3 Results and Discussion

3.1 Answer to the First Research Question

The classification of requirements in the target system (Case 1), as demonstrated by Table 2, showed that the ontology of functional requirements classes provided by the given domain model is exceedingly comprehensive; of the 11 requirements classes found in the case under study, 10 were already existing as part of the ontology of the requirement classes in the given domain model; only requirements of type post-condition with a negligible share of 0.69% of the total number of requirements in Case 1 were not covered by a requirements class in the given domain model. The domain model was capable of predicting 99.31% of all the requirements in the studied system, which is a remarkably strong result with significant potential implications for domain-specific engineering of software systems. The two requirements classes of external behaviour and external call were not observed in the studied system.

Although the domain model predicted the classes of requirements in Case 1 with a high degree of accuracy, findings from a single case, although insightful, is often not convincing enough to enable us to generalize the findings to the entire population of enterprise systems. Therefore, as mentioned earlier, we replicated the study with two more cases: Case 2 and Case 3. As shown in Table 3, there were 7 classes of functional requirements in the dataset selected from Case 2, of which 6 were predicted by the given domain model. We only observed one new type of requirement class that was not part of the domain model, namely the data source requirement class with a share of 6% of requirements in the studied dataset. 94% of the requirements in the dataset for Case 2 were covered by the domain model. We did not observe data input, data validation, external call, external behaviour, data persistence, and communication requirements in Case 2.

In Case 3, as shown in Table 4, 100% of the requirements types were predicted by the domain model; no new requirements types were observed that were not already part of the domain model. The four requirements classes of external calls, external behaviour, communication, and data persistence were not observed in the dataset of Case 3.

Overall, the fact that the domain model was capable of predicting the types of 99.31% of functional requirements in Case 1, 94% in Case 2, and 100% in Case 3 enhanced our confidence that our findings generalize to the domain of enterprise application.

Another way to look at our study is that we analyzed 677 atomic functional software requirements in the domain of enterprise application and we only found 7 statements of requirements that could not be classified under one of the categories provided by the given domain model. The remaining 670 requirements, accounting for 98.96% of the total number of requirements in our three data sets, were covered by the 12 requirements classes in the original domain model. We only need to add two new requirements classes, namely post-condition and data source, to achieve 100% coverage in all the of the three studied system.

Case studies like the ones reported in this paper not only help us to evaluate our domain models, but also to refine our original models to achieve even higher predictive power. For instance, after conducting the three case studies reported in this paper, we refined the ontology of the original domain model by adding to it the two newly discovered requirements classes. The resulting ontology with its 14 requirements classes and a brief description of each class have been presented in Table 7 in the Appendix section of this paper.

Software engineering, in general, deals with two spaces: the problem space and the solution space. While the problem space deals with exploring and specifying the problem to be solved, the solution space, deals with addressing the problems identified in the problem space through activities such as solution architecting, software design, and implementation. What make solution space activities daunting, time-consuming, error-prone, and costly is the infinite software design space, demanding a great deal of creativity and experience on the part of software engineers. With this in mind, the requirements domain ontology we evaluated in this research project can be a powerful means for advancing the field of domain-specific software engineering because, in essence, it gives us a small set of requirements categories - or classes of problems or problem dimensions - that comprehensively describes the make-up of the specifications of the requirements for software systems in a particular domain such as the domain of enterprise application. It gives us a way to organize the problem space in a domain into a small set of requirements classes, which, in turn, facilitates solution space activities.

In theory, if, as we demonstrated in this paper, the ontology of requirements types provided by the domain model for enterprise systems is comprehensive, then the solution space activities are reduced to addressing the 14 classes of requirements that make up the problem space in any enterprise application. In other words, the otherwise infinite software design space is now reduced to being able to address 14 types of problems in order to be able to develop any software application within the domain of enterprise systems. This gives us the capability to document our requirements classes along with the best practices to address them and create domain handbooks to facilitate knowledge transfer and increase productivity as well as quality in developing software systems in a domain.

In practice, we do not even need to devise a solution to every category of problems identified in the domain model as some of these requirements classes, as indicated by their frequency distributions, occur very infrequently; we just need to identify the frequently-occurring requirements categories for a domain and address those in order to be able to provide solutions for a large number of problems in a domain. This raises the questions how frequently different classes of requirements occur in systems in a domain? And what the core requirements types are in a domain? These are the subjects of our next research questions. In what follows, we will answer these questions.

3.2 Answer to the Second Research Question

The domain model identifies data outputs, data inputs, event triggers, business logic, and data persistence as the five dominating classes of functional requirements in the domain of enterprise systems. These classes of requirements each had a contribution of more than 10% to the total number of requirements in the systems that were used to develop the domain model. As indicated by Table 2, data from our study showed that data inputs, data outputs, and event triggers were indeed among the most frequently-occurring requirements classes in the studied case (Case 1) as predicted by the given domain model for enterprise systems. However, the two requirements classes of business logic and data persistence were not among the core requirements classes in the studied system; instead, the three requirements classes of user interface and user interface logic followed by user interface navigation were among the most frequently-occurring categories of requirements. This is an interesting observation because the three requirements classes that were not predicted by the given domain model as core requirement types are all user interface-related. This observation led to a hypothesis that *requirements specification practices in the industry vary considerably in terms of the emphasis they place on having a detailed textual specification of their user interface-related requirements*.

In practice, it is not uncommon for development organizations to capture their user interface-related requirements using wireframes, prototypes, screen mocks, and other similar techniques that are visual rather than textual. As a result, fewer user interface-related requirements end up in the requirements specification documents, which can introduce noise in the predictive models that are built based on these textual specifications. This phenomenon can be observed in the model presented by Table 1, where the median values for the user interface navigation, user interface, and user interface logic class of requirements in the 15 systems that were used to build the original domain model are 4.57, 0.00, and 0.49, respectively. These low median values are an indication that in many of these enterprise systems user interface-related requirements were not thoroughly specified textually within their corresponding requirements specifications.

To evaluate the validity of this hypothesis, we removed all of the three user-interface related classes of requirements from both the original dataset, comprising of the 15 enterprise systems that were used to build the domain model, as well as the main target system (Case 1) in our study. The refined ontological-statistical domain model is shown in Table 5. The frequency distribution of requirements classes in the Case 1 after removing the user interface-related requirements classes is shown in Table 6.

TABLE 5 REFINED ONTOLOGICAL-STATISTICAL REQUIREMENTS DOMAIN MODEL FOR ENTERPRISE SYSTEMS - NOT CONSIDERING USER INTERFACE RELATED CLASSES

Requirement Class	**Percentage of Total Requirements**	**Average (%) Over 15 Observed Systems**	**Standard Deviation**	**Median (%)**
Data Output	28.81	24.64	12.05	23.07
Data Input	21.72	21.81	5.36	19.56
Event Trigger	17.68	13.22	8.89	11.53
Business Logic	12.74	16.36	9.85	14.42
Data Persistence	11.84	16.64	13.46	14.45
External Call	2.87	3.30	6.03	0.00
Communication	2.51	1.43	2.21	0.00
Data Validation	1.07	1.86	2.77	0.00
External Behaviour	0.71	0.70	1.79	0.00

TABLE 6 NON-USER INTERFACE-RELATED REQUIREMENTS CLASSES AND THEIR FREQUENCY DISTRIBUTIONS IN THE TARGET SYSTEM (CASE 1)

Requirement Class	**Count of Requirements**	**Percentage of Requirements**
Event Trigger	145	45.45
Data Input	97	30.40
Data Output	55	17.24
Business Logic	11	3.44
Data Validation	4	1.25
Post Condition	4	1.25
Communication	2	0.62
Data Persistence	1	0.31
Total	319	100

As it can be observed from the data of Table 6, removing user interface-related requirements from the calculations of frequency distributions increases the rank of the business logic category of requirements, making it the forth most frequently-occurring type of requirements in Case 1. This improved the accuracy of the predictions made by the refined domain model as the domain model is now correctly predicting four out of five core requirements in the target system. However, in spite of this better prediction, we consider this improvement over the original model minimal as the business logic category of requirements has only a share of 3.44% of the total number of requirements in the target system, which is not a large enough share to make it a core requirement type. Note that in developing the original domain model the threshold for a requirement class to be considered a core requirement class was set at a share of at least 10% of the total number of requirements. As indicated by the data of Table 6, data outputs, which rank just one place above business logic class of requirements, have a total share of 17.24% of the total number of requirements in the system, creating a whopping 13.8% gap in terms of requirement class size. Nonetheless, event triggers, data inputs, and data outputs are three of the core requirements categories that both the original and the refined model correctly predicted. These three requirements classes were also observed among the core requirements in Case 3. In fact, both the original and the refined domain models correctly predicted the four core classes of requirements in Case 3. In Case 2, event triggers, business logic, and data outputs were the core classes of requirements that were correctly predicted.

Two observations deserve attention in Case 2 and Case 3. First, we noticed that there are no requirements of type data input in Case 2. Second, data validations are among the most frequently-occurring requirements in Case 3. Both of these observations are inconsistent with our previous observations in all the systems we have studied so far. In the majority of systems we have looked at in the past, data inputs have typically been among the core requirement types whereas data validations have been among the least-frequently occurring requirement types, accounting for a small share of requirements in this domain.

To understand the reason for these contrasting observations, we inspected the requirements specification documents for Case 2 and Case 3 and compared them to all the previous systems we had studied. There was a striking difference in terms of the specification style and format; whereas all of the previous cases we had studied had their requirements specified in the form of use case documents, the requirements for Case 2 and Case 3 were specified using proprietary templates. In Case 2, the template for the requirements document included a table for each application screen, with rows for each item on screen and columns for the format and data validation rules for each item. It was precisely this imposed documentation structure that had obliged the requirements engineers to capture a large number of data validation rules. In the absence of such a structure, many of these data validation requirements would remain implicit and undocumented.

In Case 3, the specification of requirements was driven by screen mock-ups. The screen mock-ups were not meant to serve as the final design for the system's user interface; they were only employed as a means to facilitate the capturing of requirements and for illustrative purposes only. Editable user interface components on screen mock-ups were meant to implicitly suggest the data input requirements for each application screen and as such they were not explicitly specified. This style of specification is in contrast to the use case format for requirements specification, where, typically, the textual description of the use case includes one or more steps to capture data input requirements. This explained why there were no statements describing data inputs in Case 3. The absence of data inputs in Case 3 was a by-product of a stylistic choice in the specification of requirements rather than any indication of features that do not require data inputs. If we were to convert the requirements documents for Case 3 into use case format, data inputs could well be among the core requirements types. To summarize, we found that:

- Data output and event trigger categories of requirements have been unanimously observed among the core requirement types in all of the systems we have studied so far.
- Requirements specification style matters. It is an important factor in determining which classes of requirements will be explicitly documented and which classes will potentially be missed or remain implicit or unnoticed.
- The classes of requirements in the domain of enterprise systems can be divided into two categories of core and non-core requirements. Core requirement classes tend to exist in almost all systems in this domain and occur more frequently, though due to specification style and other factors, some core requirement classes might remain unstated and implicit (e.g., they exist in the heads of project stakeholders). Non-core requirements classes, in contrast, are not commonly observed in systems in a domain and even when they occur, they account for a relatively small share of the total number of requirements.
- Based on analyses of 18 industrial cases in the domain of enterprise systems that we have looked at so far, we have identified 9 core requirements classes. These core requirements classes include data output, data input, event trigger, business logic, data persistence, data validation, user interface, user interface logic, and user interface navigation. Non-core requirements classes that we have observed so far include the requirement classes of external behavior, external call, communication, post-condition, and data source.

3.3 Answer to the Third Research Question

Of the 12 classes of requirements in the ontology of the given domain model, the sizes of 6 classes could be correctly estimated within one standard deviation of the mean by the statistical part of the domain model. These six classes include data input, user interface navigation, external call, communication, data validation, and external behavior. For the purposes of this study, we consider a good estimate of a requirement class size one that is within one standard deviation of the mean. With this in mind, the statistical model of the domain, presented in Table 1, predicts that data input class of requirements in applications from the domain of enterprise systems have a share of between 14.14 and 25.01 percent of the total number of requirements in their corresponding specifications. In the main target system (Case 1), we observed that data inputs had a share of 16.81% of requirements, which nicely fits within the predicted range of the domain model. In a similar fashion, the class size for the other 5 requirements classes of user interface navigation, external call, communication, data validation, and external behavior all lie within the predicted range of the domain model.

The statistical part of the refined domain model, on the other hand, were capable of predicting the sizes of the five requirements classes of data output, data validations, communication, external call, and external behaviour. The statistics in the refined domain model predicts that the data output class of requirements in applications from the domain of enterprise systems has a share of between 12.59 and 36.60 percent of the total number of requirements in their corresponding specifications. In the studied system, we observed that data outputs had a share of 17.24% of requirements, which nicely fits within the predicted range of the domain model. In a similar fashion, the class size for the other 4 requirements classes of data validation, communication, external call, and external behaviour all lie within the predicted range of the domain model.

Notice that in both the original and the refined domain model, the size of only one or two core requirements class is predicted with acceptable accuracy. However, both models accurately predict the sizes of multiple non-core requirements classes. Due to variation in applications in a domain as well as the inconsistencies in specification style, it might be difficult, if not impractical, to come up with a statistical model for the domain of enterprise systems that is capable of providing accurate predictions about the sizes of most core requirement classes. On the other hand, our study demonstrated that a comprehensive ontology of core requirement categories for the domain of enterprise systems can be built and, therefore, accurate predictions of requirements classes in the domain of enterprise systems are quite possible.

4 Background and Related Work

There is broad consensus in the software engineering community that software engineering and, in particular, requirements engineering, as knowledge-intensive activities, will benefit from advancements in approaches that provide for more effective knowledge sharing and management. Furthermore, focus on specific domains or application areas allows for the capturing of specialized knowledge that would have otherwise been difficult or led to a less-useful and over-generalized one-size-fit-all solution [12,13,14]. These considerations, among others, have lead to the growing field of domain engineering, which is concerned with the identification, modelling, construction, cataloging, and dissemination of software artifacts that can be applied to existing and future software projects in a particular application domain [29]. Domain analysis, which is a major activity in domain engineering, is concerned with developing a model of the domain [1, 5, 27], which among other forms, can be represented as an ontology.

Ontologies, as explicit formal specifications of shared conceptualizations [3, 16, 32], can be effectively employed as a means to capturing, communicating, and managing knowledge about domains or application areas. An ontology enumerates the concepts relevant in an application area, defining the classes of concepts and the relationships among the concept classes, thereby providing a universe of discourse [26]. An ontology is a theory about a domain [6] and therefore its usefulness can be measured in terms of its descriptive and predictive power. The empirical study reported in this paper was precisely meant to evaluate the predictive power of the given requirements domain model for enterprise systems.

Ontologies have received considerable attention from the research community as a promising way to address numerous current software engineering problems [4, 18, 7]. Ontologies have found wide applications in numerous areas in software engineering such as requirements engineering [20, 25], architecture [19, 33], software comprehension [35], software maintenance [23], software methodologies [15], software cost estimation [17], traceability [28, 30, 37], software modelling [24], and model transformation [21], just to name a few.

The ontology presented and evaluated in this paper shares with all of these previous studies of ontologies the common goal of capturing, as accurately as possible, the knowledge of an aspect of a domain in order to facilitate the software development process. However, the ontology described and evaluated in this paper differs from previous work in three major ways.

First, we augmented our ontological categories with statistical information, which helped us not only to make assertions about what exists in their corresponding domain of discourse (i.e., statements of requirements in the domain of enterprise systems), but also how frequently these ontological categories exist within that domain of discourse. Ontologies can be represented in various formats including textual, diagrammatic, tabular, as well as formal logic. We represented our requirements domain model in tabular format because it is particularly suitable for augmentation with statistical information. The use of statistical information about ontological categories allow for a distinction between core and non-core categories, which, in turn, can have practical implications. None of the previous studies that we are aware of had used statistical information.

Second, the functional requirements taxonomy presented as part of our domain model is both comprehensive and fine-grained. A comprehensive understanding of the problem domain is fundamental to communicate and engineer quality requirements for software-intensive systems [16]. We demonstrated through multiple case studies that the classes of functional requirements in our domain model covered at least 94% of all statements of requirements in our target systems. The important point here, however, is that this high degree of comprehensiveness was not achieved at the expense of overly generalized and all-encompassing categories. On the contrary, the categories of requirements in the domain model were at the level of atomic requirements, which are considered to be the smallest unit of requirements statements [8, 11]. In comparison, most previous ontologies used broadly generalized categories. For instance, in [2], a base requirements ontology is presented, where functional requirements are divided into three broad categories of data specification, process specification, and control specification. The specification of system functions fall under the process specification category. In comparison, in our domain model, functions of an enterprise system are decomposed and described using atomic statements of requirements, each belonging to one of the 14 categories of functional requirements types presented in our model. As another example, in [22], functional requirements are classified under the two broad categories of primary and secondary functional requirements, the difference being that primary functional requirements directly contribute to the goal of the system, whereas secondary functional requirements do not yield direct value to its users.

Functional requirements in the model presented in [22] are classified, along another dimension, into two broad categories of user task and system task based upon how they are realized. The former category includes tasks that are performed by a user of the system, while the latter is performed by the system. Although these broad classification schemes for functional requirements, to some extent, help to organize and structure functional specifications and facilitate the understanding of requirements, they are not specialized enough to effectively guide or drive the subsequent development activities. For instance, although distinguishing between primary and secondary functional requirements at the level of domain ontology can help in prioritizing and planning for requirements, it will not be of much help in the design and implementation of system functions. In contrast, an ontology, like the one presented in this paper, where functional requirements are classified along atomic problem dimensions, such as data input, data validation, business rules, data persistence, and the like, can directly impact and drive the design process. This is evident from the numerous reusable solutions such as APIs, frameworks, regularities [10], and patterns that are aligned along one of these problem dimensions. Data validation frameworks, business rule engines, and data persistence frameworks are prime examples of such reusable solutions.

Finally, our ontology differs from previous ontologies in a third way, namely its domain of discourse. For most previous domain ontologies, the domain of discourse is comprised of all the entities, whether conceptual or real, that appear in the domain. In contrast, the domain ontology presented in this paper, primarily concerns the statements that are made about the problem domain. In other words, while most ontologies are entity-based ontologies, ours is a sentential ontology. Depending upon their intended applications, both types of ontologies are desired. However, a crucial advantage of a sentential ontology over entity-based ontologies is its unifying effect upon various applications within a domain.

The domain of enterprise systems encompass a wide range of applications such as accounting, sales, inventory management, banking, insurance, human resources management, payroll processing, customer relationship management, supply chain management, enterprise resource management, and numerous other applications. An entity-based ontology for the domain of enterprise systems strives to cover entities that appear in all these various application areas. This seems a daunting task for the knowledge engineer who is tasked to develop a domain ontology and may easily lead to over-generalized concepts and categories in the resulting domain model in order to provide a broad coverage of concepts. In comparison, a sentential ontology makes use of the fact that, although there exist a wide variety of concepts in the various applications within a domain, the specifications of these applications are successfully accomplished using statements that belong to a relatively small set of statement types. The key point here is that although, in moving from one application to the next, the set of concepts change significantly, the categories of statements that describe the domain remain fairly unchanged. The dataset we collected and analyzed both during the initial development of the domain model and the evaluation of the domain model, as reported in this paper, provide empirical evidence to this unifying capability of the given sentential ontology; the 14 functional requirements categories, which form the ontological categories in our domain model, were enough to cover all of the requirements statements in the 18 enterprise systems that we have studied thus far.

4 Conclusion and Future Work

As domain models, represented as ontologies, become more prevalent for knowledge sharing and management in the field of software engineering, a rigorous evaluation of their usefulness, for instance in terms of their predictive power, becomes essential. Accordingly, this paper, through an empirical multiple case study of industrial software systems, investigated the predictive power of a given requirements domain model. Results from this evaluation, among others, demonstrated that the given domain model provides a comprehensive ontology of functional requirements categories for applications in the domain of enterprise systems. Furthermore, the calculation of frequency distributions as well as descriptive statistical information for the ontological categories allowed us to distinguish between core and non-core requirements categories. This distinction can have a practical implication for both software engineering practitioners and researchers in terms of attaining a higher return on investment through directing future efforts toward developing more effective tools, techniques, processes, and technologies to better address the core requirements categories in enterprise systems. ***As just an illustrative example of how predictive power of the proposed ontology has practical utility, take the case in our domain model, where it indicates that data output category of functional requirements, on average, accounted for over 22% of the requirements, whereas the communication requirements, on average, had a share of only slightly over 1%. Given this information, it would make sense for a development organization, with time constraints and resource limitations, that specializes in business information system development, to invest its efforts in developing reusable frameworks, tools, patterns, or any other appropriate form of best practices to address and devise a reusable solution for data output category of functional requirements rather than the communication requirements, which, as the model indicates, occur very infrequently and, as a result, afford very few opportunities for solution reuse and consequently increase in productivity.*** The domain model can be effectively employed to inform the decision making process in numerous ways within development organizations.

The work reported in this paper can be continued in several ways. First, further replications of this study with more applications in the domain of enterprise systems will help us to increase our confidence to the findings of this study or possibly challenge some of the findings of the study. In this present study, we only used data based on one complete and two partial requirements sets from three systems, which can a threat to the validity of the study. It is only through a large enough number of replications with variations and the aggregation of substantial evidence that we can build confidence to the usefulness of our models and theories. From this perspective, we view our study as a necessary first step that needs to be continued by the software engineering researchers, especially those focusing on business information systems.

A second avenue to continue the research reported in this paper is to develop and evaluate ontological-statistical requirements models for other domains such as the domain of embedded control systems, scientific simulation systems [31, 34, 36], mobile systems, medical information systems, and others. This current work was concerned with the domain of enterprise systems. However, the research methodology used for the work reported here can be applied to other software domains as well. Such similar studies conducted in other domains will not only help us to gain a better understanding of individual studied domains, but also to identify contrasts and commonalities across domains.

Yet a third direction for future research is to use the ontology presented in this paper, after sufficient validation, as a solid theoretical foundation for developing more effective techniques, tools, processes, and technologies to better support the development of enterprise systems. That is to say, every software engineering approach is based upon a set of assumptions, whether stated or implicit, about the domain of application for which it is intended. It goes without saying, then, that the degree of the effectiveness of a software engineering approach is a function of its underlying assumptions. As a result, an ontology and the predictions it makes about a domain can effectively inform the development of better approaches to the engineering of systems in a particular domain. Taking this view, we are currently developing a requirements engineering process for enterprise systems that uses the domain model presented in this paper as its underlying theoretical framework.

References:

[1] Arango G., "Domain Analysis - From Art to Engineering Discipline," in *Proceedings of the 5th International Workshop on Software Specification and Design*, Los Alamitos, CA, USA, 1989, pp. 152-159.

[2] Assawamekin N. and Sunetnanta T., "Ontology-Based Multiperspective Requirements Traceability Framework," *Knowledge and Information Systems, Springer-Verlog London*, vol. 25, no. 3, pp. 493-522, 2009.

[3] Borst W.N., *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. Berlin, Heidelberg: Doctoral Dissertation, Enschede, NL-Centre for Telematics and Information Technology, University of Tweenty, 2006.

[4] Calero Coral, Ruiz Francisco, and Mario Piattini, *Ontologies for Software Engineering and Software Technology*. Berlin, Heidelberg: Springer, 2006.

[5] Falbo R.A., Guizzardi G., and Duarte K.C., "An Ontological Approach to Domain Engineering," in *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002)*, 2002.

[6] Falbo R.A., Menezes C.S., and Rocha A.R.C., "A Systematic Approach for Building Ontologies," in *Proceedings of the IBERAMIA'98, Lisbon, Portugal*, 1998.

[7] Gašević Dragan, Kaviani Nima, and Milanović Milan, "Ontologies and Software Engineering," *HANDBOOK ON ONTOLOGIES, International Handbooks on Information Systems*, vol. 5, pp. 593-615, 2009.

[8] Ghazarian A., "A Formal Scheme for Systematic Translation of Software Requirements to Source Code," in *Proceedings of WSEAS Applied Computing Conference (ACC 2011)*, Angers, France, 2011, pp. 44-49.

[9] Ghazarian A., "Characterization of Functional Software Requirements Space: The Law of Requirements Taxonomic Growth," in *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE'2012)*, Chicago, 2012.

[10] Ghazarian A., "Coordinated software development: A framework for reasoning about trace links in software systems," in *Proceedings of the 13th International Conference on Intelligent Engineering Systems (2009)*, 2009, pp. 39-44.

[11] Ghazarian A., Tehrani M.S., and Ghazarian A., "A Software Requirements Specification Framework for Objective Pattern Recognition: A Set-Theoretic Classification Approach," in *Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems (CECCS 2011)*, 2011, pp. 211-220.

[12] Ghazarian, A., “A Domain-Specific Architectural Foundation for Engineering of Numerical Software Systems”, *WSEAS Transactions on Systems*, No 7, Vol. 10, pp. 193-208, World Scientific and Engineering Academy and Society, July 2011.

[13] Ghazarian, A., “Requirements Engineering for Business Information Systems: A Dimension-Oriented Approach”, *WSEAS Transactions on Systems*, In Press, World Scientific and Engineering Academy and Society, 2013.

[14] Ghazarian, A., “The Impact of Architecture in Engineering of Software Systems”, *WSEAS Transactions on Systems*, In Press, World Scientific and Engineering Academy and Society, 2013.

[15] Gonzalez C. and Henderson-Sellers B., "An Ontology for Software Development Methodologies and Endeavors," in *Ontologies for SOftware Engineering and Software Technology*, 2006, pp. 123-151.

[16] Gruber T.R., "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, 1993.

[17] Hamdan K. and Khatib H.E., "A Software Cost Ontology System for Assisting Estimation of Software Project Effort for Use With Case Based Reasoning," in *Innovations in Information Technology*, 2006, pp. 1-5.

[18] Happel H.J. and Seedorf S., "Applications of Ontologies in Software Engineering," in *Proceedings of the International Workshop on Semantic Web Enabled Software Engineering*, 2006.

[19] Hayes-Roth F., "Architecture-Based Acquisition and development of Software: Guidelines for Recommendations from the ARPA Domain-Specific SOftware Architecture (DSSA) Program," *Technical Report, Teknowledge Federal Systems, Palo Alto, CA*, 1994.

[20] Kaiya H. and Saeki M., "Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach," in *Proceedings of the 5th International Conference on Quality SOftware (QSIC 2005)*, 2005, pp. 223-230.

[21] Kappel G. et al., "Lifting Metamodels to Ontologies: A Step the Semantic Integration of Modeling Languages," in *Proceedings of the ACM/IEEE 9th International Conference on Model Driven*

Engineering Languages and Systems, 2006, pp. 528-542.

[22] Kassab M., Ormandjieva O., and Daneva M., "An Ontology Based Approach to Non-Functional Requirements Conceptualization," in *Proceedings of the 4th IEEE International Conference on Software Engineering Advances*, 2009, pp. 299-308.

[23] Kiefer C., Bernstein A., and Tappolet J., "Analyzing Software With iSPARQL," in *Proceedings of the 3rd ESWC International Workshop on Semantic Web Enabled Software*, 2007.

[24] Knublauch H., "Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protege/OWL," in *Proceedings of the 1st International Workshop on the Model-Driven Semantic Web*, 2004.

[25] Lee Seok Won and Gandhi Robin A., "Ontology-Based Active Requirements Engineering Framework," in *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, 2005.

[26] Musen M.A., "Domain Ontologies in Software Engineering: Use of Protege With the EON Architecture," *Methods of Information in Medicine*, vol. 37, no. 4-5, pp. 540-550, 1998.

[27] Neri, F., Learning and Predicting Financial Time Series by Combining Evolutionary Computation and Agent Simulation, Applications of Evolutionary Computation, EvoApplications, LNCS 6625, pp. 111–119, Springer, Heidelberg (2011).

[28] Noll R.P. and Ribeiro M.B., "Enhancing Traceability Using Ontologies," in *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC 2007)*, Seoul, Korea, 2007, pp. 1496-1497.

[29] Pressman Roger S., *Software engineering A practitioner's approach*, 5th ed. New york, USA: Mc Graw-Hill, 2000.

[30] R.P. Noll and M.B. Ribeiro, "Ontological Traceability Over The Unified Process," in *Proceedings of the 14th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2007)*, 2007, pp. 249-255.

[31] L. Shuang, W. Zhixin, W. Guoqiang, A Feedback Linearization Based Control Strategy for VSC-HVDC Transmission Converters, *WSEAS Transactions on Systems*, Issue 2, Volume 10, pp. 49-58, February 2011.

[32] Studer R., Benjamins V.R., and Fensel D., "Knowledge Engineering: Principles and Methods," *Data Knowledge Engineering}*, vol. 25, no. 1-2, pp. 161-197, 1998.

[33] Tetlow P. et al., "Ontology-Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering," in *W3C Working Draft.*, 2006.

[34] Tsay, T-S, Intelligent Guidance and Control Laws for an Autonomous Underwater Vehicle,*WSEAS Transactions on Systems,* Issue 5, Volume 9, pp. 463-475, May 2010.

[35] Witte R., Zhang Y., and Rilling J., "Empowering Software Maintenance With Semantic Web Technologies," in *Proceedings of the 4th European Semantic Web Conference*, 2007, pp. 37-52.

[36] Xu L., Han Y., Khan M. M., Zhou L.,Yao G., Chen C., Pan J., A Novel Control Strategy for Dynamic Voltage Restorer using Decoupled Multiple Reference Frame PLL (DMRF-PLL), *WSEAS Transactions on Systems*, Issue 2, Volume 8, pp. 261-277, February 2009.

[37] Zhang Y., Witte R., Rilling J., and Haarslev V., "An Ontology-Based Approach for Traceability Recovery," in *Proceedings of the 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006))*, 2006, pp. 36-43.