An Investigation of Software Development Process Terminology

Paul Clarke[1,2(✉)], Antoni-Lluís Mesquida[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@dcu.ie, {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,gsauberer}@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. The practice of software development has evolved considerably in recent decades, with new programming technologies, the affordability of hardware, pervasive internet access and mobile computing all contributing to the emergence of new software development processes. The newer process initiatives, which include those which are sometimes referred to as agile or lean methods, have brought with them new terms, which sometimes reflect the introduction of novel concepts. Other times, new terms correspond to long established concepts that have been repackaged. The net position is that we have a proliferation of language and term usage in the software development process domain, a problem which has implications for assessors and assessment frameworks, and for the broader community. In this paper, we explore this problem, finding that it is worthy of further research. Plus, we identify a technique suited to addressing this concern: the establishment of a canonical software process ontological model.

P.M. Clarke et al. (Eds.): SPICE 2016, CCIS 609, pp. 351–361, 2016.
DOI: 10.1007/978-3-319-38980-6_25

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

1 Introduction

Software development is a complex activity [1] that is highly sensitive to human interaction and team work [2]. We should therefore pay very careful attention to human communication mechanisms, including language and terminology. The concern of the authors of this paper is that we are perhaps not paying sufficient attention to the area of language and terminology in software development, and in particular our focus is on a potentially large, latent terminology problem concerning software development activities and roles. That a terminology problem may exist in our field ought not to come as any major surprise – our domain has witnessed rapid expansion over the past thirty years, an expansion that has been fueled by innovation. Such innovation is very welcome and a foundation for many of the advancements witnessed, and with it comes diversity and innovation in use of language. It is for this reason that we have *iterations* that are sometimes called *sprints*, *team leaders* that might be considered to be *ScrumMasters*, *use cases* that some might confuse with *user stories*, and *reviews* that some refer to as *retrospectives*. This type of drift in terminology is not always accompanied by expansion of the underlying concepts and therefore, it could be claimed that some new terminology is neither required nor desirable.

The importance of systematic terminology work is of concern to many fields of endeavour with the result that methods have been developed to help address issues related to language diversity. One technique that can be employed to address issues of terminology diversity is the grounding of a set of terms in a conceptual framework called an ontology. An ontology sets out by first identifying the concepts of importance to an area of interest, an important step as this can help to interrelate terminology which has emerged in a field. Thus, the ontological focus is first on the concepts or meanings of interest in a field and thereafter in the terms associated with these meanings.

In this paper we briefly examine the scale of the terminological problem in software development processes (Sect. 2) and introduce the methods of systematic terminology concept-orientation (Sect. 3). Section 4 presents a discussion on the implications of our initial research findings, with Sect. 5 containing the conclusion.

2 Software Development Language and Terminology

A key question to ask in the early stages of any research effort is: *Does the envisaged problem appear worthy of research?* Correspondingly, our primary work to date has focused on just this question. Although our research remains at a nebulous stage, our present findings indicate that there is problem regarding software process terminology and that this problem extends into the identification of various software development roles. In this position paper, we seek only to very broadly scope the problem such that readers can gain an initial appreciation for the impact and nature of terminology drift in the software development space. In undertaking our research, we have looked to the

early days of software development, seeking to identify the origin of some of the central concepts and terminology in our field. This search, which is far from complete, has rendered the view presented in Fig. 1.

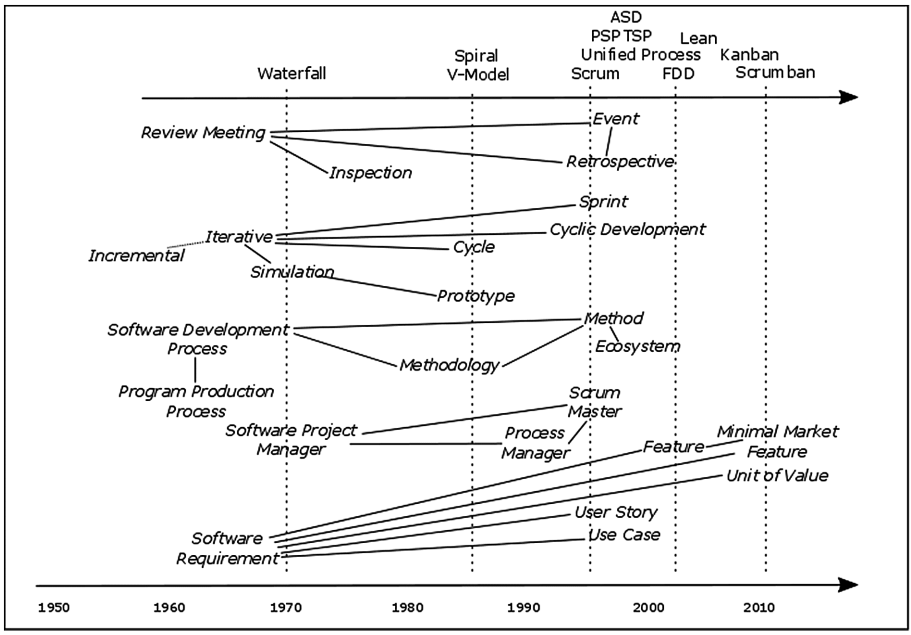

Fig. 1. Software terminology landscape – a process and role viewpoint

The *software development process* – or *software process* as it is sometimes shortened - exists as a documented concept since at least the early 1960 s [3]. More recently, the agile software development community has opted for the term *method* to identify the software process or aspects of the software process, though it has been observed by one of the agile founding fathers that the terms *method* and *methodology* should be replaced by the term *agile software development ecosystems* [4]. Perhaps the inclination to describe the *process* as a *method* or *methodology* in the agile domain emanates from the concept that the agile structure adopted should be of a *barely sufficient* nature [4], containing only as much process as is beneficial, and therefore the use of the term *method* or *methodology* sets the agile approach apart from more comprehensive process elaborations – if this was the intention, then it could have probably been satisfied just as well (and with less recourse to terminological debate) through use of an alternative label, perhaps: *agile software process.* Whatever the case, and whatever your *process* or *method* or *methodology* or *ecosystem* persuasion, that such debate and deviation exists concerning the labelling of the domain itself is indicative of intrinsic terminology issues in our field – if we cannot agree on the name for the domain, it does not bode well for our ability to consistently apply terminology in identifying concerns within the domain – including the roles involved in producing software.

When it is considered that the term *method* has a long-established and very specific meaning in programming [5], it could be suggested that it was unhelpful to overload the term *method* when labelling an agile software development process. Concerning the adoption of the term *agile method*, it may be the case that this terminological divergence from the more traditional *process* term was considered important by early agile innovators as a mechanism to distinguish the agile development philosophy from its precursors. Central to this innovation is the degree of agility enabled by agile methods, a point that is well made by Barry Boehm and Richard Turner [6]. Though, on the subject of language, it is worth highlighting that the juxtaposition of the terms *Agility* and *Discipline* in the title of Boehm and Turner's work is unfortunate as it carries with it the implicit suggestion that agile software development is something that is not disciplined or which may not require discipline (which of course is not the case, and which one suspects was not intended by the authors). And this is not an issue that is evident only in Boehm and Turner's work – one of the primary advocates for agile software development, Jim Highsmith, has employed an equally unsatisfactory juxtaposition when outlining the difference between the two camps as balancing *Flexibility* and *Structure* [4]. Of course, flexibility is not achieved through the removal of structure, rather it is achieved through the adoption of structures that support flexibility – and one suspects that this is a further instance of unintended language implications from the perspective of the original author. So, all around we appear to have some lack of clarity with respect to term usage and even a weak concept-to-language coupling, and this is something which the authors consider to be leading to misunderstanding in our profession in general, the full cost of which could be greater than many might expect.

Two concepts that appear to be central to many software development process models are *iterations* and *increments.* Iterative software development, which is a core feature of agile software development, is a not an invention of the agile movement [7], and along with incremental development, it has been noted as beneficial for software development since at least the 1960 s [8, 9]. Indeed, some in our field may be surprised to learn that the waterfall model [10] also caters for iterative development – a fact which the authors suspect may be largely over looked in some quarters. The basic point here is that the *iteration* and *increment* concepts are long established in the software development domain. Yet, these concepts are not necessarily immediately or intuitively obvious across all life cycle models – at least not from a language and terminology perspective. Perhaps the most obvious example is to be found in the term *sprint*. A *Sprint* is "*an iterative cycle of development work*" [11] and as such, is essentially the same concept as an *iteration* (in Royce's Waterfall) *or cycle* (in Boehm's Spiral [12]). One could therefore legitimately claim that a *sprint* could have been described using existing terms - perhaps as a *short iteration* - and it is not difficult to see how such language use would have benefited those hordes of software developers already familiar with the term *iteration*. Even today, one suspects that the exact relationship between a sprint and a traditional iteration is not entirely clear to all in our field. Those outside our field could not be blamed for seeing no relationship whatsoever from looking at the terminology employed.

Beyond the inconsistent use of terminology across various software development processes, in recent times we have the added confusion that there would appear to be an

increasing tendency to create new titles for individual actors (or software development roles). In [13] we are told that "*the ScrumMaster fills the position normally occupied by the project manager*" with the ScrumMaster responsible for managing the Scrum process but not for the definition and management of the work itself. However, it has been observed in some case studies that pure self-organisation can be difficult to achieve in practice, with the theoretical disjoint between work management and process management being difficult to realise in some Scrum environments where teams may need a team member pushing the workload towards completion [13, 14] or where the ScrumMaster may tend to naturally assume this authority [15] (though it should be noted that [13] puts this issue down to a failure to implement Scrum correctly). It is therefore the case, that at least in some cases, the ScrumMaster may – even if incorrectly so – operate as a traditional project manager.

Advocates of Scrum have legitimised this role naming with the assertion that the ScrumMaster needs to be distinguished from the traditional Software Project Manager role (which has existed at least since the 1960 s [16]), that their authority should essentially be indirect, with their knowledge and policing of Scrum practices being the limit of their power [13]. This being the case, the role of traditional process manager (for which the following definition has been suggested: "to provide information to specialise and instantiate the process model, and to activate and monitor the execution of this instantiated model" [17] would appear to overlap greatly with that of a ScrumMaster, thus questioning the need to introduce another new role title. Even in rugby, from which Scrum claims to draw its inspiration in metaphor, there is no such role as a ScrumMaster (there is a Scrum Half, who has varying degrees of authority in terms of calling different pre-planned plays at different times). So the software process terminology issue is broad, it is not just concerned with the adoption of different terms for similar (or equivalent) concepts across different software process models, it also extends to the terminology adopted for different roles within software development teams.

Further examples of issues related to terminology may be found in the treatment of software requirements, which may sometimes be referred to as *requirements*, other times as *use cases*, other times again as *user stories* and *features* (and one expects many other labels besides). With the passing of time, what was once the single homogenous *software requirements* activity has come to be tackled using a variety of different techniques. The term *software requirements* is in use at least as early as 1965 [18] and was quite possibly common parlance for some time prior to that point. *Use Cases* can be adopted when gathering requirements and have been reported to have "fulfilled the role of software requirements well" [19]. Within agile software development there would appear to be a number of terms used for the purpose of identifying software requirements, many of which appear to be related to the *use case* concept. In Adaptive Software Development [20], the term *feature* is preferred with a number of features constituting the *scope* (and a number of features may be required in order to deliver a single piece of *functionality*). Feature Driven Development (FDD) [21] adopts a similar convention, where *features* are small client-valued functions that can be delivered in two weeks and where sets of features may be utilised to deliver higher level complex *functions*. Consequently, on the evidence accumulated in our cursory investigation, a significant research effort might be required just to harmonise the current software requirements concepts and

terminology. The broader process terminology issue is certainly current and if anything, our findings suggest that we may have a large and perhaps mostly latent terminology problem – and to answer the question we set forth at the start of this research: *Does the envisaged problem appear worthy of research?* Our conclusion, based on early efforts, suggests that it is a problem worthy of further research.

3 Terminology and Ontology

In order to reduce a terminological problem, the common approach is to retrieve and store already existing terms, approve definitions and, if necessary, coin new terms. It is what the terminology science call systematic terminology work. In this case, we propose applying this to software development process terminology. To address this task, there is no need to start from scratch. As we have illustrated in Sect. 2, many terms are already in use and, in some cases, may be confusing users. The first step would be an assessment of the field of knowledge by identifying and evaluating the preexisting related resources. For example, the ISO terminology about software process, to be found in the official ISO Online Browsing Platform [22] or the International Software Testing and Qualifications Board Glossary [23], just to mention two examples (there are a great many sources of software process terminology in existence – too many to list in this paper). The reliability of such resources is a key factor while retrieving information.

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. A study of the field of knowledge will allow the collection of the concepts and terms of this specific field and, thus, to develop a conceptual structure of the domain in the form of an ontology. This ontology is essential to study the relations between concepts in order to reduce some of the problems presented in Sect. 2.

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 [24]. 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 characterization classes, class relations and identified properties [25]. An ontology also gives an indication about the degree of necessity of a prescriptive approach as it will show if there is proliferation of terms for one concept, why this happens and which term candidate is the most adequate in each case. The ontological approach will also set the path for the concept orientation of the terminology database. 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 [26].

This ontology approach to the software process conceptual structure would also help to delimit and clarify roles and tasks in the working environment. This can help not just to harmonise existing resources but also to standardise curricula and skills for professions related to knowledge-driven software development. The software process

community will directly benefit from a terminology database and ontology to guide them through the terminology related to tasks, roles, competences and skills.

All this work would result in a much-needed, industry standard terminological database with an ontology component for knowledge-driven holistic application development. The existence of such a terminological database (or TermBase) would facilitate lower friction, higher quality development in multi-party projects, and assist in tacit knowledge maintenance as teams evolve, and ultimately can be a canonical collection of the state-of-art terminology for the software development process that could be used as lookup reference tool not only for experts and peers, but also for new-comers in the community as well as laymen.

The effectiveness of ontologies in addressing terminology concerns has been demonstrated to be effective in many fields [27] and given the type of findings identified in Sect. 2, there are good reasons to consider its use in the software development process space. In the following section, we present some discussion on the implications of adopting ontology structures for the software process and software development roles.

4 Discussion

In Sect. 2, we demonstrated that there is diversity in the use of language and terminology in the software development process domain. This diversity has accumulated over the decades, with various waves of process innovation often introducing new terminology. For example, we highlighted the new terminology introduced in the Scrum process [28], with *ScrumMasters* and *Sprints* seeming to overlap heavily with the pre-existing concepts of *Project/Process managers* and *Iterations*. It should not be inferred from the examples that we highlight in this work that they originate from process models or approaches that might be considered especially problematic from a terminology perspective. Rather, the examples employed are often from some of the most important and impactful process innovations (for example, Scrum, the Waterfall model and the Spiral model). Through looking to some of the most impactful process models, we can also start to get some indication of the depth and nature of the diversity of language, and in this case, our finding is that a software professional familiar with Scrum may have difficulty relating some Scrum terminology to the Waterfall model (and vice versa). Indeed, when it is further considered that a wide variety of situational or environmental factors inform process selection [29], that processes may be tailored for individual project needs [30], and that the software process itself may be continually evolving [31, 32], the problem of term usage is perhaps amplified – since a hybrid software development process may further confuse language and terminology usage. Our general impression is that there is a wide variety of different terminology adopted to represent similar or overlapping concepts, and perhaps a lack of clarity with respect to the salient concepts of concern across different software development efforts.

If we accept that diversity exists in software development process terminology – and few, we suspect, would argue to the contrary – the debate shifts to examining the scale of the diversity and its potential impact. Our initial research in this space suggests that

there may be a large degree of diversity in software development process terminology and we plan further, more expansive, investigations to fully evaluate the problem size. However, our initial standpoint is that the diversity of terminology is a sizeable problem at present, with implications for many software development projects. For large software development undertakings requiring multiple suppliers, the absence of a common and cohesive understanding of scope, roles and processes may prove to be a challenging and costly issue. All we have to do is consider the case where one of the suppliers is working with a process that deals with *User Stories*, *Sprints* and *ScrumMasters*. Meanwhile, a second supplier deals in the different terminological currency of *Requirements*, *Iterations* and *Project Managers*. Given the reported tendency to tailor and adapt software development processes [30, 32] and the potential importance of such actions in supporting business performance [31, 33], the ability to precisely relate terms between different methods may be particularly beneficial for software development process evolution – and efforts in this respect would be eased through the establishment of a canonical software development process ontology.

And this is not merely a problem of terminology, it is deeper than just that – it is likely to be a problem whereby we have not as a community managed to render the core concepts of our field in a universally digestible form (a form which must permit the interaction of concepts from different process models and lifecycles in the first instance, while the labels and terms adopted in individual process approaches would ideally be related to concepts from different approaches). Added into this mix is the further suspicion of the authors that there may even an issue concerning appropriate levels of completeness of individual understandings of the various software development process models that have been proposed. Anecdotal evidence from the experience of the authors suggest that there may insufficiencies in understanding for the models that do exist – with one example being the Waterfall model which it seems may have become associated with single-pass, sequential software development in some quarters, even though Royce's original contribution in fact dedicates specific attention to the need to utilize multiple iterations in software development (those seeking clarification on this point should refer to [10]).

This problem of terminology diversity is not just manifested in large multiple-supplier software projects, it may be a problem for the field in general. Each time a company hires a new software developer, there is inevitably going to be some distance between the newcomer's personal dictionary of terms and the established practice in the new company. Partly this is a problem of education both within the educational sector and also personal professional development, but is also a problem that is not assisted by the unfortunate reality that we do not presently have a single canonical software development process ontology (incorporating roles) – and therefore, associations between individual software development process models are difficult to achieve. And this is not a problem that has gone entirely unnoticed in our field, for example [34] has proposed an initial ontology for the purpose of ISO/IEC Sub Committee 7 (SC7), a welcome contribution in the eyes' of the authors. Our proposal however is greater than just SC7 language and terminology concerns, we seek to address the broader software engineering community, large swathes of which have (at best) only loose interaction with software engineering standards. Furthermore, we have established a cross disciplinary team of

expertise that we feel is essential to achieving the goal of our research to reduce the problem of unintended or harmful terminological diversity in our field. This team includes software development process expertise, terminological and ontological specialisms, proficiency in knowledge management, and computational linguistics skills. With this team, we seek to develop a canonical ontology for software development processes which incorporates all major software development process lifecycles and associated terminology, with the systematic community-led establishment of a commonly accepted set of concepts and definitions for our field (based upon the many sources of software process terminology that are presently in existence) and the enablement of access to this knowledge store (either directly with queries or through published APIs) through readily available channels (such as internet/cloud-based services).

For the software process assessment community, especially those who are regularly engaged in process assessments, there can be a challenge when formulating discussions with individuals and organizations 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 assessment could potentially be eased – if only slightly – through the introduction of mechanisms that might improve the consistency of use of terminology related to software processes and roles such as is proposed by the authors. A cautionary note should be registered concerning our proposed undertaking though: it is neither small nor simplex. It is for this reason that we have assembled a cross-disciplinary team and it is also the foundation of our determination to pursue a community-led approach to the work program. This could include, for example, engagement with relatively 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. Finally, work of the proposed nature requires many participants and many years, and therefore substantial funding, the pursuit of which is ongoing.

5 Conclusion

In this paper, we have provided a brief snapshot of some of the terminology issues that exist in contemporary software development. This snapshot suggests that there is a large, complex and potentially very costly problem concerning the present application of terminology to both processes and roles involved in software development. This perceived problem does not have a quick or simple solution but rather a solution will require the sustained engagement of multiple disciplines, including terminology expertise, software development specialists, knowledge management know-how, and computational linguistics. It should also be emphasised that it would be a folly to attempt to eliminate the problem, but that the challenge is to reduce the problem to more manageable proportions.

Our proposal is to systematically develop a canonical software development process and roles ontology. In this proposed community-led work program, the contributions of earlier working groups and process initiatives should not be overlooked, but rather

carefully incorporated so as to maximize the benefit of earlier important work in this space. The resultant canonical ontology should be capable of seamlessly integrating emerging and future software development lifecycles, and it should comfortably accommodate the primary process models in active use, including more recent innovations in agile and lean software development – with this accommodation taking care to fully appreciate the conceptual differences between approaches rather than attempting to force dissimilar concepts together. The proposed ontology can be used in educational settings, in professional training programs, it may be integrated into existing software tooling solutions, and also adopted by industrial software developers. To draw analogy with an established programming practice, it would in a sense represent a *refactoring* of the terminology and language usage in our domain. A refactoring which, we suggest, is overdue and essential to future smooth and professional operation of our field, including but not limited to those involved in process assessment.

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. Singleton, J.W.: Software design and implementation. System Development Corporation, Santa Monica (1963)
4. Highsmith, J.: What is agile software development? Crosstalk – the journal of defense. Softw. Eng. **15**(10), 4–9 (2002)
5. Cox, B.J.: Object-Oriented Programming -an Evolutionary 'Approach, 1st edn. Addison-Wesley Inc., Reading (1986)
6. Boehm, B., Turner, R.: Balancing Agility and Discipline - A Guide for the Perplexed. Pearson Education Limited, Boston (2003)
7. Lindvall, M., et al.: Empirical findings in agile methods. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 197–207. Springer, Heidelberg (2002)
8. Larman, C., Basili, V.R.: Iterative and incremental development: a brief history. IEEE Comput. **36**(6), 47–56 (2003)
9. Basili, V.R., Turner, A.J.: Iterative enhancement: a practical technique for software development. IEEE Trans. Softw. Eng. **SE-1**(4), 390–396 (1975)
10. 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)
11. Schwaber, K.: SCRUM development process. In: Business Object Design and Implementation Workshop at the 10th Annual Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 1995). Springer-Verlag, Berlin (1995)
12. Boehm, B.: A spiral model of software development and enhancement. IEEE Comput. **21**(5), 61–72 (1988)
13. Schwaber, K.: Agile Project Management with Scrum. WP Publishers & Distributors Pvt Ltd., Bangalore (2004)

14. Cristal, M., Wildt, D., Prikladnicki, R.: Usage of SCRUM practices within a global company. In: IEEE International Conference on Global Software Engineering, pp. 222–226. IEEE (2008)
15. Moe, N.B., Dingsoyr, T., Dyba, T.: Overcoming barriers to self-management in software teams. IEEE Softw. **26**(6), 20–26 (2009)
16. Jones, M.M., McLean, E.: Management problems in large-scale software development projects. Ind. Manage. Rev. **11**, 1–15 (1970)
17. Conradi, R., Fernström, C., Fuggetta, A., Snowdon, R.: Towards a reference framework for process concepts. In: Derniame, J.-C. (ed.) EWSPT 1992. LNCS, vol. 635, pp. 1–17. Springer, Heidelberg (1992)
18. Bauer, W.F., Campbell, E.K.: Advanced naval tactical command and control study (informatics report TR-65-58-2). 1st edn. Prepared for Advanced Warfare Systems Division, Naval Analysis Group, Office of Naval Research by Informatic Inc. (1965)
19. Kulak, D., Guiney, E.: Use Cases: Requirements in Context, 1st edn. Addison-Wesley, Boston (2004)
20. Highsmith, J.: Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House Publishing, New York (2000)
21. Palmer, S.R., Felsing, J.: A Practical Guide to Feature-Driven Development. Prentice Hall, Upper Saddle River (2002)
22. ISO, Online Browsing Platform. https://www.iso.org/obp/ui/#home
23. ISTQB, Standard Glossary of Software Testing Terms. http://www.istqb.org/downloads/glossary.html
24. 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)
25. ISO: ISO 13584-32:2010 - industrial automation systems and integration - OntoML: Product ontology markup language. 1st edn. ISO, Geneva, Switzerland (2010)
26. Aardi, G., Falbo, R.D.A., Pereira Filho, J.G.: Using objects and patterns to implement domain ontologies. J. Braz. Comput. Soc. **8**(1), 43–56 (2002)
27. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information - a survey of existing approaches. In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA, pp. 108–117 (2001)
28. Schwaber, K., Beedle, M.: Agile software development with SCRUM. Prentice Hall, Upper Saddle River (2002)
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. Coleman, G., O'Connor, R.: Investigating software process in practice: a grounded theory perspective. J. Syst. Softw. **81**(5), 772–784 (2008)
31. 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)
32. 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)
33. 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)
34. 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)