# Refactoring Software Development Process Terminology Through the Use of Ontology

Paul M. Clarke[1,2], Antoni Lluís Mesquida Calafat[4], Damjan Ekert[5], J.J. Ekstrom[6], Tatjana Gornostaja[7], Milos Jovanovic[4], Jørn Johansen[8], Antonia Mas[4], Richard Messnarz[5], Blanca Nájera Villar[9], Alexander O'Connor[1,3], Rory V. O'Connor[1,2(✉)], Michael Reiner[10], Gabriele Sauberer[9], Klaus-Dirk Schmitz[11], and Murat Yilmaz[12]

[1] Dublin City University, Dublin, Ireland
{paul.m.clarke, alexander.oconnor, rory.oconnor}@dcu.ie
[2] Lero, The Irish Software Research Centre, Limerick, Ireland
[3] ADAPT, the Global Centre of Excellence for Digital Content Technology, Dublin, Ireland
[4] Universitat de les Illes Balears, Palma, Mallorca, Spain
{antoni.mesquida, milos.jovanovic, antonia.mas}@uib.es
[5] ISCN, The International Software Consulting Network, Graz, Austria
{dekert, rmess}@iscn.com
[6] Brigham Young University, Provo, UT, USA
jekstrom@byu.edu
[7] Tilde Company, Riga, Latvia
tatjana.gornostaja@tilde.com
[8] Whitebox Aps, Hørsholm, Denmark
jj@whitebox.dk
[9] TermNet, The International Network for Terminology, Vienna, Austria
{bnajera, gsaubere}@termnet.org
[10] European Certification and Qualification Association (ECQA), Krems, Austria
michael.reiner@fh-krems.ac.at
[11] Technical University of Cologne, Cologne, Germany
klaus.schmitz@th-koeln.de
[12] Çankaya University, Ankara, Turkey
myilmaz@cankaya.edu.tr

**Abstract.** In work that is ongoing, the authors are examining the extent of software development process terminology drift. Initial findings suggest there is a degree of term confusion, with the mapping of concepts to terms lacking precision in some instances. Ontologies are concerned with identifying the concepts of relevance to a field of endeavour and mapping those concepts to terms such that term confusion is reduced. In this paper, we discuss how ontologies are developed. We also identify various sources of software process terminology. Our work to date indicates that the systematic development of a software development process ontology would be of benefit to the entire software development community. The development of such an ontology would in effect represent a systematic refactoring of the terminology and concepts produced over four decades of software process innovation.


C. Kreiner et al. (Eds.): EuroSPI 2016, CCIS 633, pp. 47–57, 2016.
DOI: 10.1007/978-3-319-44817-6_4

**Keywords:** Software engineering · Software development process · Software development roles · Specialised communication · Terminology · Ontology

## 1 Introduction

Given that software development is a complex undertaking [1] which is human-centric in nature [2, 3], it follows that the consistent use of language and terminology should be an important consideration for software development. However, on the evidence of our initial research, it would appear that the software process domain suffers from an inconsistent use of terminology, to the extent that there may be large latent terminology problem concerning software development activities and roles [4]. That a terminology problem exists in our domain may to some extent be expected – since we have witnessed significant expansion over the past forty years. This expansion has been accompanied by innovation in the use of language and it is for this reason that we have *iterations* that are sometimes called *cycles*, *team leaders* that might be considered to be *project managers*, *features* that some might confuse with *user stories*, and *processes* that some refer to as *methods*. This expanse of terminology is not always accompanied by expansion of the underlying concepts and therefore, it could be claimed that new terminology is not always needed or helpful.

The consistent application of terminology is of concern to many fields of endeavour with the result that techniques have been developed to help address issues related to conceptual and terminological diversity. Ontological frameworks can be employed to reconcile diverse terminology through the systematic elaboration of the concepts of concern, while in parallel determining terminology-to-concept relationships. Once developed, an ontological framework can help to ground the language usage in a field, while it can also allow users of overlapping terminology to approximate where the conceptual scope of one term ends and another starts. Thus, a software process ontological framework could enable users of one software development process lifecycle to interact more smoothly with those using a different software development approach, while at the same time allowing all software developers to examine and clarify their own use of terminology and language. In previously published related work [4], the authors have elaborated on some examples of inconsistent terminology in the software process domain (refer to Fig. 1). In this paper, we provide some additional information on how ontologies are constructed and utilised, while also providing a brief overview of some of the present sources of software development process terminology, including books/bodies of knowledge, taxonomies and international standards.

This paper is structured as follows: Sect. 2 outlines the ontological approach and demonstrates how this technique can be of benefit to the software development community. Section 3 presents a brief overview of some of the sources of software process terminology, including an examination of the semantic distance that can be observed where multiple conflicting definitions are provided for the same term. Section 4 contains a discussion and conclusion.

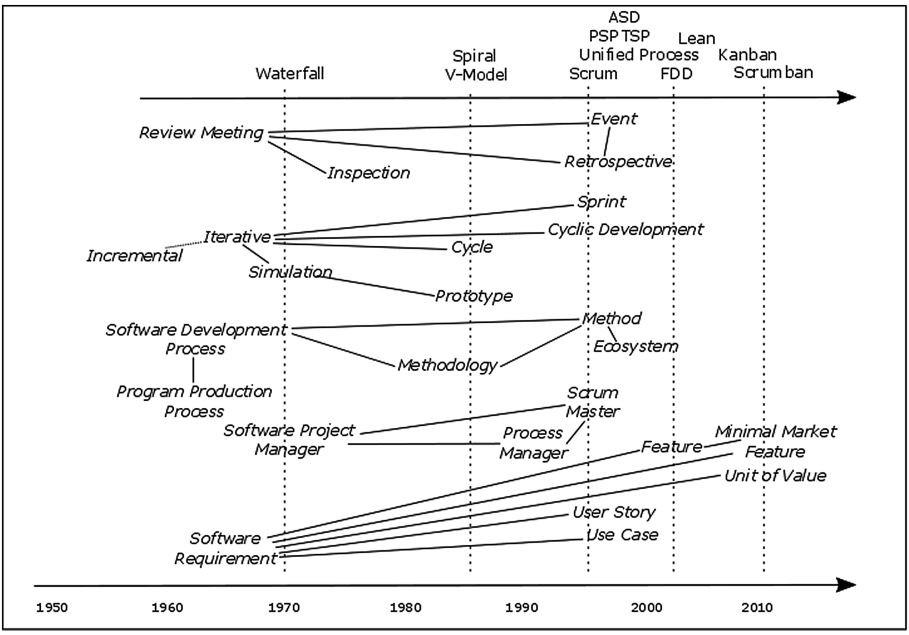


**Fig. 1.** Software terminology landscape – a process and role viewpoint

## 2 Terminology and Ontology

According to ISO 1087-1 [5], terminology work is the systematic collection, description, processing and presentation of concepts and their designations. This means that terminology is concerned with concepts and conceptual systems, making them explicit by means of definitions and designations as well as phrases within languages for special purposes. Terminology science provides the basic concepts and best practices for terminology work and terminography, i.e. for the systematic documentation and maintenance of terms. There are different ways to approach terminology work, as for example ad-hoc terminology management that focuses on solving instant problems and it is seen as a part of another process. On the other hand, systematic terminology management is based on the consistent application of working methods for a domain knowledge-oriented approach in order to harvest all the relevant concepts for a specific subject field.

In order to reduce the software development process terminological challenge, the concept orientation and the systematic terminology work approach are key: A systematic study of the field of knowledge that allows the collection of the concepts and terms and, thus, to develop a conceptual structure of the domain in the form of a concept system. The goal of our intended work, which we refer to as the SYNTHESIS Initiative, would be not only to enable clear communication between experts, but also to achieve the unique representation of concepts by avoiding redundancies, if possible, by setting a set preferred usage. This means, from the descriptive to the prescriptive work.

In order to develop this methodology for the successful harmonisation of the terminology for software development processes and roles, there is no need to start from scratch. Firstly, there are already existing standards from which to build the base for a solid terminological work (for example, ISO 704:2009 Terminology work — Principles and methods [6], and ISO 26162:2012 Systems to manage terminology, knowledge and content — Design, implementation and maintenance of terminology management systems [7]). And secondly, many terms, glossaries and resources for software development are already in use (and which in some cases are causing conflicts or unnecessary ambiguity).

Because of this, the first step would be to evaluate and assess available glossaries, documentation and resources and their reliability, information coming from authoritative bodies, any terminology work done by other institutions (for example, the ISO terminology about software process, to be found in the official ISO Online Browsing Platform [8] or the International Software Testing and Qualifications Board Glossary [9]). The reliability of such resources is a key factor while retrieving information. The assessment of the field of knowledge and identification and evaluation of the most relevant resources in the field of knowledge relating to software process terms build the basis for the ontological work. This can include domains such as security, reliability, methodology-specific terms and their interrelationship.

An ontology is the collection of concepts and terms in a certain language in a specific subject field, but also the formal, explicit (conceptual) models of object ranges in a computational representation [10]. According to the ISO, a model of product knowledge is achieved by a formal and consensual representation of the concepts of a product domain in terms of identified characterisation classes, class relations and identified properties [11]. An ontology also gives an indication about the degree of necessity of a prescriptive approach as it will show if there is a proliferation of terms for one concept, why this happens and which term candidate is the most adequate in each case. It should be highlighted that there is no single approach to ontology development that is universally applied, and that tooling can be utilised in order to support the development task [12].

According to ISO 704 [6], "it is necessary to bear in mind the subject field that gave rise to the concept and to consider the expectations and objectives of the target users, in organizing concepts into a concept system. The subject field shall act as the framework within which the concept field, the set of thematically related but unstructured concepts, is established … Characteristics shall be used in the analysis of concepts, the modelling of concept systems, in the formulation of definitions."

The terminology of a subject field always follows a concept system based on the relations existing between concepts. The unique position of each concept within a system is determined by the intension and the extension. In the case of concept systems based on generic relations, the concept system also reflects inheritance systems, because specific concepts inherit characteristics from their generic super-ordinates. The set of characteristics that come together as a unit to form the concept is called the *intension*. The objects viewed as a set and conceptualized into a concept are known as the *extension*. The two, the intension and the extension, are interdependent. For example, the characteristics making up the intension of 'mechanical mouse' determine the extension or the objects that qualify as mechanical mice. In some fields a distinction

is made between necessary, sufficient and essential characteristics. However, explaining this in this paper would exceed the scope of the same [6].

The effectiveness of ontologies in addressing terminology concerns has been demonstrated in many fields [12] and given the type of findings outlined in Sect. 1, there are therefore good reasons to consider its use in the software development process space. Indeed, the use of ontology is already being considered as a technique for the harmonisation of terminology in ISO/IEC Joint Technical Committee 1, Subcommittee 7 (JTC1 SC7) [13]. This ontology approach to the software process conceptual structure would also help to delimit and clarify roles and tasks in the working environment being an innovative and comprehensive approach in order, not only, to harmonise the existing resources, but also standardise curricula and skills for professions related to knowledge-driven software development.

A canonical software development process and roles ontology would be linked or embedded in a terminological database that would also give information about the terms behind the concepts, their definitions and characteristics that would improve the specialised communication among not only software developers, engineers, project managers, business managers and trainers, but would also provide an updated, centrally managed, comprehensive, online available resource for everyone (even laypersons).

Last but not least, the role of the experts is essential in this process. The terminologist can only draft the methodology for a successful terminology project. But the software process engineers are the experts that have the knowledge to select the best term candidates, draft definitions and validate relevant information. However, it is often the case that experts lack the basic skills and knowledge to carry out systematic terminology work. Therefore, it will be important to develop and implement an integrated, cross-disciplinary, and market-oriented training programme to create a new skills and qualifications portfolio for these professionals. This would be subject of a new ECQA [14] job role: the ECQA Certified Terminology Professional for Software Process Engineering certification and training.

## 3 Sources of Software Development Process Terminology

We do not want to give the impression that there is anything surprising in the current state of software development process terminology. Teams form around specific problems and projects and evolve a terminology for their community of practice [15]. Since the team faces common problems inside of a common set of constraints they naturally evolve a dialect that facilitates efficient communication for them. They may even publish ontological artifacts that aid others in joining the community, since turnover on teams is common.

Neither do we want to give the impression that no work has been done to create a common conceptual framework for Software Engineering. Two efforts stand out as particularly important to the development of an accepted formal vocabulary for Software Engineering: The Software Engineering Body of Knowledge (SwEBoK) [16] and the Software and Systems Engineering Vocabulary (SEVocab) [17]. The SwEBoK is a long term effort by the IEEE-CS to create a standard taxonomy for what a Software Engineer ought to know 4 years into her/his career. SEVocab is an edited aggregation

of ontological artifacts from over 100 ISO/IEC/IEEE standards. It has the appearance of a glossary since its basic organisation consists of terms followed by a list of definitions. However, since it includes *synonym* and *see-also* links to other terms it should probably be viewed as a topic map.

Professional societies can be classified as a formal communities of practice that form around a domains of expertise. Working Groups in those societies are chartered to create ontological artifacts for specific areas of interest or expertise. Since these ontological entities (Standards, Technical Reports, etc.) are designed to document a specific area of knowledge or expertise, they often contain a glossary of terms associated with the concepts used in the document. Examples include:

- ISO/IEC 24744:2007 Software Engineering–Metamodel for Development Methodologies [18]
- ISO/IEC 2382-20:1990 Information technology–Vocabulary–Part 20: System development [19]
- ISO/IEC TR 14471:2007 Information technology–Software engineering–Guidelines for the adoption of CASE tools [20]
- IEEE 1074-2006 IEEE Standard for Developing a Software Project Life Cycle Process [21]

In spite of these and many other efforts to document a standard terminology for areas in the discipline of Software Engineering, communities of practice continue to form, evolve, and create semantic drift. How many practitioners are aware that there is a standard metamodel for development methodologies? More instructive questions include: Does software development terminological semantic drift concern practitioners? Is semantic drift a latent as opposed to an open concern? Is semantic drift a worthy concern other than on large multi-supplier projects? These are questions that our ongoing efforts seek to explore.

As a measure of this drift in the system and software engineering space consider the SEVocab project. It is a database consisting of terms and definitions from 124 ISO/IEC/IEEE standards. Some terms have 7 or more associated concepts *after common definitions have been merged*. And this is from an aggregation of *formal standards*. If we are to reduce the entropy in software development process terminology, it will require significant human input even though there are some natural language processing and machine learning techniques that can reduce the manual effort. Some work has been accomplished but there is much more to be done. The Termediator project [22] currently has aggregated approximately 500 glossaries from domains closely related to Information Technology (Fig. 2 provides some background as to the types of sources of terminology incorporated into Termediator). The tool provides cluster analysis for concepts associated with a term which can aid in locating terms that are so over-used that they should be avoided as well as terms that are accepted as labels for a common concept across all of the domains represented. It also provides for rudimentary synonym analysis. This prototype demonstrates the utility of automated approaches to the initial analysis, but requires development to productise the implementation and add features to aid in analysis specific to the creation of an ontology for software development process terminology.

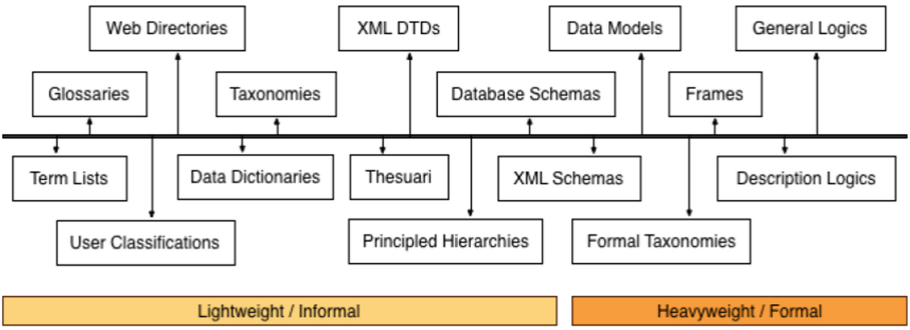


**Fig. 2.** Spectrum of software development process terminology sources

## 4 Discussion and Conclusion

According to the late-Enlightenment philosopher Georg Wilhelm Friedrich Hegel, *truth is found neither in the thesis nor the antithesis, but in an emergent synthesis which reconciles the two*. And this is precisely the type of truth that can be pursued in matters of language, as language is a representation of a concept, a concept has at its genesis an idea, and ideas do not lend themselves to perfectly complete definition or interpretation. Correspondingly, we can expect that a certain tension will necessarily arise between the correctness and common usage of terms such that absolute adherence to either is neither desirable nor advisable. So, our job with terminology is to bridge the gap from an idea to its representation, and to do so with a level of precision that is useful and effective for those who utilise the terms.

In earlier work, we demonstrated that there is a latent problem concerning the use of terminology for software development process and roles [4]. The purpose of this paper is to expand upon the ontology approach, explaining how it is suited to addressing the challenge of unifying the existing terms and concepts into a canonical software development process reference model. Such a model would also facilitate accurate interrelating of terms from different software development processes and methods, thus making it easier to understand how different software development models are similar, with positive benefits for those wishing to adapt processes or tailor processes [23]. Since such adaptive capability has been shown to be positively associated with business performance [24–26], any initiative which facilitates adaptation should be welcome. Furthermore, the reportedly high levels of SPI occurring in practice [27, 28], coupled with the rich variation in software development contexts [29] (which themselves are constantly changing [30]), suggests that greater consistency in terminology application would benefit the broader software development community.

A canonical model would also enable future software development process and method innovations to be readily interrelated to the large body of software process know-how that predated its arrival (something that is not easily achieved today). It would further have the benefit of revealing the genuine *newness* in newly proposed software development models and methods, as the conceptual mapping to the pre-existing concepts and terminology would be enabled through the ontology. This might not meet with

the approval of software process entrepreneurs seeking to cash-in on *new* approaches but it would certainly benefit the millions of software development practitioners who seek to understand each other and to robustly evaluate newly proposed approaches for (1) differences from their existing processes, and (2) integration into their present processes. Indeed, in the fullness of time, newly proposed approaches might demonstrate their uniqueness/newness through formally elaborating on the relationship to the canonical model – this way, genuine process innovation can be supported and promoted, and poorly constructed or ill-informed process innovation can be identified.

Since ontology development requires specific expertise and may be costly, it is important that we first examine the case for a software development process ontology prior to embarking on its development. Perhaps the primary benefit of ontologies is the creation and provision of intelligent, knowledge-based systems by "translating data into actionable insights for decision making" [31]. Earlier published research has reported numerous direct benefits from ontology adoption, including increased productivity of both information workers and software engineering (cost and time reduction, quality improvements) [32]. It has also been demonstrated that in safety critical and security application development, the use of ontology is crucial to fulfilling the objectives of the development work [33]. Beyond software engineering, there is widespread adoption of ontologies in domains such as biomedicine [34], oil extraction [35], and the automotive industry [36], where ontology has been shown to be an effective way of identifying, naming and relating concepts within processes and domains.

While advocating the use of ontology, we also seek to highlight that this is not simply a problem with terminology, it is a greater problem whereby we have not as a community managed to render the core concepts of our field in a universally digestible form. Added to this mix is the possibility that there may even an issue concerning appropriate levels of completeness of individual understandings of the various software development process models that are routinely adopted (or referred to). Take for example the Waterfall model which would appear to have become associated with single-pass, sequential software development in some quarters [4, 37, 38], even though Royce's original model explicitly recognises the need to utilise multiple iterations in software development (those seeking clarification on this point should refer to [39]).

For the software process improvement community, there can be a challenge when formulating discussions with individuals and organisations in order to establish precisely the extent to which a process is enacted, or to understand the boundary to individual roles within companies. Therefore, the challenge of process improvement can potentially be reduced through the introduction of mechanisms to improve the consistency of use of related terminology. It should be noted that our proposed undertaking is neither small nor simplex. Correspondingly, we have assembled a cross-disciplinary team and it is also our intention to pursue a community-led approach to the work program, including engagement with large numbers of software development experts so as to systematically agree concepts, terms and definition. Naturally, within individual software development approaches where clarity exists in relation to software process terms, we would not seek to redefine individual terms – but rather clearly identify their relationship to other process models. Furthermore, work of the

proposed nature requires many participants and many years, and therefore substantial funding, the pursuit of which is ongoing.

In software development, the importance of source code refactoring is well understood [40], without it source code can eventually become unmanageable (or at least economically challenging to maintain and extend). Terminology is no different, if allowed to drift unchecked, eventually the terminology and concepts become more and more confusing. The authors therefore propose that the time is anon to consider refactoring our software development process terminology, and that this is best achieved through the adoption of ontology.

## References


1. Clarke, P., O'Connor, R.V., Leavy, B.: A complexity theory viewpoint on the software development process and situational context. In: Proceedings of the 2016 International Conference on Software and System Process (ICSSP 2016). IEEE, San Francisco (2016)
2. Yilmaz, M., O'Connor, R.V., Clarke, P.: A systematic approach to the comparison of roles in the software development processes. In: Mas, A., Mesquida, A., Rout, T., O'Connor, R. V., Dorling, A. (eds.) SPICE 2012. CCIS, vol. 290, pp. 198–209. Springer, Heidelberg (2012)
3. Yilmaz, M., O'Connor, R., Clarke, P.: Software development roles: a multi-project empirical investigation. ACM SIGSOFT Softw. Eng. Notes **40**(1), 1–5 (2015)
4. Clarke, P., et al.: An investigation of software development process terminology. In: Clarke, P.M., O'Connor, R.V., Rout, T., Dorling, A. (eds.) SPICE 2016. CCIS, vol. 609, pp. 351–361. Springer, Heidelberg (2016). doi:10.1007/978-3-319-38980-6_25
5. ISO: ISO 1087-1:2000 terminology work – vocabulary – part 1: Theory and application, 1st edn. ISO, Geneva, Switzerland (2000)
6. ISO: ISO 704:2009 terminology work — principles and methods, 1st edn. ISO, Geneva, Switzerland (2009)
7. ISO: ISO 26162:2012 systems to manage terminology, knowledge and content — design, implementation and maintenance of terminology management systems, 1st edn. ISO, Geneva, Switzerland (2012)
8. ISO: Online Browsing Platform. https://www.iso.org/obp/ui/#home
9. ISTQB, Standard Glossary of Software Testing Terms. http://www.istqb.org/downloads/glossary.html
10. Budin, G.: Methodology for dynamic ontology creation from terminologies to ontologies – tools of knowledge organization. In: Proceedings of International Terminology Summer School 2009, TermNet, Cologne, Germany (2009)
11. ISO: ISO 13584-32:2010 - industrial automation systems and integration - OntoML: Product ontology markup language, 1st edn. ISO, Geneva, Switzerland (2010)
12. Aardi, G., de Almeida Falbo, R., Pereira Filho, J.G.: Using objects and patterns to implement domain ontologies. J. Braz. Comput. Soc. **8**(1), 43–56 (2002)
13. Henderson-Sellers, B., McBride, T., Low, G., Gonzalez-Perez, C.: Ontologies for international standards for software engineering. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 479–486. Springer, Heidelberg (2013)
14. ECQA: European Certification and Qualification Organisation. www.ecqa.org
15. Wenger, E.: Communities of Practice: Learning, Meaning, and Identity, 1st edn. Cambridge University Press, Cambridge (1998)

16. IEEE: Guide to the software engineering book of knowledge (SWEBOK). IEEE Computer Society, Los Alamitos (2004)
17. IEEE/ISO/IEC, SE Vocab - Software and Systems Engineering Vocabularly. https://pascal.computer.org/sev_display/index.action
18. ISO/IEC: ISO/IEC 24744:2007 software engineering–metamodel for development methodologies, 1st edn. ISO/IEC, Geneva, Switzerland (2007)
19. ISO/IEC: ISO/IEC 2382-20:1990 information technology–vocabulary–part 20: System development, 1st edn. ISO/IEC, Geneva, Switzerland (1990)
20. ISO/IEC: ISO/IEC TR 14471:2007 information technology–software engineering–guidelines for the adoption of CASE tools, 1st edn. ISO/IEC, Geneva, Switzerland (2007)
21. IEEE: IEEE 1074-2006 IEEE standard for developing a software project life cycle process, 1st edn. IEEE, Washington, DC (2006)
22. Riley, O., Richards, J., Ekstrom, J., Tew, K.: Termediator II: measuring term polysemy using semantic clustering. In: Proceedings of 3rd Conference on Research in Information Technology (RIIT 2014), pp. 81–86. ACM, New York (2014)
23. Coleman, G., O'Connor, R.: Investigating software process in practice: a grounded theory perspective. J. Syst. Softw. **81**(5), 772–784 (2008)
24. Clarke, P., O'Connor, R., Leavy, B., Yilmaz, M.: Exploring the relationship between software process adaptive capability and organisational performance. IEEE Trans. Softw. Eng. **41**(12), 1169–1183 (2015)
25. O'Connor, R.V., Clarke, P.: Software process reflexivity and business performance: initial results from an empirical study. In: Proceedings of the 2015 International Conference on Software and System Process, pp. 142–146. ACM, New York (2015)
26. Clarke, P., O'Connor, R.V.: The influence of SPI on business success in software SMEs: an empirical study. J. Syst. Softw. **85**(10), 2356–2367 (2012)
27. Clarke, P., O'Connor, R.V.: An empirical examination of the extent of software process improvement in software SMEs. J. Softw. Evol. Process **25**(9), 981–998 (2013)
28. Clarke, P., O'Connor, R.V., Yilmaz, M.: A hierarchy of SPI activities for software SMEs: results from ISO/IEC 12207-based SPI assessments. In: Mas, A., Mesquida, A., Rout, T., O'Connor, R.V., Dorling, A. (eds.) SPICE 2012. CCIS, vol. 290, pp. 62–74. Springer, Heidelberg (2012)
29. Clarke, P., O'Connor, R.V.: The situational factors that affect the software development process: towards a comprehensive reference framework. J. Inf. Softw. Technol. **54**(5), 433–447 (2012)
30. Clarke, P., O'Connor, R.V.: Changing Situational Contexts Present a Constant Challenge to Software Developers. In: O'Connor, R.V., Umay Akkaya, M., Kemaneci, K., Yilmaz, M., Poth, A., Messnarz, R. (eds.) EuroSPI 2015. CCIS, vol. 543, pp. 100–111. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24647-5_9
31. Stanford Center for Biomedical Informatics Research (BMIR) at the Stanford University School of Medicine, Protégé. http://protege.stanford.edu/about.php
32. Oberle, D.: How ontologies benefit enterprise applications. Semant. Web **5**(6), 473–491 (2014)
33. Greciano, G., Budin, G.: Designing linguistic support for risk management communication. https://www.uibk.ac.at/translation/aktuelles/aktuelles/unterlagen/papergrecianobudineumedinhbsept2006.pdf
34. Hoehndorf, R., Schofield, P.N., Gkoutos, G.V.: The role of ontologies in biological and biomedical research: a functional perspective. Briefings Bioinform. **16**(6), 1069–1080 (2015)

35. Kharlamov, E., et al.: Ontology based access to exploration data at Statoil. In: Arenas, M. (ed.) ISWC 2015. LNCS, vol. 9367, pp. 93–112. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25010-6_6
36. Rychtyckyj, N., Klampfl, E., Gusikhin, O., Rossi, G.: Application of intelligent methods to automotive assembly planning. In: 2007 IEEE International Conference on Systems, Man and Cybernetics, pp. 2479–2483. IEEE, New Jersey (2007)
37. Molokken-Ostvold, K., Jorgensen, M.: A comparison of software project overruns - flexible versus sequential development models. IEEE Trans. Softw. Eng. **31**(9), 754–766 (2005)
38. Larman, C., Basili, V.R.: Iterative and incremental development: a brief history. IEEE Comput. **36**(6), 47–56 (2003)
39. Royce, W.: Managing the development of large software systems: concepts and techniques. In: Western Electric Show and Convention Technical Papers. IEEE Computer Society, Los Alamitos (1970)
40. Mens, T., Tourwe, T.: A survey of software refactoring. IEEE Trans. Softw. Eng. **30**(2), 126–139 (2004)