure 33 shows. This differentiation is based on the need to evaluate and assess all the outputs of all the ontology network stages, but not after the engineering phase as it happens in software projects.

In this model it is assumed that the requirements are not completely known at the beginning of the development, and that they can change along the life of the ontology network.

This life cycle model consists of the following three repetitive phases:

❑ *Planning*: in this phase it is carried out the whole schedule for the ontology network development and the specification of the ontology network requirements.

- ❑ Risk analysis: after analysing the possible risk within the ontology network development, the decision of continuing or not with a new iteration around the spiral is taken.

- ❑ Engineering: in this phase it is developed a prototype of the ontology network based on the specified requirements, following any type of waterfall ontology network life cycle model of those presented in Section 5.1.1.
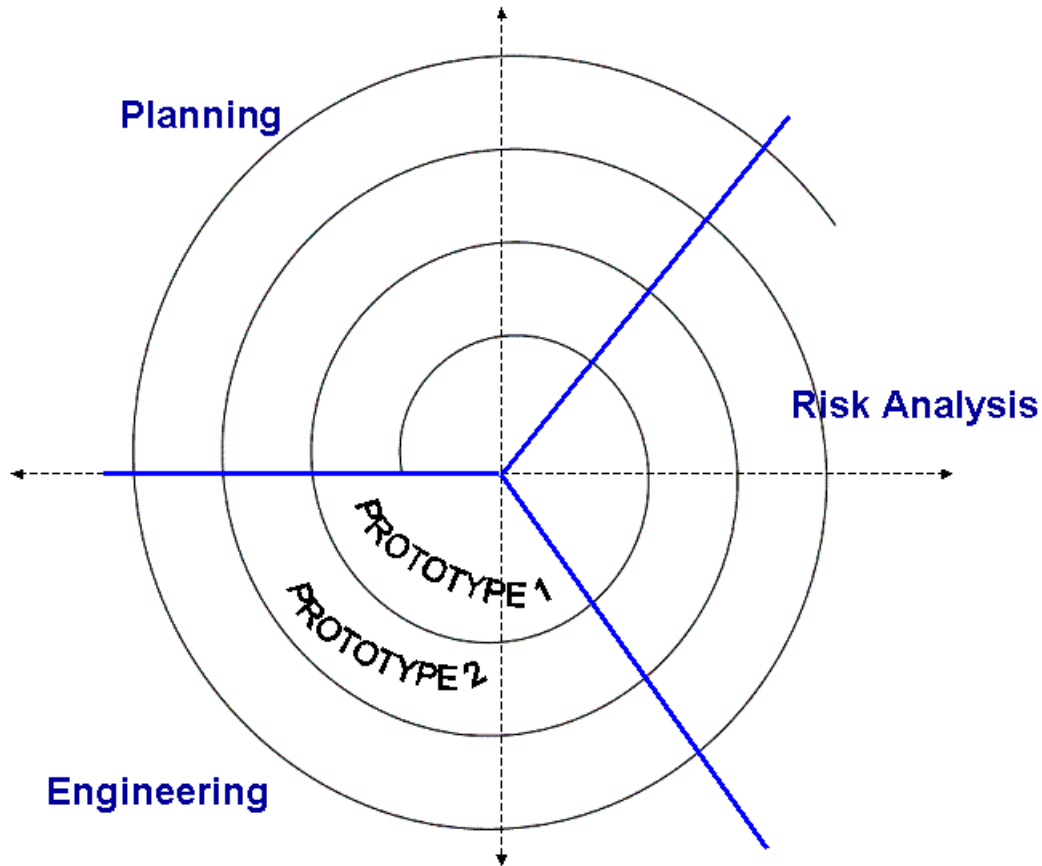
Figure 33. Spiral ontology network life cycle model

5.2. NeOn Activity Classification based on IEEE groups

Based on IEEE activity groups [4], we have grouped the 53 activities identified in the NeOn Glossary of Activities (Chapter 3) into five main activity groups. The full set of activities covers the entire ontology network life cycle. The groups provide an administrative classification to present the activities in a more coherent way.

The main activity groups, as presented in Figure 34, are the following:

1. **Management activities** include the following three activities: scheduling, control and ontology quality assurance.

2. **Development-oriented activities** are grouped into pre-development, development and post-development activities.

   The *pre-development activities* include the following activities: ontology environment study, ontology feasibility study, ontology reuse, ontology reengineering, non ontological resource reuse, and non ontological resource reengineering.

   The *development activities* include the following activities: ontology specification, ontology conceptualization, ontology formalization, ontology implementation, ontology integration,

ontology modularization, ontology restructuring, ontology merging, ontology aligning, ontology update, ontology modification, ontology localization, ontology translation, ontology annotation, and ontology customization.

The *post-development activities* include the following activities: ontology upgrade, ontology versioning, and ontology evolution.

3. **Support activities** include the following activities: knowledge acquisition, ontology evaluation, ontology documentation, ontology summarization, ontology assessment, and ontology configuration management.
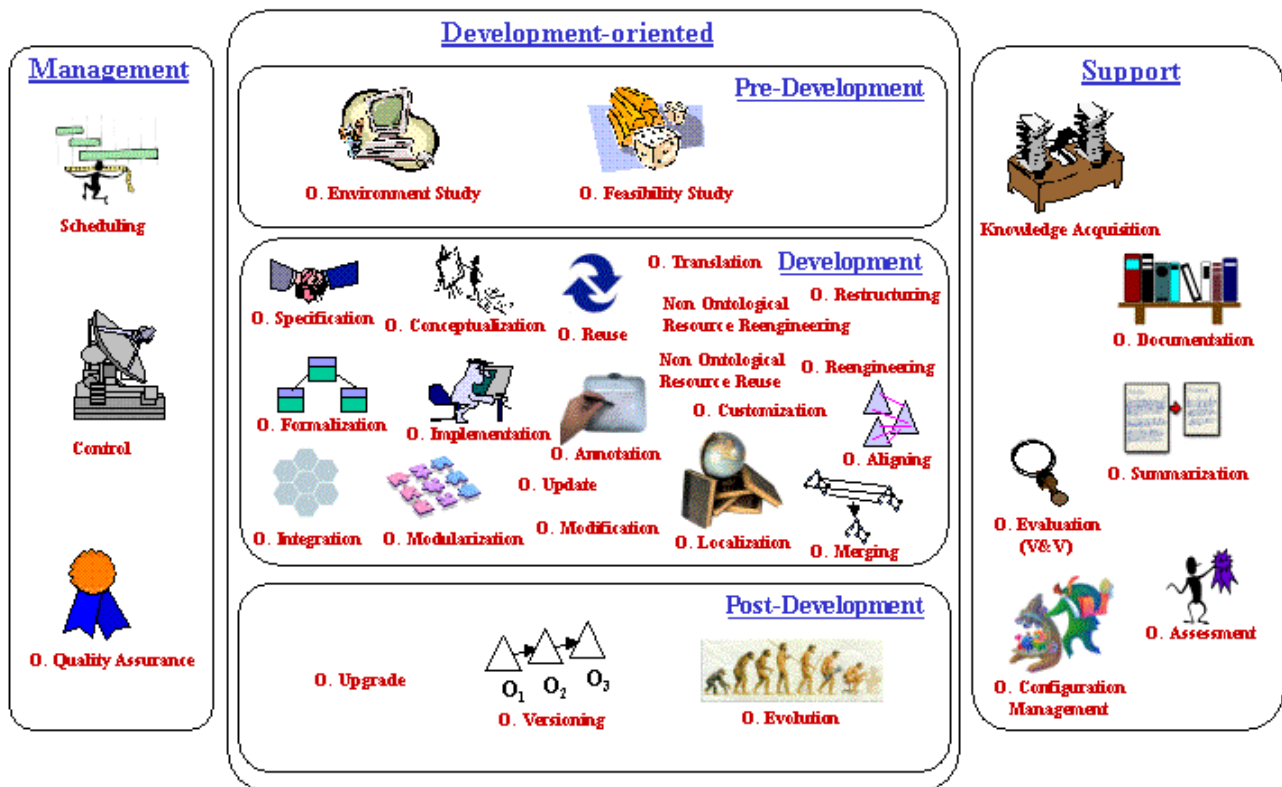

Figure 34. NeOn Activity Classification

The previous activity classification provides an implicit preliminary order for the activities, as Figure 35 shows. Figure 35 illustrates the connections among the different activities. To sum up, the first activity to be carried out is an environment and feasibility study to decide whether the ontology network should be developed or not. If software developers and ontology practitioners decide that the ontology network should be developed, then they should carry out the ontology scheduling activity to obtain the concrete plan for the ontology network development. After this activity, activities from the management, development-oriented, and support groups should be carried out following the ontology network life cycle established in the ontology scheduling activity. Section 5.3 explains in detail the different steps to be carried out within the ontology scheduling activity, which mainly includes the ontology network life cycle establishment and the setting of temporal and resource information.
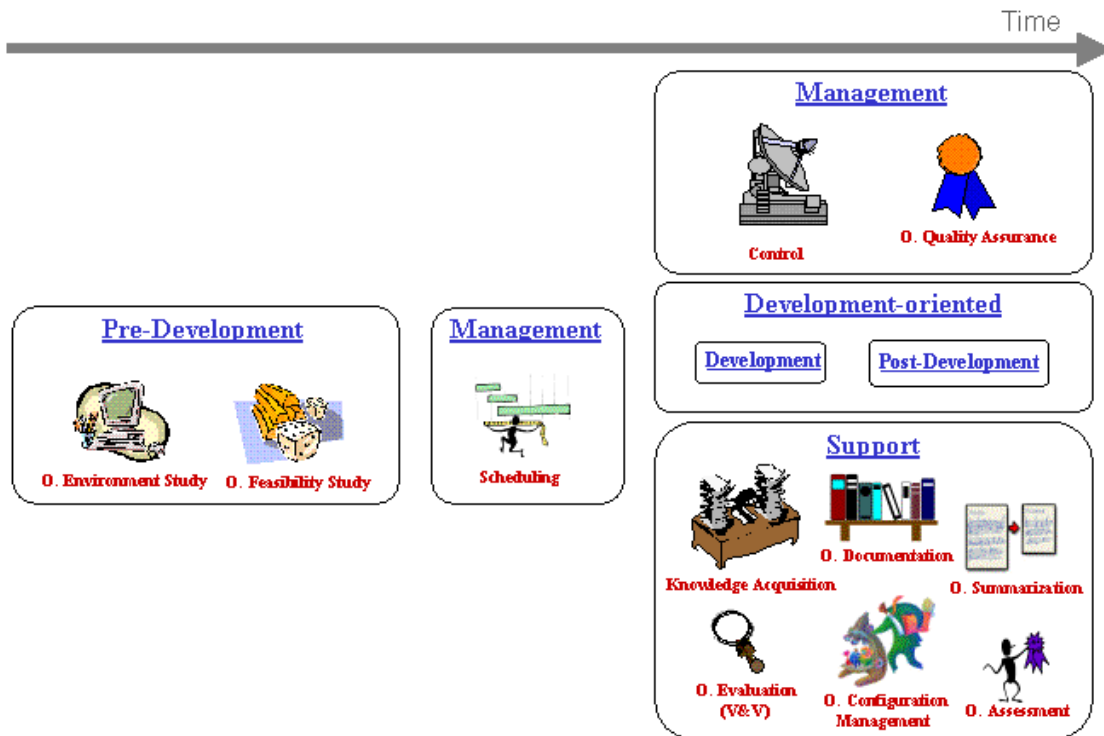
Figure 35. Temporal Order of Activities

5.3. Obtaining the Ontology Network Life Cycle

The main objective of the *ontology network life cycle* is to determine when the activities identified should be carried out and through which stages the ontology network moves during its life.

The **ontology network life cycle** is defined as the specific ordered sequence of activities that software developers and ontology practitioners carry out during the life of the ontology network.

The question now is **how software developers and ontology practitioners decide which ontology network life cycle model** (taken from the collection presented in Section 5.1.) **is the most appropriate for their ontology network and which concrete activities** (taken from those identified in Chapter 3) **should be carried out in their ontology network life cycle**.

Once software developers and ontology practitioners have taken these decisions, they obtain (by mapping the selected ontology network life cycle model and the selected activities, and then placing in order such activities) the particular life cycle for their ontology network. That is, the ordered sequence of activities to be carried our during the life of the ontology network.

To help software developers and ontology practitioners in the ontology network life cycle establishment we propose the following main steps:

1. **Identify ontology network development requirements**.

   In this step, software developers and ontology practitioners informally identify the main needs and expectations of the ontology network development.

2. **Select the ontology network life cycle model (ONLCM) to be used**.

   The main question is: "which ontology network life cycle model should be chosen?". Such choice should be based on: software organization culture, previous experience of software developers and ontology practitioners team, application area, and comprehension and volatility of the ontology network requirements. Furthermore, general ontology network development

requirements should be taken into account. Some guidelines to help with this selection are provided in this step.

Software developers and ontology practitioners should select the ONLCM. Such selection consists of locating, evaluating, and selecting an ONLCM for the ontology network project. The following proposed substeps should be carried out for selecting the ONLCM:

a) *Identify ontology network development characteristics and constraints*. The ONLCM selection should be based upon ontology network development characteristics and constraints, such as: if prototyping is required during the development, if knowledge resource reuse is recommended, if the risk impact should be evaluated, etc.

b) *Identify all the possible ONLCMs.* From the collection of ontology network life cycle models presented in Section 5.1, software developers and ontology practitioners should identify possible models to be applied, depending on ontology network development characteristics and constraints.

   The following assumptions can help software developers and ontology practitioners to identify possible ONLCMs for a particular ontology network.

   ➢ When ontology network requirements are assumed to be known at the beginning of the ontology network development, the following ontology life cycle models can be proposed: waterfall model, incremental model or iterative model.

   ➢ When ontology network requirements could be not known at the beginning of the ontology network development and could change during the development, the following ontology network life cycle models can be proposed: evolutionary prototyping or spiral model.

   ➢ When uncertainties in the ontology network requirements can derive into risks in the development, which could be analysed, then the spiral model is proposed.

   Based on the aforementioned assumptions, the informal decision tree shown in Figure 36 is proposed to help software developers and ontology practitioners to select which ontology network life cycle model is the most appropriate for their ontology network.
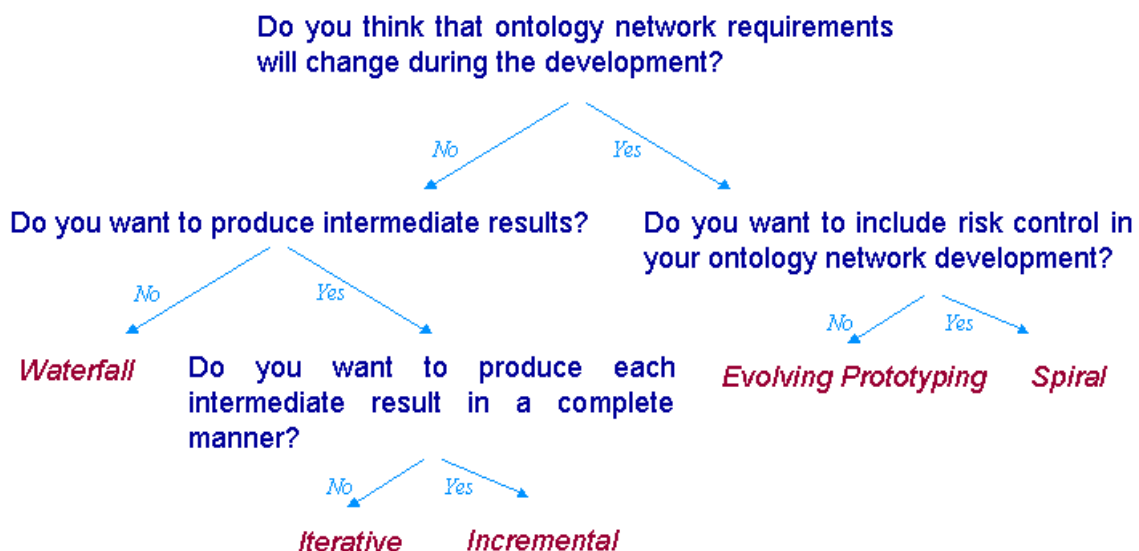

Figure 36. Decision Tree for Selecting the Ontology Network Life Cycle Model

c) *Evaluate the various ONLCMs based on past experience and organizational capabilities.* Each proposed ONLCM should be evaluated using the past records (e.g., reports of problems encountered in meeting schedules) and lessons learned.

d) *Select the ONLCM that best satisfies the ontology network development characteristics and constraints.* Based on the previous substeps, software developers and ontology practitioners should select an ONLCM for the ontology network.

If none of the evaluated ONLCM satisfies the ontology network development characteristics and constraints, it is also possible for the software developers and ontology practitioners to define a new ontology network life cycle model for the ontology network, based on those presented in Section 5.1 and adapted for the concrete development and/or organization.

3. **Select activities to be carried out from the "Required–if Applicable" table**.

Required activities, identified in the "Required-if Applicable" table (Section 3.3), plus all others applicable to the ontology network development should be selected to be carried out during the ontology network life cycle. The result of this step is the table of selected activities.

For this step, we propose to differentiate between two different kind of software developers and ontology practitioners:

❑ *Software developers and ontology practitioners that have developed more than 5 ontologies.* In this case, we assume that software developers and ontology practitioners, based on their experience, are able to select the activities to be carried out during the ontology network life cycle from the "Required-if Applicable" table.

Activities identified as "required" in the "Required-if Applicable" table are selected automatically in the table of selected activities. Software developers and ontology practitioners should only select those "if applicable" activities needed for their ontology network development.

Table 4 shows a short example of possible selected activities for a concrete project.

Table 4. Example of Table of Selected Activities

	Required	*If Applicable*	*Selected*
Ontology Annotation	X		*X*
Ontology Assessment	X		*X*
Ontology Conceptualization	X		*X*
Ontology Customization		X	
Ontology Forward Engineering		X	*X*
Ontology Reengineering		X	*X*
Ontology Restructuring		X	*X*
Ontology Reverse Engineering		X	*X*

❑ *Software developers and ontology practitioners that have developed less than 5 ontologies.* In this case we propose a list of "yes/no" natural language questions (version 1) for those "if applicable" activities (identified in Section 3.3) to be answered by software developers and ontology practitioners. Based on the responses obtained, "if applicable" activities will be then automatically selected or not.

As in the previous case, activities identified as "required" in the "Required-if Applicable" table are selected automatically in the table of selected activities. Taking into account

software developers and ontology practitioners responses (to the proposed questions), "if applicable" activities are selected in the table of selected activities. If a response of a concrete question is positive, then the corresponding activity is selected; and on the contrary the activity is not selected.

The possibility that the users can not response yes or not to the proposed natural language questions will be analysed within WP5 and such work will be included in the next version of this deliverable (that is, D5.3.2).

Table 5 presents the list of proposed questions.

Table 5. Proposed "Yes/No" Natural Language Questions

Activity	Natural Language Questions
Ontology Aligning **(or Ontology Mapping)**	Do you have two or more ontologies at your disposal that you want to examine to find correspondences and to take advantage of them? Do you want to find out correspondences among ontologies to use them?
Ontology Customization	Do you want to adapt the ontology network to a specific user profile? Do you want to modify the ontology network to meet specific user needs?
Ontology Enrichment	Do you want to widen/extend your current ontology network with additional elements (e.g., concepts, roles, axioms, etc.)?
Ontology Extension	Do you want to stretch, widen, broaden or expand your current ontology network by adding new concepts "in a horizontal way/direction" with the aim of widenning its sphere of action? (cf. Ontology Specialization)
Ontology Forward Engineering	Are you going to carry out a new implementation for a previously modified conceptual model? Are you going to produce a new implementation for a modified conceptual model, whose previous version had already been implemented?
Ontology Learning	Do you want to transform (semi)-automatically the knowledge contained in unstructured, semi-structured or structured data sources into an ontology network? Do you want to use in a (semi)-automatic way the knowledge contained in unstructured, semi-structured or structured data sources to build your ontology network?

Activity	*Natural Language Questions*
Ontology Localization	Do you want to have your ontology network in different natural languages, as for example, in English, Spanish and French?
Ontology Matching	Do you want to find correspondences between entities of different ontologies or ontology modules?
Ontology Merging	Are you going to reuse existent ontologies or ontology modules (let's say O1 and O2) to create a new one (O3) as a union/fusion of the initial ones (O3 = O1 + O2)? Are you going to build an new ontology network or a part of it as a fusion of existent ontologies?
Ontology Modification	Are you going to change or modify your ontology network?
Ontology Modularization	Do you want to identify modules in your ontology network to make the reuse or maintenance of it easier?
Ontology Module Extraction	Do you need to take out some modules of your ontology network to use them for other purposes?
Ontology Partitioning	Do you want to divide your ontology network in modules so that they can be treated separately?
Ontology Population	Do you want to transform (semi)-automatically the knowledge contained in unstructured, semi-structured or structured data sources into instances or individuals of your ontology network?
Ontology Pruning	Do you want to take out conceptual structures that are no longer relevant for your ontology network?
Non Ontological Resource Reengineering	Do you have the intention of taking an existing non ontological resource (e.g., thesaurus, data base, etc.), enhancing it and implementing it as an ontology?
Ontology Reengineering	Do you want to take an existing and implemented ontology to enhance it and implement it again?
Ontology Restructuring	Do you want to check the knowledge contained in your ontology and the organization of it, correct any kind of incorrectness or inconsistencies, and document the changes made?
Non Ontological Resource Reuse	Do you have the intention of using non ontological resources (as a controlled vocabularies or data bases) in the development of your ontology?
Ontology Reverse Engineering	Do you want to transform the code in which an ontology is implemented into its corresponding conceptual model? Do you want to output the conceptual model starting from the implementation code of an ontology?
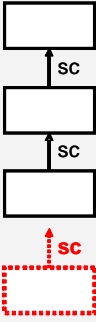
Activity	*Natural Language Questions*
Ontology Specialization	Do you want to extend your current ontology network by adding new concepts "in a vertical way/direction" with the aim of increasing the specialization of it? (cf. Ontology Extension)
Ontology Summarization	Do you want to provide an abstract of your ontology network in which you may include an extract of the highest level of the ontology hierarchy, or a summary of the Ontology Specification, or maybe a set of metadata?
Ontology Translation	Do you want to change the representation formalism (description logic, frames, rules, etc.) or language (RDF(S), OWL, etc.) of your ontology network?
Ontology Update	Are you going to carry out minor changes in your ontology network that do not imply putting out a new version of it?

Figure 37 shows the informal decision tree for obtaining the table of selected activities.

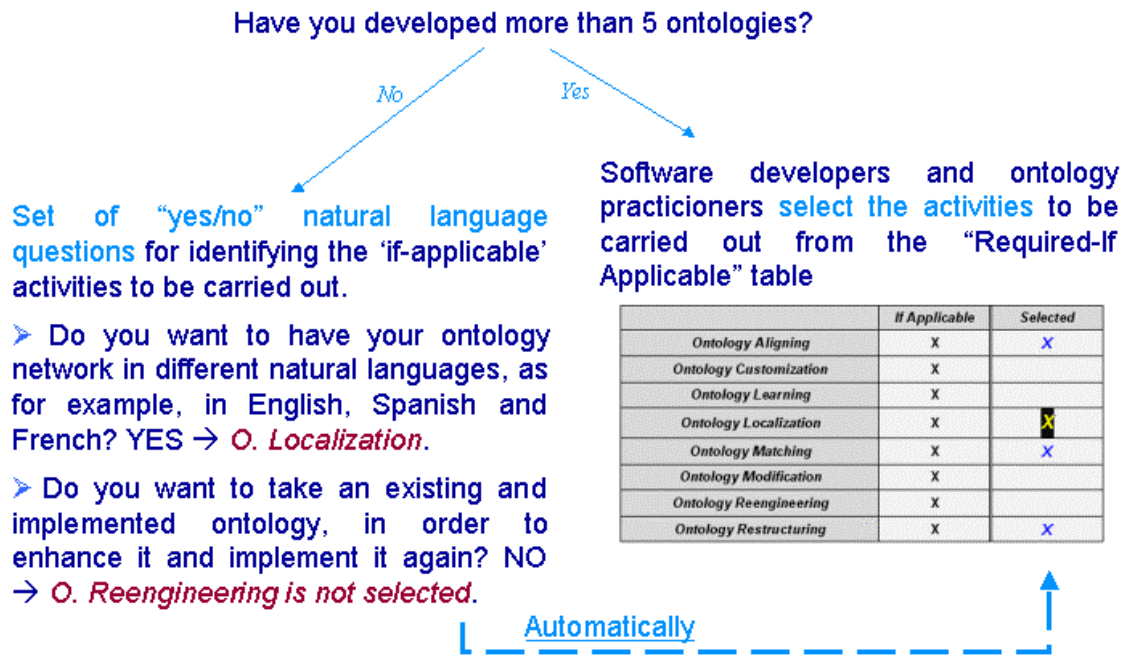**Have you developed more than 5 ontologies?**

No   *Yes*

Set of "yes/no" natural language questions for identifying the 'if-applicable' activities to be carried out.

➤ Do you want to have your ontology network in different natural languages, as for example, in English, Spanish and French? YES → *O. Localization*.

➤ Do you want to take an existing and implemented ontology, in order to enhance it and implement it again? NO → *O. Reengineering is not selected*.

Software developers and ontology practicioners select the activities to be carried out from the "Required-If Applicable" table

	If Applicable	Selected
Ontology Aligning	X	x
Ontology Customization	X	
Ontology Learning	X	
Ontology Localization	X	X
Ontology Matching	X	x
Ontology Modification	X	
Ontology Reengineering	X	
Ontology Restructuring	X	x

Automatically

Figure 37. Decision Tree for Selecting Activities

4. **Map the selected activities into the selected ontology network life cycle model**.

   For carrying out this mapping, software developers and ontology practitioners should match the selected activity outputs against the requirements of the selected ONLCM.

   Selected activities are employed in a phase or stage of the selected ONLCM to fulfil the purpose and outcomes of that phase.

   This step provides an activity map or matrix for the ontology network development. More details on this step will be included in the next version of this deliverable (D5.3.2).

   Once the activity matrix has been obtained, software developers and ontology practitioners should verify that all the selected activities are fully mapped onto the selected ONLCM, and that such matrix contains all of the activities needed for completing successfully their ontology network project.

5. **Set the order of the activities: the result is the ontology network life cycle for the ontology network**.

   After obtaining the activity map or matrix, software developers and ontology practitioners should place in order the activities of this matrix, obtaining in this way the ontology network life cycle.

   The order in which activities will be performed are determined by three major factors:

   a) The selected ONLCM will dictate an initial ordering of activities.

   b) Schedule constraints may require the overlapping of activities in the ONLCM and may thus impact the ordering. In this case, activities may be mapped for parallel execution rather than for serial execution.

   c) Selection and ordering of activities might be impacted by the entry and exit criteria of associated activities. The availability of output information from one activity could affect the start of another activity. The second activity might require, as inputs, one or more of the outputs of the first.

After obtaining the ontology network life cycle, software developers and ontology practitioners should include information about temporal scheduling and resource assignments, as part of the ontology scheduling activity. The output of such activity can be represented as a Gantt diagram (to be explained in D5.3.2).

6. NeOn Use Cases Life Cycles

This chapter includes the ontology network life cycle for the NeOn use cases (fisheries in WP7, and invoicing and nomenclature in WP8) is included.

Each use case is described in a specific section with the following structure:

> ➢ Use case description, including a general overview of the use case.

> ➢ Selected ontology network life cycle model (using the decision tree presented in step 2 within Section 5.3).

> ➢ Selected activities to be carried out (using the decision tree presented in step 3 within Section 5.3).

> ➢ Activities carried out for building the ontologies / ontology networks.

> ➢ Lessons learned by following the methodological guidelines including in this deliverable and other guidelines being carried out in WP5.

In the FAO use case (Section 6.1) the section structure is not exactly the same as the aforementioned because the activities carried out within T7.2 (*Mapping of resources and initial fishery ontology repository*) and described in detail in [60] were carried out in parallel with the activities related to T5.3 (*Identification of the networked development process and ontology lifecycle*) in WP5. In fact, no specific methodological guidelines from WP5 were available when the development of the first set of fisheries ontologies included in [60] began. The ontologies at hand have been developed according to a number of guidelines and "best practices" commonly known in the literature (Ontology 101, available documentation for OWL and for Protege-OWL). On the other hand, the FAO team dealt with relatively novel tasks, i.e. development of ontologies populated from database, which involves a novel tool (ODEMapster); unfortunately, there are no guidelines or methodologies currently available for this tool that FAO team know of.

iSOCO use case (Section 6.2) and ATOS use case (Section 6.3) are described following the aforementioned structure. Both partners have taken into account the methodological work presented in this deliverable and other methodological work being carried out within WP5 for developing their ontology networks.

6.1. FAO Ontology Network Life Cycle

6.1.1. Use case description

The aim of the WP7 case study is to create an ontology-driven stock depletion assessment system (FSDAS) (for a general overview of the WP7 use case, refer to [37]). Such a system will use data sets already available at FAO, and these data sets will be accessed by means of (a network of) ontologies. One candidate of the data sets for inclusion is the time series about fisheries. Central to all fisheries statistics are the biological species that are caught, produced and exchanged in the market. Biological species constitute a metadata (commonly called reference data, which is stored in reference tables) that is used to reference (or annotate) the statistical data. Reference tables are stored in relational tables in a RDBMS.

6.1.2. Activities carried out for building the ontologies

Since the data at hand is stored in relation form, the process of converting it into ontologies is, at the same time, a problem of domain modeling and of data reengineering.

The inventory of candidates of data sources to be included in the FSDAS was carried out in task T7.2 within the WP7, and reported in the deliverable D7.2.1. Such inventory was the starting point to select the data sets.

The objective is to create ontologies related with biological species (data came from the Reference Tables (RT) and managed by the Reference Tables Management system (RTMS); and that is used to access statistical data, that is timeseries).

Timeseries data is fundamental to any research about fishery; therefore, this type of data constitutes a core part of the FSDAS. All data stored in the RTMS and used to access timeseries must be included in the corresponding ontologies; on the other hand, all timeseries must continue to be available to query (cf. [37] for a broader view of the user requirements concerning the ontology life cycle and FSDAS). In this scenario we concentrate on the part of the RTMS data that concerns the biological species.

In this phase of the project the FAO team only deal with single ontologies (as opposed to networked ontologies); therefore, this scenario only covers the creation of single ontologies.

The FAO team have proceeded in the following way:

1. **Study of the domain.** First of all, the FAO team studied the material available on biological species that was provided by the biologists in charge of the maintenance of the list of species of global interest. Then, the FAO team went over the relevant factsheets of the CWP Handbook of Fishery Statistical Standards collected by the Coordinating Working Party on Fishery Statistics (CWP); the FAO team also studied data gathered by the ASFIS list (Aquatic Science and Fisheries Information System) and by the associated classification systems.

2. **Interviews with the domain experts.** After having obtained a general overview of the domain, the FAO team interviewed the biologists; the aim of such interviews was to understand the nature and the dynamics of the ASFIS list, that is, which is the connection with the statistical data provided to FAO by its member countries, how often member countries need to make a change in the list, where such a need comes from, who performs the changes and how. These interviews were intended to shed some light on the life cycle of the reference data in the context of the actual statistical.

3. **Interviews with the information system experts.** These people are in charge of the maintenace of the RTMS system. They periodically update the RTMS by converting the list format used by the biologists into the relational database, which is more suitable for accessing the reference data and which it is used to access the timeseries. The FAO team carried out these interviews with the aim of understanding the life cycle of the reference data in the context of real applications and actual use.

4. **Model designed.** The first two models for the ontologies based on biological species were created after an in-depth study of the data available from the RTMS (step 1). The FAO team used Protégé 3.1.1 to create and edit these ontologies, applying the FAO team's knowledge of common methodologies and best practices for creating ontology (e.g. Ontology 101). The FAO team wrote the corresponding documentation in which some pros and cons were analyzed. Then it was decided that the actual decision of selecting one model over the other was to be made on the basis of: a) good modeling; b) efficiency in actual use of the ontology; and c) efficiency in getting the data to populate the ontology. The first conceptualization activity was made before the actual implementation of the ontology; afterwards, an iterative process (further study of the data and conceptualization) followed. It must be added that the ontology was implemented in English (i.e., class and property names).

5. **Study the DB to populate the models.** This phase was carried out by studying the available documentation that describes the structure of the database of the RTMS (from now on DB) and then making a number of interviews and conversations with the information experts working with the DB and adding finally the knowledge previously gained about the domain. The data about biological species is very hierarchical in that species are grouped into Families, which

are organized into Orders, which are grouped into Main Groups. To store this data in a relational database, the DB designers organized the information into three sorts: a) hierarchy of types (groups - order - family - species); b) actual instances (i.e. all individual species, families, orders, and groups are represented as individual items); c) actual hierarchies (i.e. each individual item is linked to its "sub-item" along the line). These three sorts of information are stored in three tables linked to one another by secondary keys.

By looking at the actual data in the DB the FAO team discovered that the actual hierarchy (as stored in the table) did not include Genus (that is, no timeseries is available for that), therefore the ontology model was modified accordingly (this is an example of the iterative process mentioned above).

6. **Population of the models.** The FAO team used ODEMapster to automatically populate the ontology. The FAO team chose one model (the latest resulting from the iterative process described above), and also modified slightly the model during the work, considering the possibilities of the tool at that time. For example, some properties that were first thought to be modeled as object properties were turned into datatype properties because it was easier to populate the ontology with the version of the tool available at the time. This happened, for example, with names of species in various languages; however, some of the modeling decisions made on these considerations were later deemed appropriate, independently of the tool.

The life cycle model used for building FAO ontologies follows an iterative approach, and it is graphically represented in Figure 38 [61].
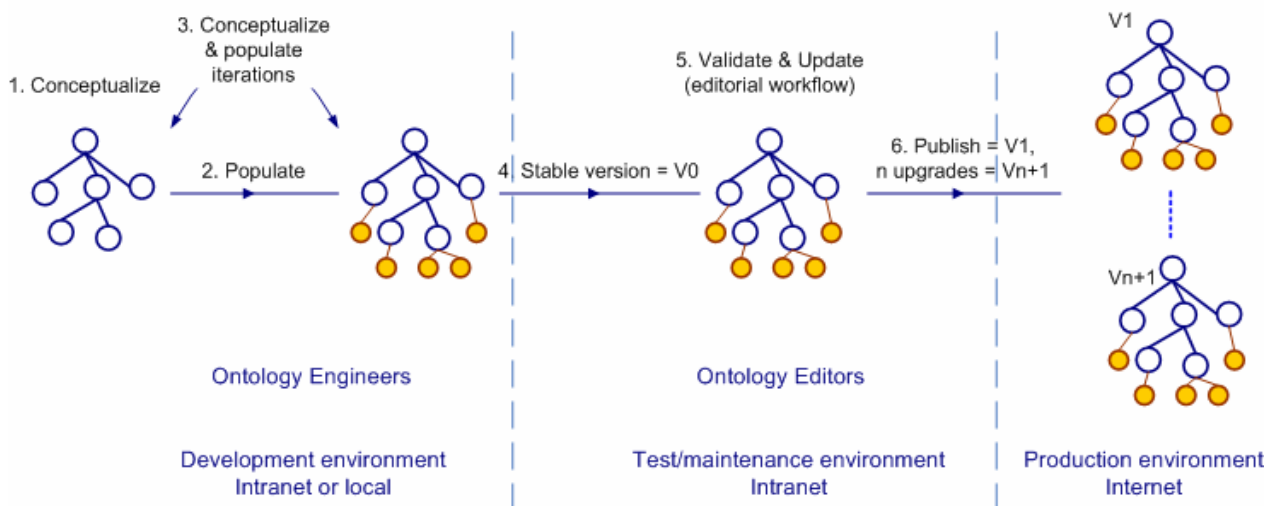

Figure 38. Life Cycle Model Selected in FAO Use Case [61]

The six activities aforementioned are placed in the first block of steps (1, 2, 3) of Figure 38. The second block (steps 4 and 5) concerns the validation and update for which there is currently no tool or dedicated environment. For this reason, it is necessary that ontology editors inspect and validate the ontologies under development; in the FAO case, the team used the html pages generated by OWLdoc (in Protégé).

The third block of Figure 38 (step 6) depicts the production environment. Since the FAO team did not have an integrated tool that would allow them to move from creation to production through validation, the FAO team itself moved the ontologies to the production environment.

In short, the process that lead to the WP7 fishery ontologies consisted in an iteration of: a) conceptualization based on well-known methodologies; and b) population from the database and consequent revision of the previous conceptualization.

6.2. Invoice Management Ontology Network Life Cycle

6.2.1. Use case description

The invoicing use case deals with the heterogeneity of the invoices emitted and received by pharmaceutical organizations. Pharmaceutical organizations range from laboratories that provide drugs and medicines to other companies to providers that supply any type of good that a laboratory may need. In this range of companies pharmacies, wholesalers or the public administration can be also included. Each organization uses its own invoice model and, therefore, they generate different invoices (instances of its own model). On the other hand, the technologies used by the companies that participate in the invoicing process to create the invoices are different. They can use comma separated value (CSV), iDOC (format developed by SAP[15]), EDIFACT[16], etc., each one with their own implementation (the format is a standard but not the use of it). A more detailed description of the problem can be found in [30].

The solution proposed in WP8 is that each company emits the invoices in its own format, and this is an instance of its own model generated from a customized ontology network. This customized ontology network will be created from a reference ontology network which contains all the elements needed for the invoicing process. This invoice reference ontology network will in turn be created following the methodological guidelines specified by WP5.

The problem to be addressed when using a reference ontology network is that of the heterogeneity of the invoices generated and received by the different companies that participate in the invoicing process. This heterogeneity is planned to be solved by a network of federated ontologies with one reference ontology, which will contain all the concepts related with the electronic invoicing process. Companies will emit invoices from a model that they have previously adapted and the receiving system will be able to process these incoming invoices.

The ontology network for the invoicing use case in WP8 must contain all the concepts related to the invoice management in the pharmaceutical industry. These concepts shall reflect all the interactions and elements that are included in the pharmaceutical invoicing process. These elements vary from the most common concepts included in every invoice and generated by a company to the technologies or the workflow that is followed by every invoice in the system.

6.2.2. Selected ontology network life cycle model

The preliminary workplan for developing the Invoice Management ontology network included 4 main tasks and followed a waterfall life cycle model. The tasks were the following:

- *Task 1. Resource research*. Here it is performed a research process over the existing resources related to the invoicing problem is done.

- *Task 2. Design of the initial ontology network*. During this task the initial invoice ontology is created using the resources previously gathered; then, an analysis of time representations in ontologies is performed; afterwards, a time ontology is designed; such ontology has to be integrated into the invoice ontology. Finally, the analysis of the invoice workflow is carried out and an ontology, which represents such workflow, is built and integrated into the invoice ontology.

---

[15] http://www.sap.com/index.epx

[16] http://www.edifactory.de/messages.php?s=D05A

▪ *Task 3. Ontology validation*. Here it is performed a validation process of the invoice management ontology network created. The validation process checks the consistency of the ontology network.

▪ *Task 4. Instantiation of the invoice management ontology network*. During this task the ontology network previously created is instantiated for the specific use case, that is, for the pharmaceutical sector.

However, after holding several bi-lateral ad-hoc meetings between iSOCO and UPM, the initial workplan for developing the ontology network in the Invoice Management use case was modified, and the new plan is now presented in this section. During such meetings iSOCO explained the Invoice Management use case whereas UPM explained methodological guidelines that are being carried out in WP5, which can be used for building the Invoice Management Ontology Network.

The ontology network life cycle model chosen in the invoicing use case is finally the incremental model. After the meetings the iSOCO team held at iSOCO with the UPM team, the iSOCO team selected the ontology network life cycle model following the guidelines and the decision tree presented in Section 5.3. Based on the decision tree shown in Figure 36, the iSOCO team first chose *"no"* for the question *"Do you think that ontology network requirements will change during the development?"*. The iSOCO team answered *"no"* because the requirements are clearly specified in [30]. The most problematic issues to be faced in the use case are: a) to deal with the heterogeneity of the invoices emitted and received and b) how to manage the incoming data. These requirements remain invariable during the life cycle of the ontology network. As for the second question *"Do you want to produce intermediate results?"* the iSOCO team's answer was *"yes"*. During the evolution of this use case several prototypes and intermediate ontology networks will be produced and used. The answer to the third question *"Do you want to produce each intermediate result in a complete manner"* was "yes". So, the selected ontology network life cycle model was incremental.

6.2.3. Selected activities

Based on the guidelines and suggestions being developed in WP5 and included in this deliverable (Section 5.3) and on their experience (some members of the team have developed more than 5 ontologies), the iSOCO team selected directly the activities to be carried out during the ontology network life cycle from the "Required-if Applicable" table.

Activities identified as "required" in the "Required-if Applicable" table (table 3 in Section 3.3) are selected automatically in the table of selected activities. The iSOCO team only selected those "if applicable" activities needed for the Invoice Management ontology network development. So, Table 6 only shows those "if applicable" activities selected by the iSOCO team.

Table 6. Selected "If Applicable" Activities in the Invoice Management Use Case

	If Applicable	Selected
Ontology Aligning	X	X
Ontology Customization	X	
Ontology Enrichment	X	
Ontology Extension	X	X
Ontology Forward Engineering	X	
Ontology Learning	X	

	If Applicable	Selected
Ontology Localization	X	*X*
Ontology Matching	X	*X*
Ontology Merging	X	*X*
Ontology Modification	X	*X*
Ontology Modularization	X	*X*
Ontology Module Extraction	X	*X*
Ontology Partitioning	X	
Ontology Population	X	
Ontology Pruning	X	
Non Ontological Resource Reengineering	X	*X*
Ontology Reengineering	X	
Ontology Restructuring	X	*X*
Non Ontological Resource Reuse	X	*X*
Ontology Reverse Engineering	X	
Ontology Specialization	X	*X*
Ontology Summarization	X	
Ontology Translation	X	
Ontology Update	X	

6.2.4. Activities carried out for building the ontology network

Figure 39 shows the activities involved in the development of the invoice management ontology network; it presents the main scenarios (of those presented in Chapter 4) covered in this NeOn use case.
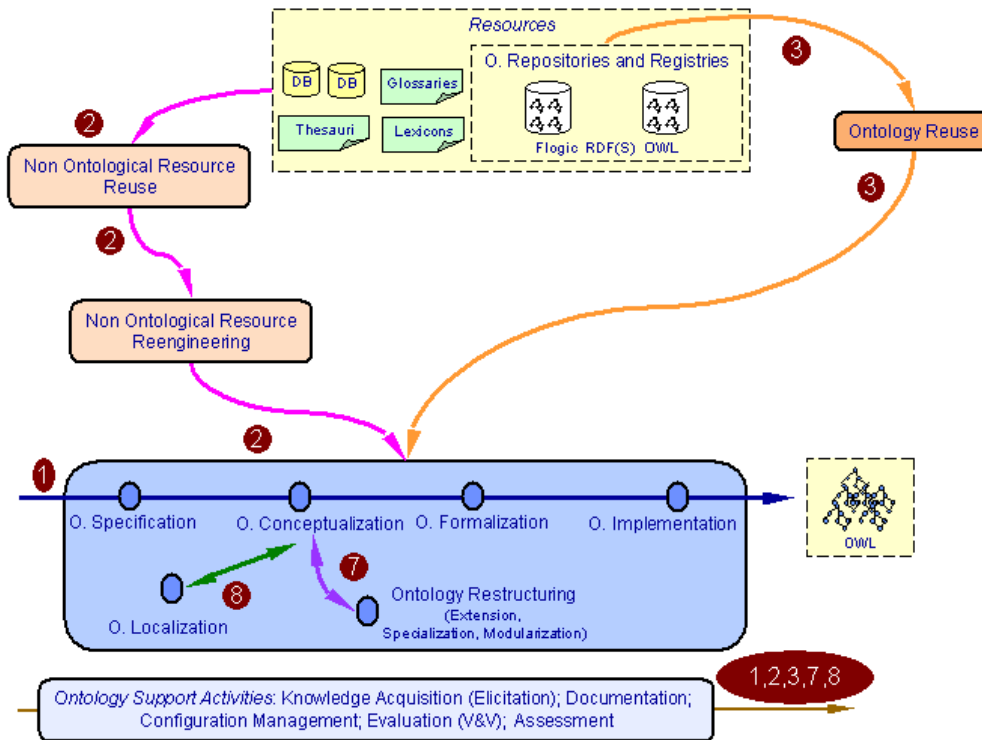
Figure 39. Main Scenarios for Building the Invoice Management Ontology Network

Here the main activities carried out or to be carried out for building the invoice management ontology network are explained. These activities are based on the methodological work being done in WP5. Some of such activities have been performed following the methodological guidelines provided by WP5.

➢ **Knowledge Acquisition and Ontology Elicitation**. The pharmaceutical domain was analysed in [30]. In such document a complete description of the pharmaceutical industry and the invoicing requirements has been given. The document also describes how the invoicing process is done and what actors and requirements are needed. It is important to highlight that such document also describes what happens to an invoice since it is emitted till the moment it is validated by the receiving company, who are the actors that participate in this process, which are the requirements of each company in each cluster (laboratories, wholesalers and providers) and how the NeOn technology will be used.

➢ **Ontology Specification**. The idea is to specify the needs that the ontology has to satisfy in the application, using competency questions (CQs). Competency questions are questions used when we want to analyze the need of use ontologies in a certain scenario. These competency questions have been written by ontology engineers to be answered by several domain experts. Competency questions highlight the need of having three different ontologies: one reference ontology that will integrate all the concepts used in the other two. This reference ontology will contain time concepts, invoicing technology terminology, invoicing concepts and other information related to the whole invoicing process. The second ontology will contain specific concepts related to the invoicing process followed by each company belonging to a cluster. (A laboratory invoice contains a high number of common concepts with a wholesaler or a provider invoice but it also contains enough different concepts to create a specific ontology for each cluster). The third ontology is a customization of the cluster ontology for each company. These companies will adapt the cluster/sector ontology to their particular needs.

The list of competency questions has been included in [31].

- ➢ **Ontology Reuse**. Ontologies are more common and available everyday. Ontologies can be reused and imported as modules into the ontology network to be built. Upper ontologies are a kind of ontology that contain a generic description of concepts. These descriptions will be used in the invoicing case study by reusing some of them, which are included in upper ontologies such SUMO [22], CyC[17] or DOLCE [27]. Additionally, specific ontologies related to the invoicing process that the companies follow and pharmaceutical ontologies are reused.

  The purpose of the reference ontology for invoicing is to be instantiated for different sectors of the industry. The first instantiation is for the pharmaceutical sector, laboratories mainly, but it will also be extended for providers of these laboratories or wholesalers. These providers supply from chemical products to clean products so they need different instantiations of the invoice reference ontology.

  Two are the upper level ontologies considered in this use case: DOLCE and SUMO. The reasons of choosing these two upper level ontologies is that both are in OWL and their scope fits into the use case. The iSOCO team want to highlight SUMO because there are projects[18] that address the extension of SUMO with invoice concepts. These two upper level ontologies were analysed before they are extended for the pharmaceutical invoicing use case and finally only the most relevant concepts are been used.

  Also identified from the competency questions was the need of using time ontologies. A set of time ontologies and its reuse study is presented in [24].

- ➢ **Non Ontological Resource Reuse and Reengineering**. From the competency questions the iSOCO team also identified the need to use invoicing technologies vocabularies. Resources about specific invoice technologies will be reused and included in the final invoice ontology (presented in [31]).

  These invoicing resources are mainly technologies for electronic invoicing. One of these technologies is the Universal Business Language (UBL) which is a common XML library of business documents, such invoices, purchase orders, etc. The vocabulary provides the user a library of XML schemata for reusable components like Address or Company and also common schemata for documents, such as "Invoice", "Order", or "Despatch Advice". UBL has been adopted by the Scandinavian governments as a legally required standard for sending invoices to governments.

  Another technology is Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT), which is a set of syntax rules to structure data. The main difference with UBL is that EDIFACT is not based on XML, though it has its own vocabulary and structure. EDIFACT is a standard developed under the United Nations and has been adopted by the International Organization for Standardization (ISO) as the ISO standard ISO 9735. EDIFACT provides a set of syntax rules to structure data, an exchange protocol (I-EDI) and standard messages.

  Other electronic invoicing vocabularies considered are Electronic Business XML (ebXML) and XML Common Business Library (xCBL). These two vocabularies are also based on XML but seldom used in the industry. The last version of xCBL dates from March of 2003; ebXML is the precursor of UBL. These technologies are not as important as the first two described before but they should be taken into account.

---

[17] http://www.cyc.com/cycdoc/vocab/merged-ontology-vocab.html

[18] http://ontolog.cim3.net/

The iSOCO team also reuse ideas from *projects whose main goal is to integrate the invoice vocabulary into ontologies*. The problem of integrating invoice vocabulary like the one provided by UBL into ontologies is not new. Projects like the ONTOLOG project[19] or the XBRL Ontology project[20] cover this issue. The goal of the ONTOLOG project is to extend SUMO with the UBL vocabulary of invoicing. The project extends SUMO with the elements necessary for representing the invoice concepts and all the related concepts and relations and also classes that define the vocabulary used in UBL. The XBRL Ontology Specification provides main concepts and properties for describing financial and economic data on the Semantic Web. This last approach creates, from scratch, the invoice ontology but it is still in an early stage. There also exist other projects about integration like [59], HARMONISE[21] or [29] in which semantic interoperability is achieved to integrate the different technologies.

➢ **Ontology Conceptualization**, **Ontology Formalization**, and **Ontology Implementation**. All the knowledge gathered from the analysis of the domain [30], from the needs identified from the competency questions, and from the existing resources identified previously are integrated in the development of the invoice ontology. The initial ontology network is built following the terminology from upper ontologies and using the knowledge extracted from the experts. Once the implementation is finished, the **Ontology Integration** of the reused time ontology and of the invoice technologies reused and reengineered is carried out.

The ontology is developed using Protégé 3.2 and the ontology language selected is OWL.

➢ **Ontology Specialization**. The final invoice reference ontology will be adapted to the cluster of companies that will use it, for instance, a laboratory. The invoice reference ontology will be specialized according to each cluster of companies' needs (in an initial phase only laboratories are included).

➢ **Ontology Localization**. Localization is not a very important need of this invoice ontology network but it should be taken into consideration. The users of this ontology belong to different regions of Spain in which different languages are used. Although Spanish is the official language in the entire national territory, in these regions other languages co-exist; that is why localization has to be taken into account.

As ontologies have to be adapted to other countries, the necessity of localization is strongly needed. Localization will be carried out initially for the different Spanish regions and then will be adapted to other countries.

The ontology network has been designed in English, so all concepts are written in English. It can be adapted by the end users to their own language and particular needs.

➢ **Ontology Evaluation**. The ontology network will be evaluated by the users of PharmaInnova. They will also create and evaluate the ontology network invoices. Their evaluation will consist of two steps: first, the end users validate the ontology, checking whether it has all the concepts they need; then they adapt the invoice reference ontology using the prototype created in the scope of WP8 and emit invoices that are instances of the adapted ontology. This second evaluation step requires longer time because the prototype is still in a development phase.

Figure 40 shows what the Invoice Management ontology network (for more detail [31]).
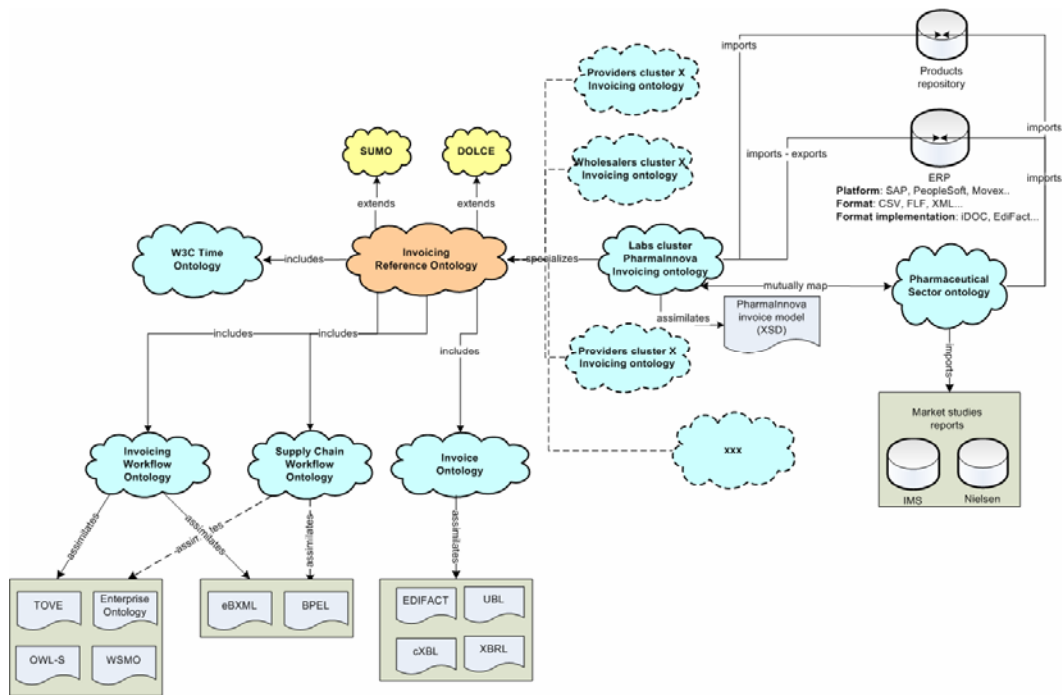

Figure 40. General View of the Invoice Management Ontology Network [31]

6.2.5. Lessons learned

The methodological proposal presented in this deliverable (Section 5.3) aims to establish the life cycle for a particular ontology network. This proposal has been an excellent guideline at the beginning of the developing process because it has permitted the iSOCO team to organize the development of the ontology network for the Invoice Management use case. The guidelines proposed in this deliverable have also helped the iSOCO team to select the activities to be carried out and to put them in order.

Furthermore, the meetings held between the UPM team and iSOCO have been extremely useful because in such meetings the UPM team provided iSOCO with methodological guidelines for some of activities to be carried out when developing the invoice management ontology network. Concretely, the use of the competency questions in the ontology specification activity helped the iSOCO team to identify the need to use a time ontology. Besides, the competency questions have shown the iSOCO team that the requirements specified in [30] were correct at the time of designing the whole ontology network. The requirements were firstly identified in [30] and then validated with the competency questions.

6.3. Nomenclature Ontology Network Life Cycle

6.3.1. Use case description

Sectors that use knowledge intensively such as the pharmaceutical, have typically to deal with heterogeneous and vast amounts of information. The Semantic Nomenclature case study is focused on integration of different and heterogeneous pharmaceutical repositories [30]. Information about medicine, medical and regulatory knowledge is highly distributed and, in some cases, inconsistent with the government regulation and its databases. As a solution to these problems, the ATOS team proposes a global Vademecum represented as a reference ontology, where the

distributed databases can be integrated and kept up-to-date with the government databases that provide the official and approved medicines information.

In the Semantic Nomenclature case study [30], semantics is used to classify meaningfully the information and to ease the integration of the distributed knowledge. This case study proposes the use of networks of ontologies against a large and unmanageable unique ontology to solve this integration problem. The main objective of the ontology network developed in the use case is to provide a model in several meaningful pieces and to bind them together, representing and managing relations between multiples ontologies that correspond to the main pharmaceutical databases in the Spanish market; another goal of this ontology network is to connect to the main international medical vocabularies.

In the last few years, several international bodies and organizations have provided a number of classifying systems, such as thesauri, taxonomical classifications and even ontologies about description and classification of medicines. However, in the current scenario there is no a unified way of dealing with and naming drug-related information [30].

The main actors in the Spanish pharmaceutical domain provide different databases with broad information about the drugs in the market [30]. Thus, the Ministry of Health along with the Spanish General Agency of Drugs and Sanitary Products edit and provide two official databases (Digitalis, Integra) with information on pharmaceutical products in Spain every month. Digitalis is the Nomenclature officially used for invoicing prescriptions and contains data such as the identification of the pharmaceutical product, prices, composition of the medicine, etc. Integra stores diverse information about pharmaceutical products consumed (and assimilated) in the hospital field.

The General Spanish Council of Pharmacists (GSCoP) provides its members with the Health Information Data Base (BOTPlus), which permits the access to harmonized, updated information on medicines, patients, diseases, treatments, etc. It includes up to 140.000 pharmaceutical products. This resource together with the two official vademecum of the government are the main sources of knowledge on drugs available in Spain [30].

There are also other international bodies providing several vocabularies and classification that should be taken into account. All the Spanish official pharmaceutical entities and companies classify drugs in their databases following the ATC (Anatomical Therapeutic Chemical) classification suggested by WHO[22]. Medicinal products are classified according to the main therapeutic use of the main active ingredient, following the basic principle of only one ATC code for each pharmaceutical formulation (i.e. similar ingredients, strength and pharmaceutical form) [30].

SNOMED (Systematized Nomenclature of Medicine) is a systematically organized computer processable collection of medical terminology that covers most areas of clinical information, such as diseases, findings, procedures, microorganisms and pharmaceuticals. SNOMED CT is used in the U.S. Federal Government systems for the electronic exchange of clinical health information. Other resource is MeSH (Medical Subject Headings), which is a huge controlled vocabulary (or metadata system) whose purpose is to index journal articles and books in the life sciences that deal with drugs and pharmaceutical preparations. MeSH contains around 23.000 subject headings arranged in a hierarchy and which can be viewed as a thesaurus [30].

Moreover, different medical thesauri, classifications and languages used by the international health community were identified in [30]. UMLS (a controlled compedium of medical vocabularies), LOINC, HL7, NCI, ATC classification, etc., are some examples of the several vocabularies that try to classify the different medical topics in the drug and pharmaceutical domains.

Besides, ontologies as Galen, OpenCyC or the NCI thesaurus try to define the concept of pharmaceutical product and classify different categories of a drug. However, up to now, there are not ontologies that describe the characteristics of the pharmaceutical products in detail. All these

---

[22] WHO: World Health Organization

resources and vocabularies are taking into account in the Semantic Nomenclature case study as source of knowledge of the domain [31].

The main scenario in the Semantic Nomenclature case study is to create the Nomenclature ontology network. The preliminary Semantic Nomenclature workplan for developing the Nomenclature ontology network is the following:

❑ First, to model and built the ontologies related to the main pharmaceutical databases of the case study (Digitalis, Integra, BOTPlus) and then, to populate them.

❑ From this point on, the following step planned was to model a reference ontology based in the previous ones.

❑ Finally, these ontologies were connected via mappings and provided with means for adding new domain-related ontologies to the network.

However, after holding several bi-lateral ad-hoc meetings between ATOS and UPM, the ATOS team agreed to modify the initial workplan for developing the ontology network in the Nomenclature use case; the modified plan is shown in this section. During such meetings, the ATOS team explained the Nomenclature use case whereas the UPM team described the methodological guidelines carried out in WP5.

6.3.2. Selected ontology network life cycle model

One of the first tasks the ATOS team carried out to obtain the Nomenclature ontology network life cycle was to identify which type of ontology network life cycle model was the most appropriate for the Semantic Nomenclature case study. Such identification was done by following the guidelines and suggestions being developed in WP5 and included in this deliverable (Section 5.3).

After studying and identifying the different requirements and restrictions of the ontology network development, reviewing the possible ontology network life cycle models and taking into consideration the past experiences of the ATOS team, the model selected was the *incremental ontology network life cycle model*, following a seven phase waterfall model.

The reason behind this decision was that the pharmaceutical scene is more or less static in their models and that the pharmaceutical sector has a low frequency of change in knowledge level (not in data level); therefore, the ATOS team deduced that nobody has planned to make changes in the ontology network requirements. Other main reason that motivated this decision was that there was already a plan to produce different versions of the ontology network and of the corresponding semantic application in the near future.

6.3.3. Selected activities

Based on the guidelines and suggestions provided in WP5 (see Section 5.3) and on the ATOS team experience (some members of the team have developed more than 5 ontologies), the ATOS team selected directly the activities to be carried out during the ontology network life cycle from the "Required-if Applicable" table.

Activities identified as "required" in the "Required-if Applicable" table (table 3 in Section 3.3) are selected automatically in the table whereas the ATOS team only selected those "if applicable" activities needed for the development of the Nomenclature ontology network. So, Table 7 shows only those "if applicable" activities selected by the ATOS team.

Table 7. Selected "If Applicable" Activities in the Nomenclature Use Case

	If Applicable	*Selected*
Ontology Aligning	**X**	*X*

	If Applicable	Selected
Ontology Customization	X	
Ontology Enrichment	X	X
Ontology Extension	X	X
Ontology Forward Engineering	X	
Ontology Learning	X	
Ontology Localization	X	X
Ontology Matching	X	X
Ontology Merging	X	
Ontology Modification	X	
Ontology Modularization	X	
Ontology Module Extraction	X	
Ontology Partitioning	X	
Ontology Population	X	X
Ontology Pruning	X	X
Non Ontological Resource Reengineering	X	X
Ontology Reengineering	X	
Ontology Restructuring	X	
Non Ontological Resource Reuse	X	X
Ontology Reverse Engineering	X	
Ontology Specialization	X	X
Ontology Summarization	X	
Ontology Translation	X	
Ontology Update	X	

6.3.4. Activities carried out for building the ontology network

The aim underlying the Nomenclature ontology network is to describe pharmaceutical information based on the different medical languages and thesaurus and on the data coming from the Spanish pharmaceutical databases and resources.

In this sense, the Nomenclature ontology network has been developed following the schemata of the main sources of information on drugs in Spain: Integra, Digitalis, BOTPlus and in the ATC Classification. Furthermore, this Nomenclature ontology network is networked with some common ontologies, such as Time ontology, Measure ontology, Geographical ontology and Laboratory ontology. The Nomenclature Ontology Network is connected via mappings to ontologies that

represent the knowledge of the main databases (Digitalis, BOTPlus, etc.) on pharmaceutical products in Spain [31]. This Nomenclature ontology network will have mappings with external sources of the same or similar domain, like Snomed, connecting it with other international medical vocabularies. Figure 41 shows how the Nomenclature ontology network looks like [31].

The Pharma Reference Ontology model is a compilation of the main terms and objects related to pharmaceutical products; it provides general characteristics of them and classifies the terms according to the ATC classification, which is the WHO recommendation that the pharmaceutical experts in Spain (Europe) follow.
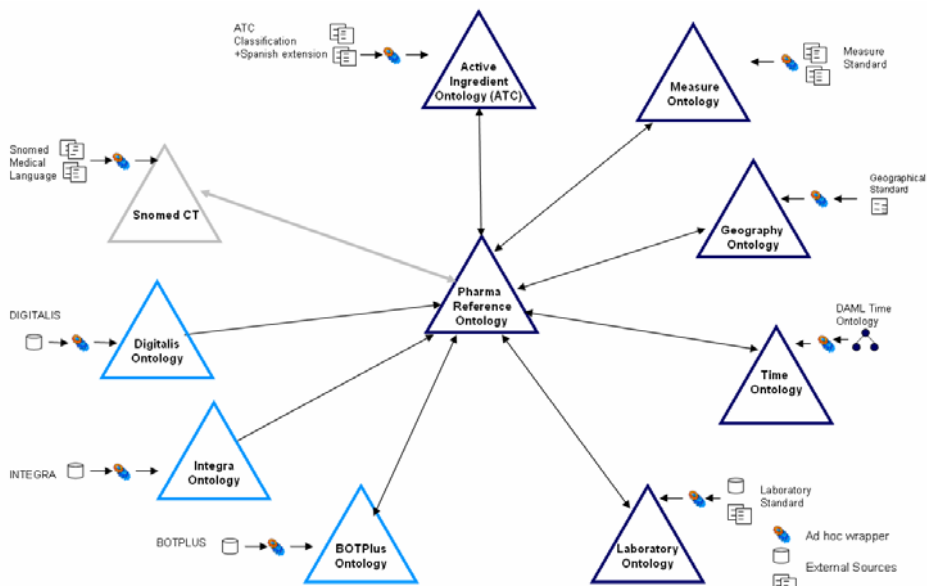

Figure 41. General View of the Nomenclator Ontology Network [31]

Figure 42 shows the activities involved in the development of the Nomenclature ontology network. It presents the main scenarios (of those presented in chapter 4) covered in this NeOn use case.
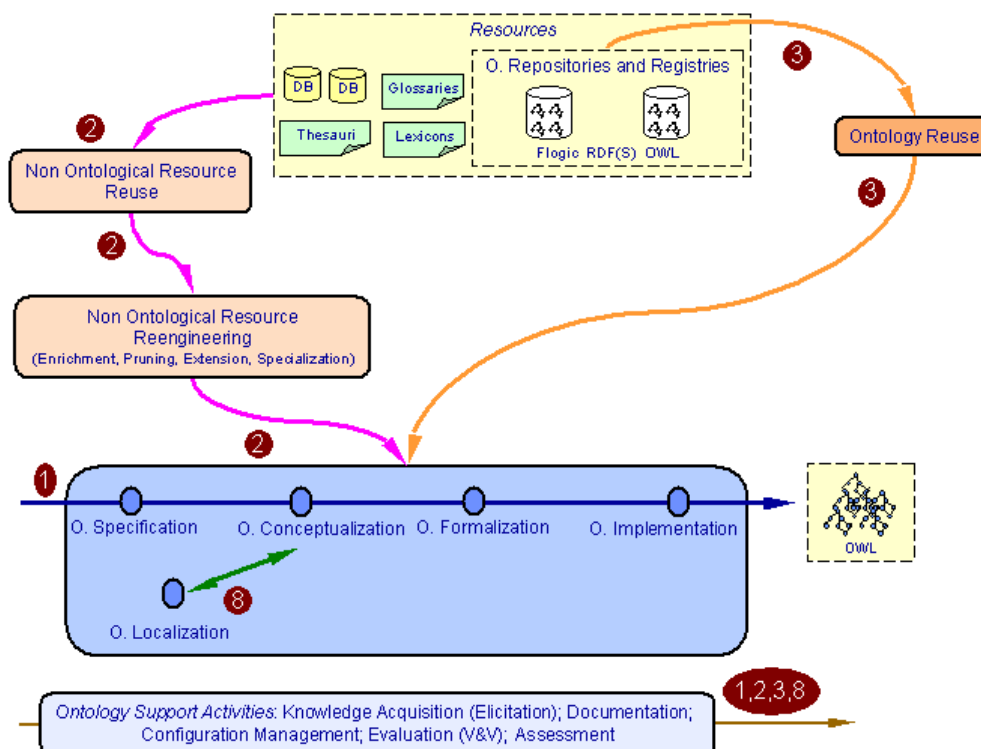
Figure 42. Main Scenarios for Building the Nomenclature Ontology Network

The main activities carried out or to be carried out for building the Nomenclature ontology network are explained here. These activities are based on the methodological work being done in WP5. Some of the activities have been carried out following methodological guidelines provided by WP5.

➢ **Knowledge Acquisition and Ontology Elicitation.** The pharmaceutical domain was specified and studied in [30]. This study analyzes the Spanish pharmaceutical sector in the recent years and studies in depth the current status of the communications as well as the problems of integration between the different actors when exchanging information about drugs. Additionally, the different interviews carried out with domain experts have given the ATOS team an insight of the Spanish pharmaceutical sector and of the resources directed into the Semantic Nomenclature case study.

➢ **Ontology Specification.** The ATOS team created a list of Competency Questions (CQs) to pinpoint the necessities that the ontology network has to satisfy in the Semantic Nomenclature application. The strategies for stating the different competency questions was: first, to define specific competency questions on the pharmaceutical and Semantic Nomenclature domain and then to compose more general competency questions from the answers taken from the specific ones. The list of competency questions has been included in [31].

➢ **Non Ontological Resource Reuse.** The idea was to select pharmaceutical standards that would cover most of the identified necessities. Pharmaceutical classification systems and different thesauri, taxonomies and vocabularies were identified. First, as previously described, the source of knowledge and data in a "local" are identified, trying to find reliable sources, such as the government and official pharmaceutical entities of Spain, since they provide the ATOS team with the main databases (Digitalis, Integra, BOTPlus). Then, the ATOS team looked for resources, such as online vademecum (Vademecum) or Catalonian sources, which were suggested to the ATOS team by pharmaceutical professionals. Then, medical vocabularies, thesauri and taxonomies (ATC, Snomed, UMLS, etc.) from international bodies related to pharmacy were analyzed to find how drugs are described, and so to find connections between

them and the Spanish models. More detailed analysis of the resources is described in [30] and in [31].

➢ **Non Ontological Resource Reengineering.** Many of the selected standards and medical vocabularies are not described in ontologies; therefore, these resources have to be reengineered and enriched into OWL ontologies before reusing them in the Nomenclature ontology network.

➢ **Ontology Reuse.** According to the needs recognized in the CQs, ontologies about general domain as Time, Measures, and Geography are identified, analyzed, evaluated and selected in order to be integrated in the Nomenclature ontology network.

➢ **Ontology Conceptualization**, **Ontology Formalization** and **Ontology Evaluation** of the Pharma Reference and the Laboratory ontologies.

➢ **Ontology Implementation** and **Ontology Integration.** The Phama Reference and the Laboratory ontologies are implemented with the first versions of the NeOn Toolkit provided by the consortium. The language selected for this implementation is OWL. Later on, the different ontologies implemented (reused and/or reengineered) are integrated in the implemented network.

➢ **Ontology Localization.** The Nomenclature ontology network is located in Spain and created in Spanish but it will be provided in English, too to internationalize the ontology and to link it to the main medical and pharmaceutical vocabularies of the world.

The Nomenclature ontology network should be maintained and supported at present and in the future. The network should evolve according to the changes and suggestions given by the Spanish pharmaceutical sector and by professionals.

6.3.5. Lessons learned

As mentioned before, this building process of the Nomenclature ontology network follows the methodological guidelines given by WP5. Such building process follows an incremental approach. The first version of the reference Nomenclature ontology network is based on the ATC classification and has been enriched by connecting it to general ontologies (e.g., time or geographical ontologies) to describe some properties and add more knowledge. Finally, some different ontologies about health vocabularies or new drug ontologies that have recently appeared in the web have been added to the network to increase and complete the knowledge of the domain.

The main reasons for following the methodological guidelines of WP5 are that they allow a good reuse of the pharmaceutical resources available and that they enrich the Nomenclator network ontology with ontology reuse of general ontologies (time, geographical) and non ontological reused standards (ATC classification).

Additionally, the approach proposed by WP5 for the Nomenclature ontologies permits modeling ontologies that are more focused on achieving goals to the Semantic Nomenclature application. With this network of ontologies it is easier to map the different ontologies and populate them. Other issue to take into account is the profit gained by carrying out Ontology Support activities that helps to develop successfully the Nomenclature ontology network.

7. Conclusions and Future Work

After analysing the state of the art, we can say that the degree of maturity of the Ontology Engineering field is very low if we compare it with the Knowledge Engineering field and, specially, with the Software Engineering field. The long term goal of the Ontology Engineering field will be to reach a degree of maturity similar to the one that the Software Engineering field has today. In this sense, the results included in this deliverable provide a step forward in the way of achieving the same degree of maturity in the Ontology Engineering field as in the Software Engineering field.

This deliverable presents and advance by means of the following contribution:

❑ The first version of the *NeOn Glossary of Activities*, which identifies and defines the activities potentially involved in the ontology network construction, included as part of the NeOn ontology development process, as a first step for solving the lack of a standard glossary in contrast with the *IEEE Standard Glossary of Software Engineering Terminology* [1] in the Software Engineering field.

  The definitions in the NeOn Glossary of Activities have been collaboratively built and consensuated by all NeOn partners.

  The activities in the NeOn Glossary have been divided into activities required for ontology network development and other that could be or not applicable, depending on the concrete case, and consequently non-essential or dispensable.

❑ Identification and description of *eight scenarios for building network of ontologies* collaboratively with special emphasis in reuse, reengineering and merging ontological and non-ontological resources. Such complex scenarios have been taken into account for the development of the work included in this deliverable, and they should be covered by the future methodological work to be carried out within WP5.

❑ The first collection of several *theoretical ontology network life cycle models*, based on those defined in the Software Engineering field and taking into account the specific features of the ontology network development. Such collection has been proposed in order to solve the lack of ontology life cycle models and ontology network life cycle models, in contrast with the collection of software life cycle models existing in the literature.

  Related to this issue, METHONTOLOGY proposes a unique type of life cycle model based on evolving prototypes for building single ontologies; On-To-Knowledge proposes an incremental and cyclic ontology life cycle model based on evolving prototypes; and DILIGENT proposes an ontology life cycle model based on evolving prototypes. However, it is known that there is no a unique life cycle model valid for all the developments. Each life cycle model is appropriate for a concrete development, depending on several features. For this reason, to propose a unique life cycle model for all the ontology network developments is not very realistic, and it is better to propose a collection of models and guidelines for deciding which model is the most appropriate in a concrete case.

  From the collection presented in this deliverable, we can briefly say that the waterfall ontology network life cycle model is the easiest model to understand, and that with this model it is also easy to schedule an ontology development. The same applies to the different proposed versions of this waterfall model that include knowledge resources reuse and reengineering and ontology merging. Regarding the incremental ontology network life cycle model, it permits to develop the ontology network by complete layers by following any type of waterfall model. This incremental model is also easy to understand, and has as advantage the development of the network in increments. Finally, the most sophisticated model is the spiral one that permits to analyze the different risks during the ontology network development.

❑ *Guidelines for obtaining the concrete life cycle for an ontology network*. Such guidelines are mainly based on two decision trees: one for selecting the ontology network life cycle model most appropriate for the concrete case and another for selecting which activities, from the NeOn Glossary of Activities, should be carried out.

❑ *Application of the contributions of this deliverable in two NeOn use cases (invoicing management and nomenclature ontology networks)*.

After the first step forward in the way of making progress the degree of maturity in the Ontology Engineering field presented in this deliverable, future methodological work (methods, techniques and tools for a selected set of activities in the NeOn Glossary of Activities) for continuing the advance will be included in D5.4.1 (*NeOn methodology for building contextualized networked ontologies*) at M24, with emphasis on pattern-based conceptualization, reengineering and reuse.

Deeper evaluations of the different ontology network life cycle models proposed and of the proposed guidelines in this deliverable will be carried out within task T5.6 (*Evaluation of the NeOn methodologies*) and will be included in deliverable D5.6.1 (*Experimentation with NeOn methodologies*) at M24.

The next version of this deliverable (D5.3.2 due to M36) will include a revision and an update of the presented work based on the experiments carried out in T5.6. And also we will include new models and guidelines based on the newest methodologies in Software Engineering.

References

1. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990 (Revision and redesignation of IEEE Std 792-1983).

2. *IEEE Guide for Developing Software Life Cycle Processes*. IEEE Std 1074.1-1995.

3. *IEEE Standard for Developing Software Life Cycle Processes*. IEEE Std 1074-1997 (Revision of IEEE Std 1074-1995; Replaces IEEE Std 1074.1-1995).

4. *IEEE Standard for Developing a Software Project Life Cycle Process*. IEEE Std 1074-2006. (Revision of IEEE Std 1074-1997).

5. ISO 12200:1999. *Computer applications in terminology. Machine-readable terminology interchange format (MARTIf). Negotiated interchange*.

6. ISO 1087-1:2000. *Terminology work. Vocabulary. Part 1: Theory and application*.

7. ISO 12207 Systems and Software Engineering - Software Life Cycle Processes. 2006.

8. German Ministry of Defense. *V-Model: Software lifecycle process model*. 1992. General Reprint Nº250. Bundesminister des Innern, Koordinierungs-und Beratungstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung.

9. NeOn Consortium. NeOn: Lifecycle Support for Networked Ontologies. *NeOn Technical Annex*. 2006.

10. US Department of Transportation. *Systems Engineering for Intelligent Transportation Systems*. 2007. Available at: http://www.ops.fhwa.dot.gov/publications/seitsguide/seguide.pdf.

11. F. Alonso, N. Juristo, J. Pazos. *Trends in life-cycle models for SE and KE: Proposal for a spiral-conical life-cycle approach*. 1995. International Journal of Software Engineering and Knowledge Engineering, Vol. 5, No. 3 (1995) 445-465.

12. F. Alonso, N. Juristo Juzgado, J. L. Maté, J. Pazos. *Software engineering and knowledge engineering: Towards a common life cycle*. Journal of Systems and Software 33(1): 65-79. 1996.

13. K. Beck, C. Andres. *Extreme Programming Explained: Embrace Change* (2nd Edition), 2004. Addison-Wesley, Boston.

14. M. Blázquez, M. Fernández-López, J.M. García-Pinar, A. Gómez-Pérez. *Building Ontologies at the Knowledge Level using the Ontology Design Environment*. 1998. In: Gaines BR, Musen MA (eds) 11th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'98). Banff, Canada, SHARE4:1–15.

15. B. W. Boehm. *Software engineering*. IEEE Transactions on Computers. Vol. C-25, pp. 1226-1241, December. 1976.

16. B. W. Boehm. *A spiral model of software development and enhancement*. ACM SIGSOFT Software Engineering Notes. Vol. 11, no. 4, pp. 14-24, August 1986; reprinted in Computer, vol. 21. no. 5, pp. 61-72, May 1988.

17. E.J. Byrne. *A conceptual foundation for software re-engineering.* In Proceedings of the International Conference on Software Maintenance and Reengineering, pages 226–235. IEEE Computer Society, 1992.

18. A. Cockburn, L. Williams. *The Costs and Benefits of Pair Programming* (2000). Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2000).

19. A. M. Davis, E. H. Bersoff, E. R. Comer. *A Strategy for Comparing Alternative Software Development Life Cycle Models.* IEEE Transactions on Software Engineering 14-10. 1453-1461. 1988.

20. Y. Ding, D. Fensel. *Ontology library systems: The key to successful ontology reuse.* In Proceedings of SWWS'01, The first Semantic Web Working Symposium, pages 93-112, Stanford University, California, USA, July 30 - August 1. 2001.

21. M. Engler, D. Vrandecic, Y. Sure. *A Tool for DILIGENT Argumentation: Experiences, Requirements and Design.* In Robert Tolksdorf and Elena Paslaru Bontas and Klaus Schild, 1st International Workshop on Semantic Technologies in Collaborative Applications STICA 06. IEEE, IEEE, Manchester, UK, June 2006.

22. S. Farrar, W. Lewis, T. Langendoen. *A Common Ontology for Linguistic Concepts.* In Proceedings of the Knowledge Technologies Conference, Seattle, Washington, March 10-13, 2002.

23. M. Fernández-López, A. Gómez-Pérez. *Overview and Analysis of Methodologies for Building Ontologies.* Knowledge Engineering Review (KER). Vol. 17, Nº 2, pp 129-156. 2002.

24. M. Fernández-López, A. Gómez-Pérez. *Searching for a Time Ontology for Semantic Web Applications.* International Conference on Formal Ontology in Information Systems, Turín, 2004.

25. M. Fernández-López, A. Gómez-Pérez, M. D. Rojas-Amaya. *Ontologies' crossed life cycles.* In: Dieng R, Corby O (eds) 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW 2000). Juan-Les-Pins, France. (Lecture Notes in Artificial Intelligence LNAI 1937) Springer-Verlag, Berlin, Germany, pp 65–79.

26. M. Fernández-López, A. Gómez-Pérez, N. Juristo. *METHONTOLOGY: From Ontological Art Towards Ontological Engineering.* 1997. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, pp 33–40.

27. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider. *Sweetening Ontologies with DOLCE.* In Proceedings of Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002. Lecture Notes in Computer Science 2473 Springer 2002, ISBN 3-540-44268-5.

28. R. García-Castro, M. C. Suárez-Figueroa, A. Gómez-Pérez, D. Maynard, S. Costache, R. Palma, J. Euzenat, F. Lécué, A. Léger, T. Vitvar, M. Zaremba, D. Zyskowski, M. Kaczmarek, M. Dzbor, J. Hartmann, S. Dasiopoulou. *Knowledge Web. Deliverable 1.2.4. Architecture of the Semantic Web Framework.* February 2007. Available at: http://knowledgeweb.semanticweb.org/.

29. N. Gessa, M. Busanelli, P. De Sabbata, F. Vitali. *Extracting a semantic view from an ebusiness vocabulary.* 8th IEEE International Conference on E-Commerce Technology (CEC 2006) and 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (EEE 2006). 26-29 June 2006, Palo Alto, California.

30. J. M. Gómez, C. Daviaud, B. Morera, R. Benjamins, T. Pariente Lobo, G. Herrero Cárcel, G. Tort. *NeOn Deliverable D8.1.1 Analysis of the Pharma Domain and Requirements*. September 2006. Available at: http://www.neon-project.org/.

31. J. M. Gómez-Pérez, C. Buil-Aranda, T. Pariente, G. Herrero. *NeOn Deliverable D8.3.1 Ontologies for the Pharmaceutical Case Studies*. 2007.

32. A. Gómez-Pérez, M. Fernández-López, O. Corcho. *Ontological Engineering*. November 2003. Springer Verlag. Advanced Information and Knowledge Processing series. ISBN 1-85233-551-3.

33. A. Gómez-Pérez, M. D. Rojas-Amaya. *Ontological Reengineering for Reuse*. Knowledge Acquisition, Modeling and Management: 11th European Workshop, EKAW 1999. Dagstuhl Castle, Germany, May 1999. LNCS Volume 1621/1999.

34. A. Gómez-Pérez, N. Juristo, C. Montes, J. Pazos. *Ingeniería del Conocimiento*. Editorial Centro de Estudios Ramón Areces, S.A. 1997. ISBN: 84-8004-269-9.

35. P. Haase, S. Rudolph, Y. Wang, S. Brockmans, R. Palma, and J. Euzenat, M. d'Aquin. *NeOn Deliverable D1.1.1 Networked Ontology Model*. November 2006. Available at: http://www.neon-project.org/.

36. E. Herrera-Viedma, F. Mata, L. Martínez, L. G. Pérez. *An Adaptive Module for the Consensus Reaching Process in Group Decision Making Problems*. V. Torra et al. (Eds.): MDAI 2005, LNAI 3558, pp. 89–98, 2005. Springer-Verlag Berlin Heidelberg 2005.

37. M. Iglesias, C. Caracciolo, Y. Jaques, M. Sini, F. Calderini, J. Keizer, F. Le Hunte Ward, M. Nissim, A. Gangemi. *NeOn Deliverable D7.1.1 WP7 User Requirements*. September 2006. Available at: http://www.neon-project.org/.

38. T. C. Jones. *Reusability in programming: A survey of the state of the art*. IEEE Transaction on Software Engineering. Vol. SE-IO, pp. 488-494, September 1984.

39. N. Juristo, J. Pazos. *Towards a joint life cycle for software and knowledge engineering*. 1993. Knowledge Oriented Software Design (A-27). Ed. J. Cuena. Pages: 119-138.

40. Y. Kalfoglou, M. Schorlemmer. *Ontology mapping: the state of the art*. The Knowledge Engineering Review. Volume 18, Issue 1 (January 2003). Pages: 1-31. 2003. ISSN:0269-8889.

41. J. Kingston. *Linking Knowledge Acquisition with CommonKADS Knowledge Representation*. BCS SGES Expert Systems 1994 conference, St John's College, Cambridge, 12-14 December 1994.

42. K. Kotis, G. Vouros. *The HCONE Approach to Ontology Merging*. In: The Semantic Web: Research and Applications. First European Semantic Web Symposium (ESWS 2004). Pp. 137-151.

43. S.L. Landau. *Dictionaries. The art and craft of lexicography*, Cambridge: Cambridge University Press. 1984.

44. B. Leuf, W. Cunningham. *The Wiki Way*. Addison-Wesley, 2001.

45. A. Newell. *The Knowledge Level*. Artificial Intelligence 18(1):87–127. 1982.

46. S. Pfleeger. *Software Engineering: Theory and Practice*. 2nd edition. Prentice Hall 2001. ISBN: 0-13-029049-1.

47. H. S. Pinto, C. Tempich, S. Staab. *DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies*. In Ramón López de Mantaras and Lorenza Saitta, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th, pp. 393--397. IOS Press, Valencia, Spain, August 2004. ISBN: 1-58603-452-9. ISSN: 0922-6389.

48. W. W. Royce. *Managing the development of large software systems: concepts and techniques*. In Proceedings Western Electronic Show and Convention (WESCON), Los Angeles. August 25th-28th 1970.

49. S. Saint, J. R. Lawson. *Rules for Reaching Consensus. A Modern Approach to Decision Making*. Jossey-Bass, San Francisco (1994).

50. G. Schreiber, B. Wielinga, R. de Noog, H. Akkermans, W. van de Volde. *CommonKADS: A comprehensive methodology for KBS development*. IEEE Expert: Intelligent Systems and Their Applications. Volume 9, Issue 6 (December 1994). Pages: 28-37. ISSN:0885-9000.

51. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van de Velde, B. Wielinga. *Knowledge engineering and management. The CommonKADS Methodology*. MIT press, Cambridge, Massachusetts. 1999.

52. I. Sommerville. *Software Engineering*. Eighth Edition 2007. Addison-Wesley. ISBN 0-321-31379-8.

53. S. Staab, H.P. Schnurr, R. Studer, Y. Sure. *Knowledge Processes and Ontologies*. IEEE Intelligent Systems 16(1):26–34. (2001).

54. L. Stojanovic. *Methods and Tools for Ontology Evolution*. Dissertation. 2004. Referee: Rudi Studer, Christof Weinhardt, Asunción Gómez-Pérez.

55. L. Stojanovic, N. Stojanovic, S. Handschuh. *Evolution of the Metadata in the Ontology-based Knowledge Management Systems*. 1st German Workshop on Experience Management: Sharing Experiences about the Sharing of Experience. Berlin. March 7-8, 2002.

56. R. Studer, V. R. Benjamins, D. Fensel. *Knowledge Engineering: Principles and Methods*. Data & Knowledge Engineering 25 (1998). PP. 161-197.

57. F. B. Viégas, M. Wattenberg, K. Dave. *Studying cooperation and conflict between authors with history flow visualizations*. Conference on Human Factors in Computing Systems 2004. Vienna, Austria. Pages. 575-582. ISBN:1-58113-702-8.

58. D. A. Waterman. *A Guide to Expert Systems*. Addison-Wesley, Boston, Massachusetts. 1986.

59. Y. Zhao. *Develop the Ontology for Internet Commerce by Reusing Existing Standards*. International Workshop on Semantic Web Foundations and Application Technologies. Nara Prefecture Public Hall, Nara, Japan. March 12th 2003.

60. *NeOn Deliverable D7.2.2. Revised and Enhanced Fisheries Ontologies*. NeOn project report. August 2007.

61. *NeOn Deliverable D7.4.1a. Software architecture for managing the fishery ontologies lifecycle based on the NeOn architecture and tools*. NeOn project report. August 2007.