OntEQAM – A Methodology for Assessing Evolvability as a Quality Factor in Software Ecosystems

Aseel Hmood[1], Philipp Schugerl[1], Juergen Rilling[1], Philippe Charland[2]

[1]Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada
{a_hmood, p_schuge, rilling}@cse.concordia.ca

[2] System of Systems Section
Defence R&D Canada – Valcartier
Québec, Canada
philippe.charland@drdc-rddc.gc.ca

Abstract

Software development and evolution are highly distributed processes that involve a multitude of supporting tools and resources. Knowledge relevant to these resources is typically dispersed over a wide range of artifacts, representation formats, and abstraction levels. In order to stay competitive, organizations are often required to assess and provide evidence that their software meets expected quality requirements. Similarly, the quality assessment of open source or third-party components is a crucial factor in software development. In our research, we focus specifically on modeling and assessing evolvability as a quality factor. We introduce our OntEQAM methodology that supports the integration and semantic analysis of knowledge resources typically found in software ecosystems. We further illustrate how our OntEQAM methodology can be applied to support the automated analysis and assessment of different types of quality attributes related to the evolvability of software ecosystems.

1. Introduction

The requirements software systems have to satisfy are affected by constant changes due to refinements by costumers or internal quality control. Developers are responsible for identifying functional requirements, implementing them, and ensuring that these requirements evolve according to customer needs during system maintenance tasks. However, from an organization perspective, it is not sufficient to only satisfy these functional requirements. Software systems must also satisfy non-functional requirements (NFR) such as security, safety, and dependability [3, 4], which are critical for the success of most software systems.

NFRs are often referred to as the qualities of a system and can be divided into two main categories: (1) execution qualities, such as performance and usability, which are observable at run time; and (2) evolution qualities, such as testability, maintainability, extensibility, and scalability, which are embodied in the static structure of the software system. Achieving these NFRs often plays a critical aspect for the acceptance of a final software product.

Despite the importance of NFRs in general and the fact that software maintenance contributes up to 90% of the total software life cycle cost [5, 6], most existing processes still lack adequate support for NFRs in software maintenance. Only more recently, organizations have started to change their traditional maintenance view from supporting perfective and corrective maintenance activities to a more comprehensive and complete life cycle perspective. As part of such an evolution perspective, evolvability has to become an integrated part of a maintenance process, involving planning, standardization, assessment, and prediction of maintenance activities. Our research addresses this need of potential stakeholders to assess the quality and more specifically the evolvability of software systems and addresses the challenges typically faced during assessment tasks.

The work in this paper is a continuation of our previous research on semantic modeling of software artifacts [20], process modeling [2], and semantic analysis [19, 45, 46]. We introduce our novel Ontology-Based Quality Assessment Methodology (OntEQAM) that focuses on the integration, semantic analysis, and assessment of knowledge resources typically found in software ecosystems. We illustrate how our OntEQAM methodology can be applied to derive a model that takes advantage of a unified knowledge representation and supports the automated analysis and assessment of different types of quality attributes related to the evolvability of software ecosystems. We argue that ontologies not only promote and support the conceptual representation of a software ecosystem, but also support the semantic modeling and assessment of quality factors.

Our research is significant for several reasons: (1) We introduce a generic methodology that can be applied to instantiate specific quality models. (2) We present an evolvability assessment model that not only provides a formal modeling of evolvability quality attributes, but can also be easily extended/customized to specific stakeholder needs. (3) We introduce a semantic-rich unified ontological representation for knowledge resources, digital components, and the quality assessment model. (4) We provide automated tool support for evolvability assessment of such software ecosystems.

The remainder of the paper is organized as follows: Section 2 provides background relevant to evolving software ecosystems, NFRs, and ontologies. Section 3 discusses the Ontology-Based Quality Assessment Methodology. Section 4 introduces the implementation of our assessment tool. Section 5 illustrates its applicability in assessing various quality aspects. Section 6 presents results, followed by conclusions and future work in Section 7.

2. Related Work

2.1 Evolving Software Ecosystems

Evolvability, the capacity for non-lethal heritable variation, is a striking property of biological systems that has not been successfully understood in its formal and system-theoretic aspects, nor has it been successfully modeled computationally or applied in software systems. In the software engineering domain, evolvability as a term has been widely popularized by Lehman et. al. [1] and their eight laws manifesting what constitutes software evolution. The challenge of achieving robustness, adaptability, and flexibility while facing changing requirements and environments remains a paramount issue for software ecosystems.

The term ecosystem was introduced in 1930 by Roy Clapham [16], who was asked if he could think of a suitable word to denote the physical and biological components of an environment that are considered in relation to each other as a unit. Some consider it as a basic unit in ecology, characterized by energy and matter flows between the different elements that compose it. From a software perspective, a *software ecosystem,* sometimes called a *digital ecosystem*, refers to software/digital components, which exist within a computer as a functioning unit. Digital components could be software components, applications, services, knowledge, business processes and models, training modules, contractual frameworks, or laws. A digital component is a useful idea, expressed by a language (formal or natural), digitized and transported within the ecosystem, which can be processed by humans or by computers.

From our perspective, a *software ecosystem* employs a broader set of components, such as software services, knowledge, representations of processes, artifacts at different levels of abstraction and at various levels of semantics. The infrastructure of such a software ecosystem consists of a pervasive digital environment, which is mainly populated by digital components and supports the description, composition, evolution, integration, sharing, and distribution of knowledge. The social and business network topologies found in these ecosystems are often not hierarchical [17], making it difficult to identify and implement a single functional reference model, due to the heterogeneity of the components. Similar to biological ecosystems, software ecosystems are not static. They must evolve in order to adapt to changing requirements and environmental contexts and still satisfy their expected functional and non-functional requirements. Most organizations have made a significant investment in developing and evolving their software ecosystems to ensure that their digital components within their ecosystem meet expected quality factors. For this reason, it becomes essential for an organization to assess, evaluate, and compare the quality of digital components, either in their existing digital ecosystem or product, to be added to the systems.

2.2 Quality and Quality Assessment

Quality is a widely used term to evaluate the maturity of development processes within an organization from a business point of view. Defining quality allows organizations to specify and determine if a product has met certain non-functional and functional requirements. However, as Kitchenham [21] states: "*quality is hard to define, impossible to measure, easy to recognize.*" Unlike functional requirements, where a single analysis technique (e.g., use case modeling) is sufficient to identify essentially all requirements, the same analysis is not appropriate for all quality requirements. Quality, as defined by ISO 9000:2000 [22], is the "*degree to which a set of inherent characteristics fulfills requirements*", where a requirement is a "*need or expectation that is stated, generally implied or obligatory*". There has also been a significant body of work on classifying requirements in order to establish links between qualities and requirements.

Software quality models are artifacts for describing the quality factors of a single software product of different types and domains. Quality Models have been defined by the ISO/IEC 14598 [7] standard as: "*The set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality*." Existing quality assessment models fall in one of two categories: (1) Models that are specific to commercial software systems (a.k.a. traditional or classical quality assessment models), focusing on the quality of the product itself [8, 9]. (2) Assessment and quality models that focus on open source projects emphasize often product, community aspects involved in the development of these systems (e.g., QSOS 2004 [10], QUALOSS [13]). Common to most traditional and open source models is that they share common classification structures to describe what constitutes quality and how it should be assessed.

Figure 1 shows a general structure used by most quality assessment models. In this model, the *quality dimension* perspective is used to group quality factors (e.g., external and internal qualities [8]). *Quality factors,* or also sometimes referred to as quality characteristics, can be any feature or characteristic of a product/service that is required to satisfy customer needs or achieve fitness for use. These quality factors are often non-measurable and typically used to classify more measurable quality sub-factors and quality attributes.

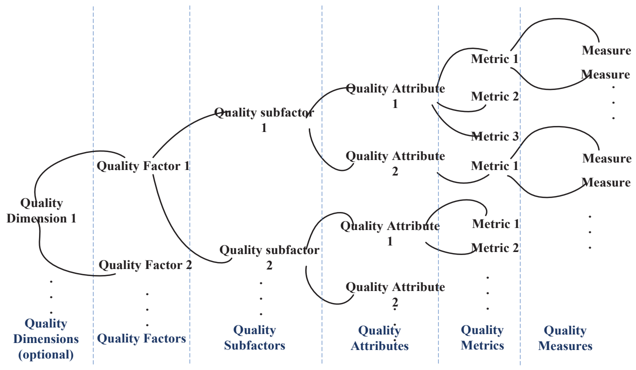

Figure 1. Generic structure of quality assessment models

These subjective measureable *quality sub-factors* can be further decomposed into *quality attributes*, the measurable physical or abstract properties of an artifact(s). Quality metrics are the directly measurable attributes of software and they are used to express quality aspects of a product through a numerical measurement. Each of these quality metrics is supported by *quality measures* that can be used to ascertain or appraise value by comparing it to a norm.

2.3 Ontologies

The Merriam Webster online dictionary[1] defines the term ontology as *"a branch of metaphysics concerned with the nature and relations of being and a particular theory about the nature of being or the kinds of existents."* Ontologies in computer science have been widely used as a formal, explicit specification of a shared conceptualization [23, 24]. Formal in this context refers to the fact that an ontology should be machine-readable, explicit to its ability to define concept types (vocabularies) and constraints (assumptions). Shared describes the fact that an ontology captures common agreed knowledge, that is, it is not private to some individual, but accepted by a group and conceptualization relates to the abstraction model used in ontologies to represent relevant concepts in a domain. Ontologies allow for the definition of basic terms and relations comprising the vocabulary of a domain as well as the rules for combining terms and relations to allow for the extensions to the vocabulary [23]. Ontologies are an important part of knowledge modeling and sharing by acting as a common language [25]. The Web Ontology Language (OWL) [27, 28] has long been standardized by the W3C and has paved the way for a machine-understandable Internet, in which Web resources can be automatically processed and reasoned about. More recently, ontologies have been introduced in engineering domains and are now widely accepted as an enhanced form of knowledge representation. Our research uses OWL-DL, a sub-language of OWL that is based on Description Logics (DLs) [26]. DL-based knowledge systems allow us to enrich our platform with reasoning services (e.g., Pellet). Reasoning systems within OWL-DL have proven to be sound, complete, terminating, and provide us with automatic inferences. For a more detailed discussion on OWL ontologies, DL, and reasoning, we refer the reader to [26, 28].

[1] http://www.merriam-webster.com/

3. OntEQAM Methodology

Given the financial investments organizations are facing when selecting, adopting, or replacing existing software systems, decision makers are required to assess potential solutions and their alternatives in order to justify their choice. In this research, we present our OntEQAM methodology to establish a quality assessment for software products. The Merriam-Webster dictionary defines a methodology as *"a body of methods, rules, and postulates employed by a discipline: a particular procedure or set of procedures"*. Based on this definition, we now derive our OntEQAM methodology for the quality assessment, which is shown in Figure 2.

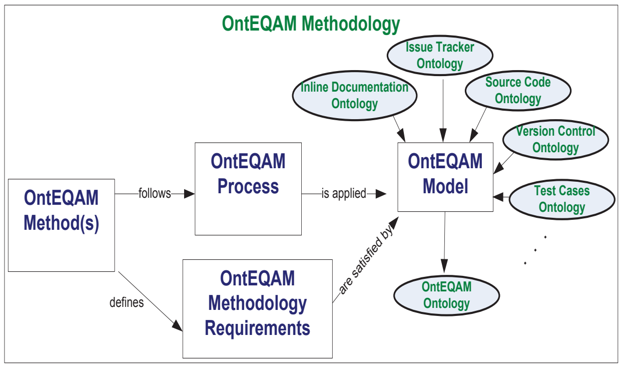

Figure 2. OntEQAM methodology

In our methodology, an OntEQAM method is a specific instance of the methodology that is defined by a user based on the context in which the methodology will be used. Our OntEQAM methodology assumes that in order for a newly defined method to be considered as an OntEQAM method, it has to satisfy both a set of predefined OntEQAM requirements and follow the OntEQAM process.

3.1 The OntEQAM Process and Requirements

The OntEQAM process represents a set of tasks and activities to be followed as part of instantiating and customizing an evolvability assessment method to meet a stakeholder's specific evolvability quality needs. The input to the OntEQAM process are software artifacts that are selected based on the specific stakeholder's needs and their availability, as well as the measurements that are going to be used to assess these artifacts. In the next step, a common ontological representation for these artifacts will be established by re-using existing models or customizing existing models to meet the requirements of these artifacts. As part of the model adjustment activity, quality metrics and measurements from the initial model can be customized and added to reflect a specific model context. The output of this process is a final assessment (ranking/score or rate) at both the individual artifact level and the overall product level. Figure 3 illustrates the high-level activities and major tasks involved in our OntEQAM assessment process to be followed during each assessment.

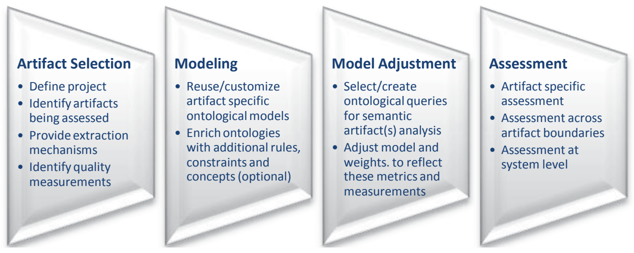
Artifact Selection
• Define project
• Identify artifacts being assessed
• Provide extraction mechanisms
• Identify quality measurements
Modeling
• Reuse/customize artifact specific ontological models
• Enrich ontologies with additional rules, constraints and concepts (optional)
Model Adjustment
• Select/create ontological queries for semantic artifact(s) analysis
• Adjust model and weights. to reflect these metrics and measurements
Assessment
• Artifact specific assessment
• Assessment across artifact boundaries
• Assessment at system level

Figure 3. OntEQAM process

For any method to become an OntEQAM method, the following set of constrains need to be satisfied (Figure 4).

#	Description
1	*Naming:* Requires a unique name and version number
2	*Objective:* Main objective has to be assessment of quality from an evolvability perspective
3	*Process:* Has to follow the OntEQAM process
4	*Model:* Has to be based on the generic OntEQAM model structure
5	*Users:* Specify subset of stakeholders provided by OntEQAM
6	*Usage scenarios*: A method must clearly state its usage scenario(s)
7	*Data sources:* Specify artifacts and resources being assessed
8	*Metrics:* Specify a set of (cross) artifact specific metrics
9	*Assessment weights:* The customization or ratings and weights has to follow the OntEQAM process

Figure 4. OntEQAM requirements

3.3 OntEQAM Model

Most of the quality factors, attributes, and metrics used in the base OntEQAM model are derived from existing work in quality assessment of open source and traditional software projects. The scope of our model however focuses specifically on factors, attributes, and metrics used to assess the evolvability quality of a software product.

The model assesses evolvability regarding two dimensions: *product* and *community*. These dimensions are similar to the dimensions used by QSO-OSS [11] and QUALOSS [13]. The *community* dimension looks at a product community and its ability to evolve the product, including developers, contributors, integrators, users (individuals or organizations) involved in the project, and how actively a product is maintained over an extended period of time.

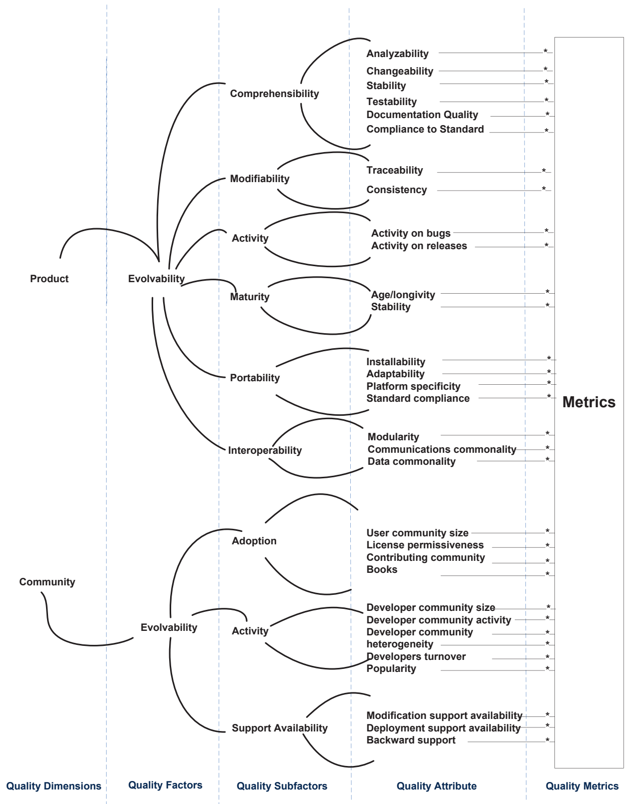

Figure 5. OntEQAM evolvability assessment model

Product has been previously considered in both traditional and open source assessment models**.** In our model however, we specifically assess a product's evolvability, which includes all major artifacts created as part of the development and evolution process. In our assessment model, *evolvability* is the only factor that we consider for the assessment and use it as a generic and not directly measurable upper categorization for evolvability sub-factors.

3.4 OntEQAM Metrics

One of the major challenges in assessing the quality of digital ecosystems is that relevant information in these systems is often distributed across artifacts. As a result, most of the current (semi-) automated assessment tools and models [14, 35] rely on measurements based on basic information retrieval techniques or token parsing within individual artifacts. Consequently, these metrics typically ignore structural and semantic information that otherwise could be found in either the individual software artifacts or by linking different artifacts. However, measuring quality factors, such as a system's evolvability, depend greatly on the ability to correlate distributed artifacts or knowledge resources. For example, the prediction of faults in a component might require access to bug tracking information, the source code of the component, and data from the version control system [11, 18]. In order to facilitate more semantic rich forms of analysis, knowledge resources need to be integrated in a common, semantic rich representation. Within our OntEQAM methodology, users can customize the OntEQAM model by integrating different types of queries, ranging from simple metrics to semantic rich metrics that take advantage of implicit knowledge inferred by ontological reasoners. The analysis results and metrics enrich our existing knowledge base and can be reused throughout the assessment process to support other metrics. In section 6, we discuss in more details some of the semantic rich metrics being supported by our OntEQAM methodology.

4. System Architecture Overview

Automated assessment support is provided by our SE-Advisor implementation, which is a continuation of our previous work [2, 19, 20]. In our current work, we extend our SE-Advisor to support our OntEQAM methodology. Knowledge integration and sharing, key aspects of our methodology, are performed through a unified ontological representation. For a more detailed discussion of our ontologies, their design, and integration, we refer the reader to [2, 20]. The ontological knowledge base (KB) is hosted as a central repository on a server, with persistent storage being provided by a Sesame triple store. Given the distributed environment in which our SE-Advisor is used, we adopted a client-server architecture in which the clients communicate over a network with a knowledge/advice server (Figure 6). The communication between the clients and server is implemented as a Representational State Transfer (REST) web service.

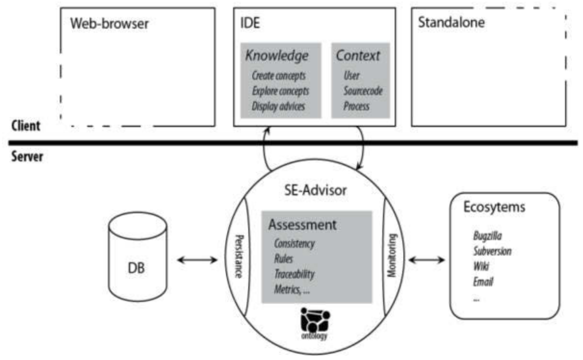

Figure 6. SE-Advisor system architecture

REST is a software architecture for stateless client-server communication and is used as a simple and intuitive method to realize well-defined CRUD (create, read, update, delete) operations through the standard HTTP protocol. The exchange format used is XML, which allows human-readable and therefore, easily analyzable and maintainable communication. In order to serialize objects to XML, the XStream [29] library is used. Our server application is deployed on a Tomcat server and secured through HTTPS.

The server is responsible for analyzing an ecosystem and assessing the collected information in order to populate the software engineering ontology (see Section 5). Monitoring is performed through system hooks or is time-triggered (i.e., nightly scheduled). Updates are propagated to the ontological KB, whereby information is pre-processed, aggregated, and assessed. Pre-processing includes the standardization of date/time types or the elimination of non-valid characters. Aggregation calculates certain metrics (e.g., LOC) or clusters text to paragraphs (e.g., in wikis and emails). Finally, during the assessment step, we generate and extract additional semantic rich information. A more detailed discussion of the analysis process is provided in the next section.

5. Assessing Evolvability Quality

In this section, we discuss in more detail how semantic rich evolvability quality metrics can be integrated within an assessment process. The quality of bug reports is presented as one metric not available to traditional assessment methodologies. It benefits from the ontological uniform representation that allows formulating assessment queries across artifact borders.

5.1 Artifact Metrics - Quality of Bug Reports

It has been shown [30] that the quality of reports originating from bug trackers or ticketing systems can vary significantly and have a direct impact on the time it takes to fix a particular bug. As a result, we include in our OntEQAM model a metric for assessing the quality of bug reports. For the metric, we apply both Information Retrieval (IR) and Natural Language Processing (NLP) techniques for mining bug repositories. Natural language properties influencing the report quality are automatically identified and applied as part of a classification task. As a result, our automated approach provides a semantic rich metric to assess the evolvability and maturity of bug trackers.

For the automated assessment, we introduced a new set of quality guidelines used for the evaluation of free form bug description typically associated with bug reports. The measurements themselves are derived from results observed in [23, 24] and from general guidelines for good report qualities, such the ones discussed in [18]. In order to validate our findings, we performed an assessment of the ArgoUML [31] bug repository. ArgoUML is a popular and mature open source UML drawing tool. We extracted a dataset consisting of 4,839 bug reports, which included 3,731 reported defects and 1,108 features and enhancements requests. An evaluation of our automatic classification approach with 178 manually classified bug reports showed a relatively high precision (86%) for our automated approach in assessing and classifying the bugs correctly.

5.2 Metrics across Artifact Boundaries

The extracted quality of bug reports and other bug tracker properties (such as links between bugs defined through *depends-on* or *resolved-by* properties) were mapped within our SE ontology. Through ontology alignment [32] between our bug tracker sub-ontology and source code ontology (described in Section 6.2), the revision numbers, bug-IDs, class, method, and namespace identifiers extracted through NLP were compared. It must be noted at this point that this process is not complete (in the case of ArgoUML, only about 50% of bugs could be matched with source code or vice-versa), but any new information gained through this alignment is valuable.

Given such knowledge about bug quality and its link to source code and other artifacts, we are able to derive within our OnteQAM approach additional metrics to correlate quality aspects across artifacts. This allows for a more precise evaluation of software systems in terms of their evolvability quality. For a more detailed discussion on our approach of evaluating the quality of bug reports, we refer the reader to [46, 47].

Similarly, we have developed quality assessment metrics for Javadoc comments and were able to demonstrate that the quality of Javadoc comments has a direct correlation to the number of bugs being reported in these classes.

5.3. Threats to Validity

OntEQAM is a novel assessment methodology, which is distinguishable from other existing work by its focus on the evolvability aspect of both commercial and FLOSS software products. The methodology takes into account the open source community, which is an aspect neglected in traditional models [14, 35]. While most existing models do not provide a full methodology to perform the assessment, OntEQAM represents a complete methodology that has a model, process steps, and requirements.

In contrast to other approaches [10, 11, 13, 15], our OntEQAM methodology takes advantage of a unified ontological representation that allows us to use ontological queries and reasoning services to derive semantic rich metrics, while supporting a fully automated assessment process.

Due to its open definition and stack of different assessment metrics that can be tailored for individual needs, OntEQAM can be ported to different product domains with ease. In comparison to Navica OSMM [14], OntEQAM provides a reasonable final evolvability assessment score (range 0-4). Figure 7 illustrates the use of assessment weights and scoring in our OntEQAM model. The weights in the model can be customized depending on the specific assessment context.

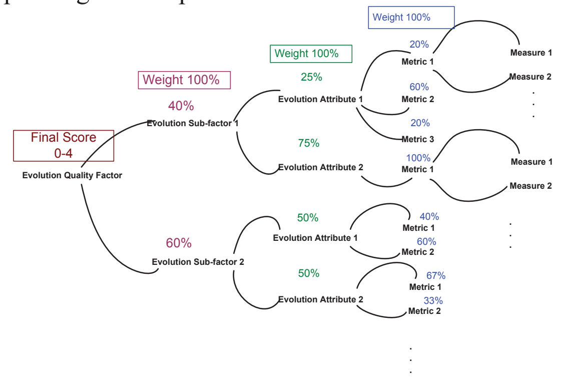

Figure 7. Customizable weights

Software ecosystems consist of a number of digital components and artifacts that need to be analyzed and evaluated as part of an assessment. Our OntEQAM methodology addresses this issue by supporting a set of predefined artifacts (e.g., issue trackers, version control systems, source code, documentation). However, our ontological model can easily be extended to accommodate additional artifacts (e.g., wiki pages, mailing lists, test repositories). In contrast to our OntEQAM methodology, many existing tools focus mainly on source code artifacts and bug reporting system, and lack support for additional artifacts, limiting their ability to assess the evolvability quality of software ecosystems.

Furthermore, our ability to explore and analyze information at different abstraction levels such as artifact level (e.g., issue tracker), instance level (e.g., source code file), inter-artifact level (e.g., shared concepts between two artifacts) provides additional insights to support the assessment of evolvability quality.

6. Expected Results

Typically, NFRs are gathered and documented during system analysis to guide decisions during system design and implementation. Existing work on requirements classification has focused on establishing links between qualities and requirements [33]. Other work has focused on making NFRs an integrated part of the development [34] or certification process (e.g., DO-178B [35]) of newly developed software. The focus of our research however is on the evolvability assessment of existing software ecosystems. There exist a number of open source and traditional quality assessment models. The main difference of these models compared to our approach is that they try to assess quality of a software product in general, with no specific focus on the evolvability aspects. However, our model focuses mainly on the evolvability quality. Another body of research exists in determining how open source development differs from the traditional methods used in the proprietary world and whether these differences impact the quality of code and of software products. In contrast, our research objective is not to demonstrate that one development method is superior over another one. Rather, we focus on a methodology that specifically focuses on assessing evolvability quality of software products found in digital ecosystems.

There are many application examples for OWL as a knowledge representation in the software engineering domain, such as model-driven software development [36], a CMMI-SW model representation and reasoning for classifying organizational maturity levels [37], component reuse [38], and open source bug tracking [39]. However, there is only limited research in modeling software evolution using ontologies. Ruiz et al. [40] present a semi-formal ontology for managing software maintenance projects. They consider both the static and dynamic aspects such as the workflow in software maintenance processes. Kitchenham et al. [41] designed a UML-based ontology for software maintenance to identify and model factors that affect the results of empirical studies. Dias et al. [52] extended the work of Kitchenham by applying a first order logic to formalize knowledge involved in software maintenance. González-Pérez and Henderson-Sellers present a comprehensive ontology for software development [53] that includes a process sub-ontology modeling, among others, techniques, tasks, and workflows. Despite considerable research on ontologies representing functional requirements, little work exists in using ontologies to represent non-functional requirements. In [21], a NFR ontology was introduced that can be used to structure and express constraints as part of a quality of service specification. More recently, the collaborative nature of software engineering has been addressed by introducing Wiki systems into the SE process. Semantic Wiki extensions like Semantic MediaWiki [45] or IkeWiki [44] add formal structuring and querying extensions based on RDF/OWL metadata. This work can be seen as complementary to our approach, in that they can support the creation and visualization of the developed KB. However, by themselves they do not address the main concern covered by our approach, i.e., the automated assessment of the evolvability quality of software ecosystems.

7. Conclusions and Future Work

Software systems in general must satisfy NFRs that are often also referred as quality properties, such as security, safety, and evolvability.

The main contribution of the research is the introduction of our OntEQAM assessment methodology that focuses on the assessment of the evolvability quality of software ecosystems. Our OntEQAM methodology supports both the integration and semantic analysis of knowledge resources typically found in these ecosystems. We further illustrate how our OntEQAM methodology can be applied to support the automated analysis and assessment of different types of quality attributes related to the evolvability of software ecosystems.

As part of our future work, we plan to enrich our assessment model with additional metrics. We are also planning to expand our work in other NFR areas, such as security and performance assessment.

Acknowledgements:

This research was partially funded by DRDC Valcartier (contract no. W7701-081745/001/QCV)

References:

[1] M. Lehman, "On understanding laws, evolution and conservation in the large program life cycle," Journal of Systems and Software, vol. 1, 1980, p. 213–221.

[2] W. J. Meng, J. Rilling, Y. Zhang, R. Witte, and P. Charland. An Ontological Software Comprehension Process Model. In: 3rd Int. Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa, Italy, 2006.

[3] I. Sommerville, Software Engineering (7th Edition), Addison Wesley, 2004.

[4] D. Parnas, "Software aging," Proceedings of the 16th international conference on Software engineering, IEEE CS Press Los Alamitos, CA, USA, 1994, p. 279–287.

[5] H. Muller, K. Wong, and S. Tilley, "Understanding software systems using reverse engineering technology," Object-Oriented Technology for Database and Software Systems, pp. 240–252.

[6] D. Edelstein, "Report on the IEEE STD 1219–1993 standard for software maintenance," ACM SIGSOFT Software Engineering Notes, vol. 18, 1993, p. 95.

[7] L. Erlikh, "Leveraging Legacy System Dollars for E-Business," IT Professional, vol. 2, 2000.

[8] J. McCall and P. Richards, "Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality.," Quality, vol. I, 1977.

[9] B. Boehm, J. Brown, and M. Lipow, "Quantitative evaluation of software quality," Proc. of the 2nd Int. conference on Software engineering, IEEE Computer Society Press Los Alamitos, CA, USA, 1976, p. 592–605.

[10] http://www.qsos.org/

[11] http://www.sqo-oss.eu/

[12] G. Gousios, V. Karakoidas, K. Stroggylos, P. Louridas, V. Vlachos, and D. Spinellis, "Software Quality Assessment of Open Source Software," Proceedings of the 11th Panhellenic Conference on Informatics, 2007

[13] http://www.qualoss.org/

[14] http://www.navicasoft.com/pages/osmm.htm

[15] http://www.osspartner.com/.

[16] Willis A J. The ecosystem: an evolving concept viewed historically, Functional Ecology 11:2, pp 268-271,1997.

[17] A. L. Barabási, H. Jeonga, Z. Néda, E. Ravasz, A. Schubert and T. Vicsek. Evolution of the social network of scientific collaborations, 2002.

[18] K. Cs and M. IT, "Supporting Software Evolution in Agent Systems," goanna.cs.rmit.edu.au, 2008.

[19]Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "Ontological approach for the semantic recovery of traceability links between software artefacts," IET Software, vol. 2, 2008, p. 185.

[20] J. Rilling, R. Witte, P. Schugerl, and P. Charland. *Beyond Information Silos — An Omnipresent Approach to Software Evolution.* Int. Journal of Semantic Computing (IJSC), Special Issue on Ambient Semantic Comp., May, 2009.

[21] Kitchenham, B., Travassos, G.H., Mayrhauser, A.V., Niessink, F., Schneidewind, N.F., Sing-er, J., Takada S., Vehvilainen R., and Yang H., "Towards an Ontology of Software Maintenance", Journal of Software Maintenance and Practice, 11(6), 365-389, 1999.

[22] ISO/IEC 9126-1. Software engineering – Product quality – Part 1: Quality model. International Organization for Standardization, 2001.

[23] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj and T. Zimmermann. What Makes a Good Bug Report? (Revision 1.1), 2007.

[24] Simon Tatham. How to Report Bugs Effectively. www.chiark.greenend.org.uk/~sgtatham/bugs.html (last accessed 1/2010).

[25] Gruber, T. R. A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5(2):199-220, 1993.

[26] F. Baader et al. The Description Logic Handbook, Cambridge Univ. Press, 2003.

[27] Web Ontology Language, www.w3.org/2004/OWL/ (last accessed 1/2009).

[28] OWL Web Ontology Language Reference, W3C Recommendation, www.w3.org/TR/owl-ref (last accessed 1/2009).

[29] The XStream XML Serialization Library, xstream.codehaus.org (last accessed 2/2009).

[30] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj and T. Zimmermann. Quality of Bug Reports in Eclipse, Proc. of the 2007 OOPSLA Workshop on Eclipse Techn, eXchange, Montreal, Canada, 2007.

[31] ArgoUML Open Source Modeling Tool, argouml.tigris.org (last accessed 2/2009).

[32] N. F. Noy and H. Stuckenschmidt. Ontology Alignment: An annotated Bibliography. In Semantic Interoperability and Integration, Schloss Dagstuhl, Germany, 2005.

[33] IEEE. IEEE Recommended Practice for Software Requirements Specifications. IEEE Standard 830, 1993.

[34] Dobson, G.; Hall, S.; Kotonya, G. (2007).A Domain-Independent Ontology for Non-Functional Requirements, e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on e-Business, Volume , Issue , Page(s):563 – 566, 2007.

[35] RTCA Inc., RTCA/DO-178B: Software considerations in airborne systems and equipment certification, 1992.

[36] Ontological Driven Architectures and Potential Uses of the Semantic Web in Systems and SE, www.w3.org/2001/sw/BestPractices/SE/ODA/.

[37] Soydan, G. H. and Kokar, M., "An OWL Ontology for Representing the CMMI-SW Model", W. on Semantic Web Enabled Software Engineering (SWESE), 2006.

[38] Happel, H.-J., Korthaus, A., Seedorf, S., and P.Tomczyk, "KOntoR: An Ontology-enabled Approach to Software Reuse", 18th Int. Conference on Software Engineering and Knowledge Eng. (SEKE), July, 2006.

[39] Ankolekar, A., "Supporting Online Problem –Solving Communities with the Semantic Web", Ph.D.Thesis, Carnegie Mellon University, Pittsburgh, PA, 2005.

[40] Ruiz, F., Vizcaíno, A., Piattini, M., and García F., "An Ontology for the Management of Software Maintenance Projects", International Journal of Software Engineering and Knowledge Engineering, 14(3), 323-349, 2004.

[41] Kitchenham, B., Travassos, G.H., Mayrhauser, A.V., Niessink, F., Schneidewind, N.F., Sing-er, J., Takada S., Vehvilainen R., and Yang H., "Towards an Ontology of Software Maintenance", Journal of Software Maintenance and Practice, 11(6), 365-389, 1999.

[42] Dias, M. G. B., Anquetil, N., and Oliveira, K. M.: Organizing the Knowledge Used in Software Maintenance, J. of Universal C.S., pp. 641-658, 2003.

[43] C. Gonzalez-Perez, B. Henderson-Sellers,"Modelling software development methodologies: A conceptual foundation", Journal of Systems and Software 80(11): 1778-1796, 2007.

[44] S. Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In WE-TICE, pages 388–396, IEEE Computer Society, 2006.

[45]M. Krötzsch, D. Vrandečić, and M. Völkel. Semantic MediaWiki. In I. Cruz, S. Decker, D. Allemang, C. Preist,D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, The Semantic Web – ISWC 2006, volume 4273 of LNCS, pages 935–942, Springer, 2006.

[46] Schugerl, P. and Rilling, J. and Charland, P. Mining Bug Repositories – A Quality Assessment, In Proc. Of the International Conference on Innovation in Software Engineering (ISE 2008), Vienna, Austria, 2008.

[47] Schugerl, P. and Rilling, J. and Charland, P. Enriching SE Ontologies with Bug Report Quality, 4th Int. Workshop on Semantic Web Enabled Software Engineering (SWESE 2008), Germany, 2008.