International Journal of Software Engineering
and Knowledge Engineering
Vol. 24, No. 3 (2014) 329–345
© World Scientific Publishing Company
DOI: 10.1142/S0218194014500132

Ontology for Systems Development

Niklas Hallberg*,†,‡ and Erland Jungert*,§

*The Swedish Defence Research Agency (FOI)
P. O. Box 1165, S-582 26, Linköping, Sweden

†School of Computer Science and Communication
KTH Royal Institute of Technology
SE-100 44 Stockholm, Sweden
‡nikha@foi.se
§jungert@foi.se

Sofie Pilemalm

Linköping University
SE-581 83 Linköping, Sweden
sofie.pilemalm@liu.se

Received 30 April 2010
Revised 26 March 2013
Accepted 6 December 2013

Software-intensive information systems have a major impact on our lives, both privately and professionally. Development of these systems is a complex activity that requires the involvement of people with different competences and skills. Even though software-intensive systems have been developed since the 1960s, the success rate is still low. A major hindrance to successful system development projects is the lack of consistent terminology. Since systems development is a collaborative activity, involving not only systems developers but also domain experts and user representatives, the understanding of each other is a prerequisite for an effective collaboration. The aim of this paper is to explore and present definitions, dependencies, and relationships of the most fundamental concepts in systems development in the form of an ontology. The ontology consists of four categories of concepts: General concepts, Description concepts, Realization concepts, and Appearance concepts. The two core concepts in the ontology are Systems and Systems development.

Keywords: Terminology; ontology; systems development.

1. Introduction

The area of systems development targets the creation of systems, either by the development of completely new systems or by the modification of existing ones. The difficulty in developing systems on budget and on time has been acknowledged for a long time, but the problems have, so far, not been entirely resolved [1].

Large resources are spent on the correction of mistakes made during the development. Deming [2] expressed it as “Defects are not free. Somebody makes them, and gets paid for making them” and stated that quality in products comes from improvements of the development process. Systems development is a complex activity that includes, e.g. (1) understanding the context of use and need for changes, (2) elicitation and determination of functional and non-functional requirements, (3) selection of the best design and technical realizations, (4) ensuring that human interaction and IT-security aspects are covered, and (5) integration of the systems in the context of use [3]. The diversity of activities in systems development requires that several competences from different knowledge domains are considered and that the representatives from these domains collaborate, including user representatives [3]. An obstacle for efficient collaboration has been perceived as the lack of understanding of and respect for other disciplines [4]. An additional, but related, problem is that the meaning of a given concept might differ in different disciplines. Consequently, the same concept can be used with different meanings and different concepts can be used for the same phenomena, e.g. concepts such as requirements and usability [5, 6]. Requirements is the concept in systems development that is used with the most ambiguous meaning, which causes problems [7]. Further, user representatives preferably express themselves in terms of experienced problems, requests, and solutions they believe would be beneficial to the project in focus. Meanwhile, the professional developers ask the users for their needs and requirements in a structured format. If original user statements are not correctly translated into requirements, this can be a reason for flaws in the specifications of the system [8]. Hence, there is no unambiguous and comprehensive use of concepts in the field of systems development. This causes misunderstandings, misinterpretations, and irritations in the development of systems, in the most severe cases inhibiting the functioning and usability of the emerging system [9–11].

An ontology is defined as an explicit specification of a conceptualization and it is used to define concepts and their relations [12]. Hence, the concepts of interest in the field (e.g. systems development) can be organized as an ontology. The ontology can be used to establish and communicate the meaning of the conceptualized terms to obtain clarity among the actors involved. An ontology should be coherent, which means that the defined set of concepts should be consistent. Further, it should be extendable, which means that it should be possible for stakeholders to define and include new terms and concepts. Extendibility is of particular importance in systems development where the terminology tends to grow fast and sometimes become overwhelming.

In this paper, we propose an ontology for systems development. The ontology is intended to be used as a basis for the terminology applicable in systems development projects. However, since the conditions for systems development projects vary it may be necessary to perform some adaptation of the ontology for specific projects. Still, by using such a predefined ontology that defines the concepts used, it is possible to reduce the number of misinterpretations within projects. Further, no project has to generate its own terminology, which enhances the possibility to reuse definitions of concepts in several categories of projects. Hence, the objective of the paper is an ontology for systems development concepts, which is consistent and coherent. This is achieved by exploring existing definitions and suggesting new definitions on some of the significant concepts related to systems development, describing dependencies and relations between these concepts.

This paper has the following outline. Section 1 provides a background and a motivation for the rest of the paper. Section 2 describes briefly how the study was carried out. Section 3 presents the systems-development concepts in the form of an ontology. Section 4 presents how the systems-development concepts are related to each other. Section 5 describes the efforts to develop ontologies and taxonomies of words related to the field of systems development. The last section discusses the main contribution of this paper, how we attempt to use the ontology further and suggestions for future work.

2. Methods

The work to develop an ontology for systems development was performed in three steps. In the first step, a literature study was carried out to identify significant and commonly used concepts and their definitions within systems development. The study was performed based on systems and software engineering standards, books, scientific papers, and articles. Thereafter, the identified concepts were categorized together with similar concepts and relationships between the concepts were determined. The outcome of the categorization was used to construct an initial ontology.

In the second step, an extended literature study was performed. The outcome of the study was used to augment the ontology, by introducing concepts to fill the identified gaps, resolving contradicting definitions, reducing ambiguity in the meaning of concepts, and eliminating overlaps among the concepts. For concepts with several definitions, the definition most commonly used and suitable for the aim, i.e. to obtain a consistent and coherent ontology, was selected.

In the third step, the ontology was reviewed by two senior systems engineers. The two systems engineers individually inspected the ontology to find uncertainties and ambiguities, and to recommend improvements to resolve these issues. Based on the suggestions of the systems engineers, the ontology was revised and finalized.

3. The Systems Development Concept Ontology

In this section we propose an ontology for systems development. In the ontology, the main concepts related to the systems development process are divided into four categories; (1) *General concept*, (2) *Description concept*, (3) *Realization concept*, and (4) *Appearance concept* (Fig. 1).

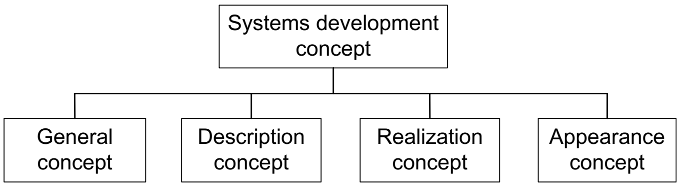

Fig. 1. The top level of the ontology for systems development concepts consists of four branches: General concept, Description concept, Realization concept, and Appearance concept.

3.1. *General concepts*

The general concepts branch includes three sub-branches (1) *System structure*, (2) *System production*, and (3) *System role* (Fig. 2). The System structure branch exhibits terms that on high abstraction and logical levels describe a structure of systems including also the term system itself; examples are *System, View*, *Architecture*,

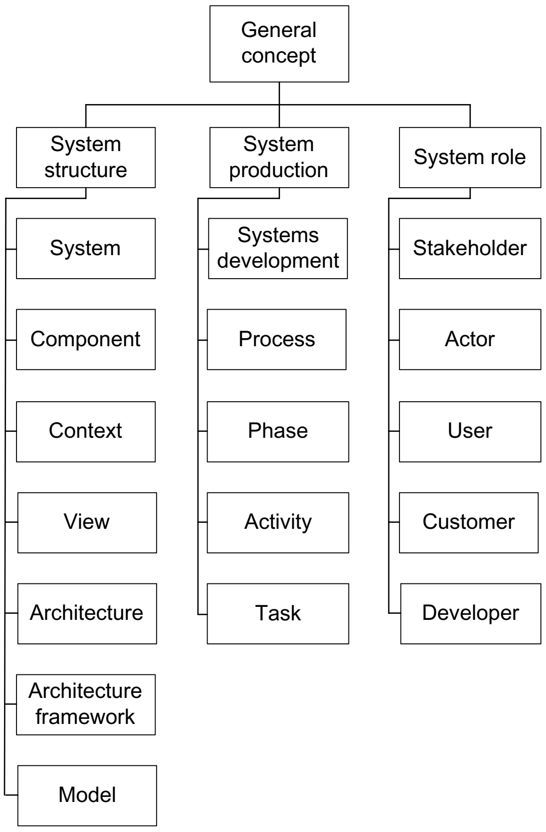

Fig. 2. The General concept branch including the sub-branches System structure, System production, and System role.

and *Model*. System production constitutes terms related to the process of producing systems. It includes the concepts *Systems development*, *Process*, *Phase*, *Activity*, and *Task*. System role concerns the individuals involved in the development process and includes *Stakeholder*, *Actor*, *User* and *Customer*.

3.1.1. *System structure concepts*

System structure concepts are concepts that on a general level are used to describe systems. The branch of the ontology includes *System*, *Component*, *Context*, *View*, *Architecture*, *Architecture framework*, and *Model*.

A *System* is defined as a collection of components organized to accomplish a specific function or a set of functions [13, 14]. Systems can be technical, human, organizational or a combination of those. Further, systems consist of components and exist in a context. A distinct and clarified border between the system and its context is essential in systems development [3].

A *Component* is one of the parts that make up a system. It can in itself be a system or an indivisible part [14]. In the literature, the terms *module*, *units*, and *elements* are used with the same meaning [13].

Context of use contains the users, tasks, equipment (hardware, software, and materials) and the physical and social environment in which a product is used [15]. Hence, it is the environment in which a system will operate or operates, which contains elements that the system interacts with [3].

A *View* is a representation of a system from the perspective of a related set of concerns [16]. Views are used in *architectures* and *architecture frameworks* to make it possible to understand complex systems, by leaving out unnecessary information.

Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [16]. Architectures are used to represent systems, regarding components with descriptions of how they fit together, their interrelations, and the purposes of their combined parts. Further, they specify rules for how components in systems should interact. The *enterprise architecture* is a coherent whole of an enterprise, as it holds information of all possible things of importance to the enterprise, like its organization, objectives, business tasks, activities, relations, and technological infrastructures [17].

An *Architecture framework* establishes the practice for creating, interpreting, analyzing, and using architecture descriptions. Thereby, it is used to guide the development of architectures used for specific applications, as different frameworks concentrate on and stress different, e.g. scopes, intentions and domains [18]. Hence, a *framework* is a basic conceptual structure, which hosts, e.g. models, tools, and methods.

A *Model* is an abstract description of a phenomenon [19]. The phenomenon can be real or virtual. Models are used to describe structures, activity flows, relationships, and information flows [20]. A *business model* is an abstraction used to support the key understanding of a business [21]. Thereby, it provides a measure for analysis and for communication of business related matters [22]. A *prototype* is a model that is used to illustrate the design and functionality of, e.g., information systems [3].

3.1.2. *System production concepts*

System production concepts include the general concepts related to the development of systems including *Systems development*, *Process*, *Phase*, *Activity*, and *Task*.

Systems development is the process carried out to develop systems. Several approaches exist that suggest how systems development ought to be performed, for instance the Waterfall, the Spiral models, and the Rational Unified Process (RUP) [3].

A *Process* is a sequence of activities designed to produce a specified output [14]. The Waterfall process, introduced by Royce in 1970, was one of the first attempts to describe software developments as a flow of activities [23].

A *Phase* is part of a process, i.e. a process can be divided into several phases [14]. Examples of phases in systems development are project initiation and planning, analysis and design phases. Hence, a phase consists of a sub-set of activities to be performed during systems development.

An *Activity* represents a set of tasks that consume time and resources and whose performance is necessary to achieve, or contribute to the realization of one or more outcomes [24]. Examples of activities are stakeholder analyses and needs analyses.

A *Task* is a requirement, recommendation, or permissible action, intended to contribute to the achievement of one or more outcomes of a process [13]. An *action* is the atomic building block that edifies the activities, which accepts inputs and produces outputs [25].

3.1.3. *System role concepts*

The *System role* concepts describe different roles that are related to systems development including *Stakeholder*, *Actor*, *User*, *Customer*, and *Developer* [24].

A *Stakeholder* is an individual or a group of individuals who are affected by, or able to affect the system [3]. They have the right, share or claim in a system [13]. Hence, stakeholders include users, actors, customers, and developers.

An *Actor* is someone or something, outside the system that interacts with the system [26]. Actors can interact intentionally or unintentionally, with or without a goal.

A *User* is an individual or individuals that intentionally operate or interact with the system [27]. Primary or direct users interact directly with the system, while secondary users or indirect users interact with the system via direct users [28].

A *Customer* is an individual or organization that directly purchases or has a strong influence on the decisions on purchase systems. *Key customers* are the customers that have the strongest influence on the decision to purchase the systems [29].

A *Developer* is an individual or organization that performs the development of systems but also performs or participates in this process [14]. This role can be divided into several roles, e.g., systems architect, systems engineer, requirements engineer, interface designer, and user representatives.

3.2. *Description concepts*

The *Description concepts* describe the system from different perspectives during the different stages of development including *Statement*, *Need*, *Requirement*, *Design*, and *Solution* (Fig. 3).

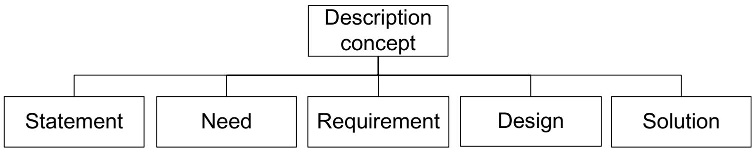

Fig. 3. The Description concept branch of the systems development ontology.

A *Statement* is an expression that contains information relevant to the development of the system, which may consist of problem descriptions and ideas for future solutions [30]. Statements are expressed by stakeholders as observations, assumptions, and remarks of interest. They can include descriptions about the stakeholders, the actual or wanted situations of use, perceived problems, business goals, and visions [31].

A *Need* is a condition or capability needed by a user to solve a problem or achieve an objective. Hence, it is something required, desirable, or useful, preferably expressed in the language of the users or customers [30, 32]. The term *need* is frequently used in standards, but not explicitly defined, e.g. [14]. Further, needs could be explicitly expressed or only implied [33].

A *Requirement* specifies *what* systems should accomplish [34]. Requirements are divided into *functional requirements* and *non-functional requirements* [35]. Functional requirements specify what systems should perform while non-functional requirements specify what abilities systems should possess in order to perform well. For instance, the system shall provide communication facilities and the system shall function in sub-tropical areas.

Design specifies *how* functionality and features in the system should be implemented and realized [36]. That is, the design specifies how the requirements should be fulfilled. It describes both visible features such as the graphical interface as well as invisible features such as database structures.

A *Solution* is the description of a system or a component that realizes the design, which means that it should meet both the requirements and the identified needs. The solution is the produced system, which may be technical (e.g. IT systems), organizational (e.g. new working groups), human competence (e.g. recruit personnel with specified skills), and combinations thereof [36].

3.3. *Realization concepts*

The *Realization concepts* include activities performed during the systems development process (Fig. 4). These activities include *Context analysis*, *Stakeholder analysis*, *Needs Analysis*, *Requirements engineering*, *Design*, *Implementation*, *Deployment*, *Verification*, and *Validation.*

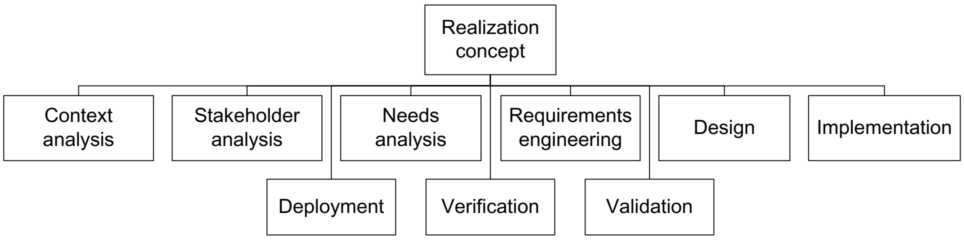

Fig. 4. The Realization concept branch with its sub-branches relating to different system development activities.

Context analysis is the activity which examines the context in which the eventually developed system should be used. If the development is to modify an existing system, the system itself should be scrutinized. A form of Context analysis is *Business analysis*, which is carried out to get an understanding of the organizational context of the information system and to provide a foundation for changes in the organizations [24]. The outcome of a Context analysis and a Business analysis is generally a business model. These models can be used as input into stakeholder analyses and needs analyses [5].

Stakeholder analysis is the activity to identify and assess the importance of individuals, groups of individuals, and organizations that may significantly influence the development and shaping of the system [37].

Needs Analysis is the activity to determine the needs that are the foundation for the systems development [8]. Needs analysis requires insight to the system context. The outcome of a needs analysis is a specification of the identified needs.

Requirements engineering is the activity to specify what the systems should accomplish without saying how [35]. That is, to translate needs into requirements. The outcome of requirements engineering is a specification of the identified requirements.

Design is the activity to specify how requirements should be realized by the system [38]. This includes, for instance, how the user interface should look, how user interaction should be experienced, and what technologies should be used.

Implementation is the activity to realize the design; the outcome of this process is the corresponding product. It can be performed in several different manners depending on what kind of system is targeted. For instance, software systems can be coded, organizations can be reorganized and new competences can be recruited [39].

Deployment is the activity to integrate the developed system into its context, for instance, when the new information system is integrated into the organization [40].

Verification is the activity to confirm that the specified requirements have been fulfilled by an objective review of the design or system [33]. Hence, verification means to determine in the "contract" (i.e. requirements specification) that which is to be fulfilled [19].

Validation is the activity to confirm that the intended usage has been fulfilled by the requirements, the design, or the system [33]. Hence, validation means to explore whether stakeholders gain the intended utility [19].

3.4. *Appearance concept*

The *Appearance concept* concerns the systems' external appearances, i.e. how the systems look and behave from the point of view of the stakeholders, for instance the users (Fig. 5). These concepts include *Function*, *Features*, *Capability*, *Capacity*, *Effect*, and *Service*.

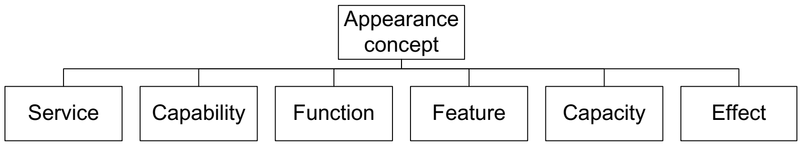

Fig. 5. The Appearance concept branch with its six sub-branches relating to the system external appearance.

Function describes an action that a system performs [30]. *Functionality* is the set of functions that a system provides. For instance, an operating system provides the functionality of executing software programs and handling the storage and retrieval of data files.

Feature describes the distinguishing characteristic of a system [41]. That includes its structure, form, performance, and appearance.

Capability describes the ability that a system has to provide, specifying functions during specified circumstances. Even though this term is used in the literature, it is not explicitly defined [42].

Capacity describes quantitative features and qualitative features of specified functions that systems can provide. Quantitative features can, e.g., be provided as units per time.

Effect describes the results caused by systems; that is, the results of the functions affecting the systems. For instance, higher turnover and shorter production cycles are effects that a system can provide.

Service is work done by a service provider to achieve desired results for a service consumer [43]. Further, it is an abstraction of how a system (service producer) can accomplish usage for other systems (service consumers) without expressing how this is done [44]. The definition of services is independent of how they are implemented, i.e. independent of which systems deliver the effect.

4. Relationships

The core concepts of the ontology are System and Systems development (belonging to the General concepts branch). The system exists in a system context and is constituted by system components (Fig. 6). The design and architecture are models of the system. The design can be of various levels of detail, meanwhile the architecture presents the overall structure of the system. The architecture framework is a tool for the development of architectures, which prescribes a set of system views. In the architecture, a system view represents the system through a set of concerns.

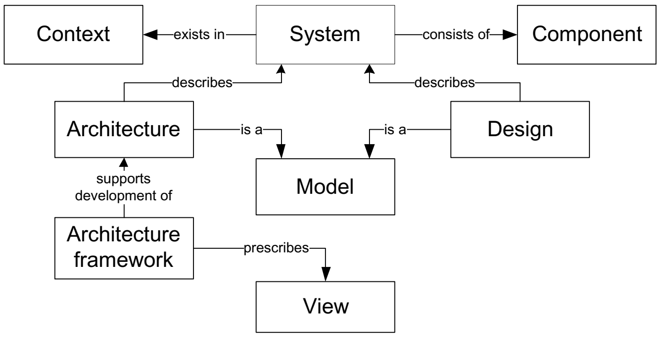

Fig. 6. System structure concepts and how they relate to each other. System is a core concept in the ontology. It belongs to the General concepts branch.

Systems can be described regarding how they appear seen from the outside (Fig. 7). Systems provide functionalities, which have effects. It is those effects that provide the utility for the stakeholders. The systems also have features, which for instance might make them more or less easy to interact with or offering different degrees of possibility to make use of them during different circumstances. Some of the features are closely related to how and when the systems can provide their functionality. Thereby, it is the functionality and features of the system that determine its capabilities. Hence, the system has capability to provide functionality; to a certain

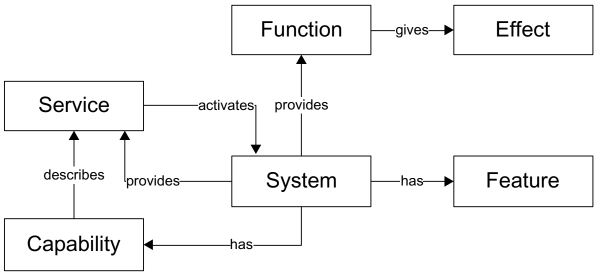

Fig. 7. The General concept System has close relations to the Appearance Concepts. This figure displays the most vital relationships.

amount and during specified circumstances. The capability of the systems can be visible and accessible through services. Thereby, service provides an interface to access the capability of the systems. The stakeholder is the all-embracing system role, which includes actors, users, customers, and developers (Fig. 8). Actors and users are closely related since both categories interact with the system.

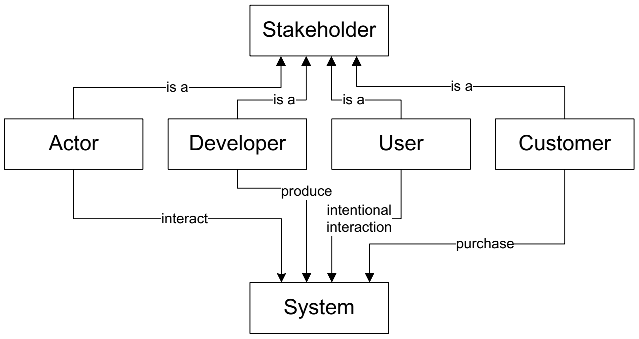

Fig. 8. The Role concept involves different members and how they relate to the System concept respectively. The overall role is defined as stakeholder. A stakeholder can be an actor, a developer, a customer, a user, or a combination of those.

Systems development is the all-embracing concept in the Systems production branch of the ontology (Fig. 9). It is the process aimed at producing. This process consists of activities, which can be broken down into tasks. The process of developing systems can be divided into different phases, e.g. the specification phase and the realization phase.

Conceptually, systems development can be seen as a sequential process, where each activity produces information that constitutes input for the next activity. The

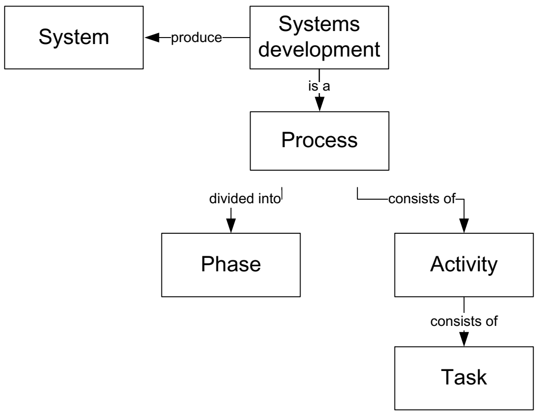

Fig. 9. The General sub-concepts relating to Systems development. The concepts also relate to System, since they aim at producing a system.

process starts with the *Context analysis* aimed at exploring the context in which the system will be operational. The second activity, the *Stakeholder analysis* is meant to identify the stakeholders and to determine which stakeholders' opinions should be taken into consideration in the development of the system. The third activity, *Needs analysis* is aimed to determine and specify the needs of those stakeholders that have been selected in the previous activity. The fourth activity, the *Requirements specification*, is intended to determine and specify the requirements, based on the identified needs. The sixth activity, *Design specification*, is aimed at and based on the requirements to develop a design, i.e. a model, of the system.

The seventh activity, *Implementation*, is aimed at realizing the design, to produce a system on the given design specifications. The last activity in the sequence, *Deployment*, is aimed at incorporating the systems in their operational context. In practice, several possible and different means of iterating activities are necessary. Iterations may imply returning to a previous activity or looping the whole chain of activities. In incremental development, additional functionality and features are gradually incorporated into the system. Hence, the development process is passed through several times to gradually extend the system.

Besides this development process flow, there are two additional activities; Validation and Verification (Fig. 10). The *Validation* activity can be performed on the output of Requirements specification, Design and Implementation and is intended to determine if those outputs correspond to what is needed. The *Verification* activity can be performed on the output from the Design and the Implementation to determine if those outputs meet the specified requirements.

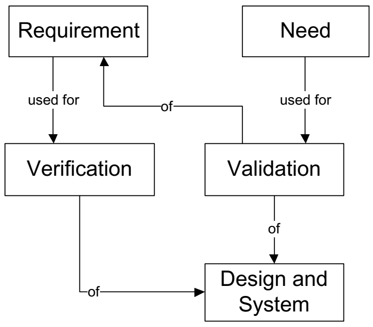

Fig. 10. The system evaluation step of the systems development process, including verification and validation and how they relate to different Description concepts.

5. Related Works

Misunderstandings are a major source of disturbances in systems developments [9, 10]. Thereby, most standards, books, and several papers within the field of software and systems engineering provide sets of definitions of concepts; commonly, in the form of a list of defined terms. For instance, Brikkemper [45] suggested a framework for the engineering of methods for information systems development and tools. To enhance the scientific communication regarding method engineering the framework provides definitions of five terms: *Method*, *Technique*, *Tool*, *Methodology of information systems*, and *Method engineering*.

The standard ISO/IEC/IEEE 24765:2010 provides an extensive standardized glossary of software engineering terminology [14]. However, this standard also points at the problem by providing, e.g. four definitions of requirements, six of design and seven of system. As stated by Honour and Valerdi [11], there is still, unfortunately, no coherent use of a common terminology within widely-used "standards" and there is a need for a common ontology of systems engineering concepts.

Several taxonomies and ontologies have been proposed in domains within and closely related to systems development. Honour and Valerdi [11] presents an ontology with broad definitions of key concepts aimed to judge the success of systems engineering efforts. Their ontology includes the terms Systems engineering effort, Amount of effort, Type of effort, Quality, Success, and Optimum. The ontology is based on a review of, e.g. systems engineering standards. Further, Chikofsky and Cross [46] provide a framework for understanding techniques for reverse engineering. For the framework, they define the concepts of *Forward engineering* and *Reverse engineering*. When it comes to reverse engineering, activities such as *Redocumentation*, *Design recovery*, *Restructuring*, and *Reengineering* are defined. Blanchard [30] presents several models that describe how concepts and terms relate to each other. For instance, models of concepts for *Decompositions of systems*, *Total systems value*, and *Design requirements*.

Ducasse and Pollet [47] present a process-oriented taxonomy for software architecture recovery, where the main categories of concepts are *Goals*, *Processes*, *Inputs*, *Techniques* and *Outputs*. Feiler and Humphery [9] provide extensive definitions of core concepts related to software engineering; the categories comprise *Software process context*, *Software process*, and domain specific use of process concepts. To some extent they also describe how the concepts relate to each other. Their focus is on the engineering of the software development process, rather than on the systems to be developed. Oliver, Andary, and Frisch [48] present a thorough ontology defining several concepts related to systems and systems engineering. The focus of the ontology is on the concepts: *System*, *Requirements and trades*, *System structure and emergent property*, *Function, and Behavior*. Their ontology embraces *Stakeholders* and *their needs*.

The Method Framework for Engineering System Architectures (MFESA) is an extensive meta-method for development of architecture engineering methods [49]. MFESA consists of a comprehensive ontology of concepts and terminology related to system architecture engineering. The MFESA ontology covers 41 concepts and their relations. The main concepts include *System*, *System architecture*, *Architectural decisions*, and *Architectural risks*. Ruijven [50] has developed an ontology for systems engineering performed according to IEEE 15288 [13]. The ontology is based on a number of information models, e.g. models for physical system elements and interaction with the environment.

In the field of computer security several taxonomies exist. For instance, Harrison and White [51] present a taxonomy for classifying cyber events affecting communities. They hope that the taxonomy will support governments and industry to communicate about IT security affecting communities and critical infrastructures. Further, Avizienis, Laprie, Randell, and Landwehr [52] provide a taxonomy of dependable and secure computing concepts. Their aim is also primarily to enhance communication and collaborations regarding system failures and causes of system failures.

Several comprehensive attempts and efforts have been performed in order to define the terms and concepts used in fields related to systems development. However, to our knowledge there exists no ontology that unambiguously defines the concepts used in systems development and that extensively covers the approach presented in this paper.

6. Conclusion

This paper outlines an ontology of core concepts used in systems development. The absence of a widely accepted and consistent terminology in systems development creates a risk of misinterpretations and misunderstandings in the communication within system development projects. Further, it brings confusion to customers and users confronted with inconsistent use of concepts in their discussions with developers. For instance, it is not uncommon for developers to talk about *requirements* when actually intending *needs* and *design*. Additionally, several ontologies and taxonomies related to systems development exist. Some of those are presented in the related works. However, to the knowledge of the authors, no ontology exists that unambiguously defines and relates the core concepts of systems development. For these reasons, this extensive ontology has been developed.

We attempt to use the ontology as a baseline, defining the core concepts, and extending it with project specific concepts. It is our ambition that the ontology should lead to a more consistent use of language in development projects, thereby avoiding flaws due to misconceptions and lengthy discussions about the meaning of different concepts. We believe, further, that the ontology can contribute to increased quality in project generated documentation.

The future work related to the proposed ontology includes further elaborations of the relationships between the concepts included and empirical evaluations in different systems development settings. Although the terminology and the ontology need to be further elaborated, the work presented in this paper should be seen as an initial step towards reaching consensus of the meaning of central concepts in systems development. Thus, this paper outlines an attempt for a consistent and coherent terminology embracing central concepts used in systems development.

Acknowledgments

This work was supported by the FOI research project "Architecture based development of command and control systems," which is funded by the R&D program of the Swedish Armed Forces.

References

1. Y. K. Dwivedi, K. Ravichandran, M. D. Williams, S. Miller, B. Lal, G. V. Antony and M. Kartik, IS/IT project failures: A review of the extant literature for deriving a taxonomy of failure factors, in *Grand Successes and Failures in IT: Public and Private Sectors*, eds. Y. K. Dwivedi, H. Z. Henriksen, D. Wastell and R. De (Springer, 2013), pp. 73–88.
2. W. E. Deming, *Out of the Crisis* (Cambridge University Press, Cambridge, 1986).
3. I. Sommerville, *Software Engineering* (Addison-Wesley, Harlow, 2001).
4. D. Mankin, S. Cohen and T. Bikson, *Team and Technology* (Harvard Business School Press, Boston, 1996).
5. A. Seffah and E. Metzker, The obstacles and myths of usability and software engineering, *Commun. ACM* **47**(12) (2004) 71–76.
6. A. M. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing* (Dorset House Publishing, New York, 2005).
7. H. Kaindl and D. Svetinovic, On confusion between requirements and their representations. *Requirements Engineering* **15**(3) (2010) 307–311.
8. N. Hallberg, S. Pilemalm and T. Timpka, Quality driven requirements engineering for development of crisis management systems, *International Journal of Information Systems for Crisis Response and Management* **4**(2) (2012) 35–52.
9. P. H. Feiler and W. S. Humphrey, Software process development and enactment: Concepts and definitions, in *Proc. Second International. Conference on Continuous Software Process Improvement*, 1993, pp. 28–40.
10. H. Enquist and N. Makrygiannis, Understanding misunderstandings [in complex information systems development], in *Proc. 31st Hawaii International Conference on System Sciences*, 1998, pp. 83–92.
11. E. C. Honour and R. Valerdi, Advancing an ontology for systems engineering to allow consistent measurement, in *Proc. Conference on Systems Engineering Research*, Los Angeles, CA, 2006.
12. M. Uschold and M. Grüniger, Ontologies: Principles, methods and applications, *Knowledge Engineering Review* **11**(2) (1996) 93–155.
13. ISO/IEC 15288, System Engineering — System Life Cycle Processes (International Organization for Standardization/International Electrotechnical Commission, Genève, 2002).
14. ISO/IEC/IEEE 24765:2010, Systems and software engineering — Vocabulary (The Institute of Electrical and Electronics Engineers, New York, 2010).
15. ISO 9241-11, Ergonomic requirements for office work with visual display terminals (Geneva, International Organization for Standardization, 1998).
16. IEEE 1471, Architecture Working Group, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (The Institute of Electrical and Electronics Engineers, New York, 2000).
17. M. Lankhorst, *Enterprise Architecture at Work: Modelling, Communication and Analysis* (Springer, Heidelberg 2009).
18. D. G. Firesmith, P. Capell, D. Falkenthal, C. B. Hammorns, D. T. Latimer and T. Merendino, *The Method Framework for Engineering System Architectures* (CRC Press, Boca Raton, 2009).

19. D. M. Buede. *The Engineering Design of Systems: Models and Methods* (Wiley, Hoboken, 2009).
20. D. C. Hay, *Requirements Analysis: From Business Views to Architecture* (Prentice Hall, Upper Saddle River, 2003).
21. T. Morgan, *Business Rules and Information Systems: Aligning IT with Business Goals* (Addison-Wesley, Boston, 2002).
22. I. Jacobson, M. Ericsson and A. Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology* (Addison-Wesley, Reading, 1995).
23. W. W. Royce, Managing the Development of Large Software Systems, in *Proc. IEEE WESCON*, 1970, pp. 1–9.
24. P. H. Kruchten, *The Rational Unified Process. An Introduction* (Addison-Wesley, Boston, 2004).
25. S. Friedenthal, A. Moore and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language* (Morgan Kaufmann, Amsterdam, 2008).
26. D. Kulak and E. Guiney, *Use Cases: Requirements in Context* (Addison-Wesley, Upper Saddle River, 2003).
27. IEEE 830. IEEE Recommended Practice for Software Requirements Specifications (The Institute of Electrical and Electronics Engineers, New York, 1998).
28. N. Hallberg, S. Pilemalm and T. Timpka, Participatory design of inter-organizational systems: A method approach, in *Proc. Fifth Biennial Participatory Design Conference*, 1998, pp. 129–136.
29. L. Cohen, *Quality Function Deployment: How to Make QFD Work for You* (Addison-Wesley, New York, 1995).
30. B. S. Blanchard, *System Engineering Management* (Wiley, Hoboken, 2008).
31. N. Hallberg and J. Hallberg, The Usage-Centric Security Requirements Engineering (USeR) Method, in *Proc. 7th IEEE Workshop on Information Assurance*, U.S. Military Academy, West Point, NY, 2006, pp 34–41.
32. A. Hari, J. E. Kasser and M. P. Weiss, How lessons learned from using QFD led to the evolution of a process for creating quality requirements for complex systems, *Systems Engineering* **10**(1) (2007) 45–63.
33. ISO/IEC 25030, Software Engineering — Software Product Quality Requirements and Evaluation (SQUARE) — Quality Requirements (International Organization for Standardization/International Electrotechnical Commission, Genève, 2007).
34. S. Lauesen, *Software Requirements: Styles and Techniques* (Addison-Wesley, London, 2000).
35. A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software.* (Wiley, West Sussex, 2009).
36. H. van Vliet, *Software Engineering: Principles and Practice* (John Wiley & Sons, New York, 2000).
37. I. F. Alexander and N. Maiden, *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle* (Wiley, West Sussex, 2004).
38. T. Lockwood and T. Walton, *Building Design Strategy: Using Design to Achieve Key Business Objectives* (Allworth Press, New York, 2008).
39. A. M. Davies, *Software Requirements: Objects, Functions, and States* (Prentice Hall, Upper Saddle River, 1993).
40. CMMI for Development Version 1.2: Improving processes for better products (Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA).
41. ANSI/IEEE Std. 829-1983, Software Test Documentation (The Institute of Electrical and Electronics Engineers, New York, 1987).
42. C. E. Dickerson and D. N. Marvis (eds.), *Architecture and Principals of Systems Engineering* (CRC Press, Boca Raton, 2010).

43. H. He, What Is Service-Oriented Architecture? http://www.xml.com/pub/a/ws/2003/09/30/soa.html (2011-08-19) (2003).
44. S. Pilemalm and N. Hallberg, Exploring service-oriented C2 support for emergency response for local communities, in *Proc. Fifth Int. Conf. Information Systems for Crisis Response and Management*, 2008, pp. 159–166.
45. S. Brinkkemper, Method engineering: Engineering of information systems development methods and tools, *Journal of Information and Software Technology* **38**(4) (1996) 275–280.
46. E. J. Chikofsky and J. H. Cross, Reverse engineering and design recovery: A taxonomy. *IEEE Software* **7**(1) (1990) 13–17.
47. S. Ducasse and D. Pollet, Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering* **35**(4) (2009) 573–591.
48. D. W. Oliver, J. F. Andary and H. Frisch, Model-based systems engineering, in A. P. Sage and W. B. Rouse, *Handbook of Systems Engineering and Management* (Wiley, Hoboken, 2009), pp. 1361–1400.
49. D. Firesmith, P. Capell, D. Falkenthal, B. Hammons, D. Latimer and T. Merendino, *The Method Framework for Engineering System Architectures (MFESA)* (Auerbach Publications, Boca Raton, 2008).
50. L. C. van Ruijven, Ontology for systems engineering, *Procedia Computer Science* **16** (2013) 383–392.
51. K. Harrison and G. White, A taxonomy of cyber events affecting communities, in *Proc. 44th Hawaii International Conference on System Sciences*, 2011, pp. 1–9.
52. A. Avizienis, J.-C. Laprie, B. Randell and C. E. Landwehr, Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and Secure Computing* **1**(1) (2004) 11–33.