In Search of the Origins and Enduring Impact of Agile Software Development

Paul M. Clarke
School of Computing
Dublin City University, Ireland
Lero - Irish Software Research Centre
Paul.M.Clarke@dcu.ie

Rory V. O'Connor
School of Computing
Dublin City University, Ireland
Lero - Irish Software Research Centre
Rory.OConnor@dcu.ie

Murat Yilmaz
Department of Computer Engineering
Çankaya University
Ankara, Turkey
MYilmaz@cankaya.edu.tr

ABSTRACT

The Agile Manifesto is a philosophical touchpoint for all agile software development (ASD) methods. We examine the manifesto and some of its associated agile methods in an effort to identify the major impacts of ASD. We have encountered some difficulty in delineating agile and non-agile software processes, which is partially the result of terminological confusion. It is clear from the volume of published research that ASD has made a significant contribution, and we have identified two lasting and important impacts. Firstly, the reduction in iteration durations and secondly, the push for reduced levels of documentation (especially in relation to software requirements). Other aspects of the Agile Manifesto may not have exerted a significant impact; for example, the use of tooling to automate processes has become central to continuous software engineering (CSE) and may not be wholly congruent with the manifesto. Furthermore, many organisations may still rely on business contracts despite calls in the manifesto for greater levels of informal customer collaboration.

CCS CONCEPTS

• **Software and its engineering → Software development process management → Software development methods**.

KEYWORDS

Agile Software Development; Continuous Software Engineering

1 INTRODUCTION

Agile software development is underpinned by the Agile Manifesto [1] and can be considered to represent a philosophical adjustment to the traditional software lifecycle models – though many of the underlying concepts adopted in ASD are reincarnations of previously existing concepts from traditional software lifecycle models (a point that is explicitly recognised by Kent Beck [2]).

The Agile Manifesto itself sets out the philosophy of ASD, and it is to this philosophy that the various ASD approaches ascribe[1]. As such, the manifesto does not actually identify a lifecycle model but rather the principles that agile lifecycle models should aspire to, for example: regular customer collaboration as opposed to strict contract adherence, and responding to changing requirements as opposed to rigidly restricting the requirements over long periods. However, this facility for changing requirements on a regular basis is enabled through iterative development that although a core feature of ASD, is not an invention of the agile philosophy [3]. Along with incremental development, iterative development has been noted as a beneficial software process characteristic since at least the 1960s [4], [5]. This in part accounts for the difficulties that sometimes arise when trying to classify the Unified Process [6] which might be categorised as falling under a traditional lifecycle classification but which other research has suggested to be aligned with ASD [7]. Therefore, it seems that pinpointing the novelty of aspects of ASD is perhaps not a straightforward proposition.

Our research presented herein focuses on the values presented in the Agile Manifesto as a means to clarifying the origins and impact of ASD. This exercise has highlighted some difficulties in the very terminology adopted across software development in general, which shares some conceptual space with earlier research conducted by the authors [8], [9]. The first value of the Agile Manifesto states that "*Individuals and interactions [are valued] over processes and tools*" thus promoting the roles of humans and their interactions in software development efforts - which is intuitively appealing given that most software development is a human-intensive activity. However, even in this first principle we find potential difficulties in language. For example, how exactly do individuals interact as they go about the task of developing software? One possible response to this question could be that individuals go about the task of producing software through the disciplined application of a sequence of steps that will result in a viable software product (a job which is enabled through communication and collaboration); a response which we find to be largely congruent with a long-established definition for the

[1] Note that certain agile methods were published prior to the Agile Manifesto, but for the purpose of the discussion in this paper, the salient concern is that all agile methods are philosophically grounded in the Agile Manifesto to some extent.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

ICSSP '18, May 26–27, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6459-1/18/05 $15.00
https://doi.org/10.1145/3202710.3203162

software process as "the sequence of steps required to develop or maintain software" [10][2].

It appears therefore that the intended meaning of the term *process* in the agile manifesto may not be consistent with certain *process* definitions (many of which predate the manifesto). It may be the case that the creators of the manifesto intended the term *process* to refer to a *large* or *bureaucratic* process as opposed to simply a *process* (or even a *software process*). A concern of the manifesto creators, one suspects, centres on the potential for an inflexible process to inhibit natural human faculties (such as face-to-face communication and creativity) or to hinder efforts to address innovation. It is nonetheless interesting that the term *process* is positioned as somehow being less valued than other considerations, when clearly if the *process* is "the sequence of steps required to develop or maintain software" there is an apparent disadvantage in devaluing its contribution in a business that is beset with complex and interconnected activities.

A further interesting language observation in relation to the manifesto concerns the increased value associated with "*responding to change over following a plan*". This is interesting because any software development effort that is complex or large or involves a number of team members inevitably requires *planning* in order to achieve economic efficiencies and deliver working software. Again, it is likely that the creators of the manifesto intend a *plan* to mean the sometimes large, medium to long-term and somewhat inflexible software development plans that were (and continue to be) a feature of certain software development approaches. One could purport that ASD involves lots of planning - it is planning for changing requirements rather than planning for static requirements [11]. It therefore seems to be the case that there is some room for improvement in the language adopted in the Agile Manifesto itself – this however does not detract from the many advantages that the manifesto and aligned approaches have conveyed and we must not overlook the significant and positive impact that ASD has had on our field [12].

2 AGILE METHODS

2.1 Methods, Methodologies and Processes

In the context of ASD, it is interesting to note that it appears to today be commonly accepted that agile software processes are referred to as agile *methods* or agile *methodologies* [3], [13]. A *method* "supplies a framework that tells how to go about … [writing software] and identifies the places where creativity is needed" [14] but this *is* a software process (i.e. a framework that tells us how to go about writing software, which places an emphasis on incorporating creativity). As such, the case for introducing *methodologies* or *methods* in place of the established term *process* is debatable, and this extends to their atomic components, often referred to as *practices* but which could perhaps be referred to using various pre-existing terms, for example *tasks* and *activities* [15]. Applying this type of logic, the label *agile software development process* may have been adopted (in place of *agile methods*) in the first instance. Of interest, the software development process was in earlier times referred to as a software *development methodology*, as early as 1963 [16]. The use therefore of the term *methodology* as an alternative label to *process* is also not an invention of the agile movement (though it does not appear to have been in widespread use prior to the advent of ASD).

Even within the agile community, there is some discord over how exactly the family of approaches should be termed, and it has been observed by one of the founding fathers that the terms *method* and *methodology* should be replaced by the term *ecosystem* [17]. Perhaps the inclination to describe the *process* as a *method* or *methodology* or *ecosystem* in the agile domain emanates from the concept that the structure adopted should be of a "barely sufficient" nature [17], containing only as much formal or documented 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 debate on meaning) through use of an alternative label, perhaps: *agile software process*.

2.2 Agility, Rigidity and Discipline

A central innovation of agile methods is the degree of *agility* they support, a point that is well made by Barry Boehm and Richard Turner [18]. This agility relates to an ability to change requirements more frequently, the capacity to resolve client interfacing issues through dialogue rather than litigation, and a focus on producing working software rather than other traditional deliverables such as supporting documentations. However, it seems that drawing a clear distinction between agile and non-agile processes is somewhat problematic, as is demonstrated through examination of the language used to describe this notional dichotomy. The juxtaposition of the terms *Agility* and *Discipline* in the title of Boehm and Turner's work [18] is unfortunate as it carries with it the implicit suggestion that ASD is something that may not be disciplined or which may not require discipline (which of course is not the case, and which one suspects was not intended by the authors). Perhaps an alternative title might have read *Balancing Agility and Rigidity*?

A demand to increase the breadth of agile development methods for the purpose of scalability to large software development enterprises has given rise to what we describe as a set of quasi-agile process frameworks, including the Scaled Agile Framework [19] and the Disciplined Agile Framework [20]. In the case of the latter we see that the term *disciplined* is used to augment the general agile concept, again with the implication that more general agile processes are somehow lacking discipline (when clearly any software development effort that involves groups of individuals demands discipline in order to deliver useful software). Such quasi-agile processes can incorporate some non-

[2] Note that there are many published definitions for the term *software development process* but that this particular one has been identified purely for illustrative purposes.

agile practices and have sometimes been referred to as *hybrid* processes, which it seems are quite common in practice and which are designed to meet the needs of widely varied software development contexts [21].

The challenge of defining a clear dichotomy between agile and non-agile processes is not an issue that is evident only in Boehm and Turner's work. One of the primary advocates for ASD, Jim Highsmith, has employed an equally unsatisfactory juxtaposition when outlining the difference between the two approaches as balancing *Flexibility* and *Structure* [17]. 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.

2.3 Specific Agile Methods

There is considerable variation in the scope of different agile approaches and significant research has been focused on the ASD space [22]. Given the volume and diversity of approaches belonging (or claiming to belong) to the agile family, it is not surprising that it has been observed that there may be an absence of attention to the methodology-independent truths of software development, an effort which ought to be grounded in sound theoretical frameworks that readily enable the evaluation of the *newness* of approaches claiming to offer new conceptual impetus [23], [24].

Efforts to pin down exactly what might be new in ASD reveal that at an atomic practice level, many of the agile techniques predate the agile movement [25], though this same source acknowledges that although there may not be a great deal of newness, the packaging and structure of agile methods has ensured that certain earlier concepts that were perhaps underappreciated prior to ASD have now come to be quite effective, including, short iterations, customer engagement and the frequent delivery of working software. Evaluations of *newness* would be greatly aided through the reuse of existing accepted terminology rather than through the creation of new terms that serve to obfuscate pre-existing conceptual constructs such that assessments of conceptual newness are rendered quite difficult. Indeed, the development of a unified theory underpinning software development would inevitably have to identify and utilise terms in a precise and consistent manner, with the result that new terms would ideally be reserved for genuine instances of newness, such as can only be established through a robust understanding of the history and evolution of the domain.

The advent of ASD has heralded the arrival of a large variety of methods, including Extreme Programming (XP) [2], Adaptive Software Development (ASD) [26], Feature Driven Development (FDD) [27] and Scrum [28]. And while there is compelling evidence to suggest that agile methods have had a significant impact on software development, there are some deficiencies in the currently available evidence surrounding the exact nature and extent of this impact [29]. It has further been suggested that individual constructs may be adopted and adapted from different agile methods depending on the demands of the situational context [30], an observation that legitimises efforts to map the various practices across various agile methods [31].

2.4 New Terminology for Existing Concepts

A *Sprint* is "*an iterative cycle of development work*" [32] and as such, is essentially the same concept as an *iteration* (in Royce's Waterfall [33]) or *cycle* (in Boehm's Spiral [34]). One could therefore legitimately claim that a *sprint* could have been described using a combination of existing terms, perhaps as a *short iteration* and it is not difficult to see how such language use would have benefited the numerous software developers already familiar with the term *iteration*. Of further interest from a terminological perspective, Scrum is generally referred to by its creators as a development *process* [32] or as a *process framework* [35] but not as a *method* or *methodology* even though it is generally classified as being in the ASD family. And the term *sprint retrospective* is essentially equivalent to a *review* meeting wherein the last iteration is evaluated for effectiveness and improvements are proposed for future similar iterations.

The reviewing concept itself is older than software development, dating back at least to Edwards Deming's plan, do, check, act approach [36] and in effect, the term *sprint retrospective* might have been more intuitively accessible to the broader software development community (and beyond to the many interfaces to the development process) if identified using the terminology: an *iteration review* meeting. Clearly Scrum has met with considerable success, nevertheless, Scrum's diversions from a basic terminology perspective is an instance of terminological drift that is perhaps an undesirable feature of software development process terminology.

2.4.1 Agile Requirements.

The term *software requirements* is in use at least as early as 1965 [37] and was possibly commonly adopted for some time prior to that point. *Use Cases* may be utilised when identifying requirements and have been reported to have "fulfilled the role of software requirements well" [38] and within ASD there are 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 ASD, the term *feature* is adopted 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*). FDD adopts a similar convention to ASD, 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*. In both cases, the concept of *function* or *functionality* is likely to resonate somewhat with a *use case*, with individual *features* holding the potential to deliver some value to the customer on a regular basis through short development iterations.

Other agile approaches, for example Scrum and XP, encourage the adoption of *user stories* for requirements identification (though Scrum product backlogs do not insist on the use of user stories [39]). Scrumban [40] provides a new set of definitions again for some pre-existing requirements-related terms, including: a *feature*

is "an atomic use case… the simplest practical expression of: what does the user want?" and a *use case* is a "description of how the product will be used, in the context of the user". Scrumban further asserts that a *feature* is "the minimum testable unit of customer value". Why exactly there may sometimes be a reluctance to work with established definitions (as is the case in Scrumban's treatment of the terms *feature* and *use case*) is difficult to fully qualify but it may be that where process innovations are industry-led, there is a lack of familiarity with academic processes such as literature reviewing and peer-reviewing. It could also be the case that innovators seek to differentiate their contributions from existing or related concepts – perhaps even in cases where there is an absence of meaningful differentiation. In other instances, it may simply be the case that new process architects are unfamiliar with the entirety of the existing software process landscape (a position for which some sympathy is warranted as there is now such a large body of complicated material published in this space).

Returning to the *user story* terminology, it is perhaps most appropriately described as a brief, written description of functionality that will be valuable to either a user or purchaser of a system or software, and which is often accompanied by a rough estimate of the associated implementation effort [41]. *User stories* can be visualised on paper cards, which can be considered to *represent* rather than *document* a software requirement [42] and subsequent dialogue about the paper card story will flesh out the detail via conversation, leading ultimately to confirmation of what exactly is needed to satisfy the *user story* [43]. While *user stories* share some conceptual ground with *use cases*, the two can be considered to be fundamentally different – *use cases* place an earlier focus on larger volumes of documentation which can be subject to maintenance throughout the lifetime of the software [41] – whereas *user stories* (as identified on story cards) tend to be discarded once the story has been dealt with [41]. *User stories*, as adopted in Scrum, are likely to be broadly equivalent to *features* in FDD and ASD.

Whether they be captured as *user stories*, or *use cases*, or *features*, there are benefits to the general agile approach to requirements management, not least the fact that the definition of detailed and sometimes inadequate software requirements specifications which are a feature of many non-agile approaches may be replaced with a flexible, temporal, interactive, and a just-in-time treatment of user requirements [44]. Beyond language considerations, we would like to highlight that within ASD, different approaches to requirements engineering have emerged, for example concerning the extent to which requirements are documented [45].

2.4.2 Agile Roles.

In [46] 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, pure self-organisation may be unworkable in practice, with the theoretical disjoint between work management and process management being difficult to realise in certain Scrum environments where teams may need a team member pushing the workload towards completion [46], [47]. In some cases, the ScrumMaster may tend to naturally assume this authority [48] (though [46] puts this issue down to a failure to implement Scrum correctly). Therefore, in at least some instances, 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 (a role which has existed at least since the 1960s [49]), that their authority should essentially be indirect, with their knowledge and policing of Scrum practices being the limit of their power [46]. This being the case, the traditional *Process Manager* role would appear to overlap greatly with that of a ScrumMaster, especially when the Process Manager role is "to provide information to specialise and instantiate the process model, and to activate and monitor the execution of this instantiated model" [50]. Even in rugby, from which Scrum claims to draw its inspiration in metaphor, there is no *ScrumMaster* (there is a *Scrum Half*, who has varying degrees of authority in terms of calling different pre-planned plays at different times).

4 CONCLUSIONS

In this paper, we have sought out the origin and enduring impact of ASD. We have identified two clear characteristics of agile methods as practiced that distinguish them from earlier approaches. First, the general treatment of requirements tends to be different. Whereas earlier approaches mostly focused on elaborating detailed documented requirements, in ASD requirements may be addressed via discardable user story cards. Second, the insistence on more frequent releases differentiates agile methods from earlier process frameworks. From the 1980s onwards there have been documented efforts to reduce the durations of iterations [34], [51], [52], however, the Agile Manifesto and the methods that have drawn inspiration from it, have dialed up the intensity of the drive towards very short iterations, each producing releasable software.

It may be the case that the classical notion of ASD is somewhat passé. The rise of continuous software engineering [53] (CSE) and the adoption of tooling to continually integrate software and automate deployments may consign many agile methods into history. However, the Agile Manifesto's DNA is evident in these CSE approaches, since the ambition towards "continuous delivery of valuable software" is stated in the first principle of the manifesto (such that CSE could be classified under ASD). The Agile Manifesto itself does show some signs of misfit with emerging practice, especially in its advocacy of "individuals and interactions over processes and tools"; CSE places a central emphasis on the automation of processes via tooling [53]. CSE at least represents a new phase in ASD.

Many agile practices were *in situ* prior to ASD and the insistence on developing new terms for existing concepts was perhaps unhelpful to the broader community. Since it is a generally accepted fact that no single software process is perfectly suited to all software development settings [54], a significant complexity must arise for software practitioners in shaping the

many available process frameworks (agile and non-agile alike) [55], each of which often adopts differing terms. Clearly, however, we must also accept that the wide variety of application domains and development settings may necessarily frustrate attempts to unify terminology for software development as a distinct discipline (each different domain may demand its own terms or term adaptation). Finally, we suggest that certain aspects of the manifesto may not have had the impact that was envisaged, for example, one suspects that contract negotiation remains a fundamental business instrument in many engagements.

ACKNOWLEDGMENTS

This work was supported by Science Foundation Ireland grant 13/RC/2094 to Lero – the Irish Software Research Centre.

REFERENCES

[1] M. Fowler and J. Highsmith, "The Agile Manifesto," Software Development, pp. 28-32, 2001.
[2] K. Beck, Extreme Programming Explained: Embrace Change. Reading, Massachusetts, USA: Addison-Wesley, 1999.
[3] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams and M. Zelkowitz. "Empirical findings in agile methods," Extreme Programming and Agile Methods — XP/Agile Universe 2002, pp. 197-207, 2002.
[4] C. Larman and V. R. Basili, "Iterative and incremental development: a brief history," IEEE Computer, vol. 36, no. 6, pp. 47-56, 2003.
[5] V. R. Basili and A. J. Turner. "Iterative enhancement: A practical technique for software development," IEEE Transactions on Software Engineering, vol. SE-1, no. 4, pp. 390-396, 1975.
[6] I. Jacobson and S. Bylund, The Road to the Unified Software Development Process. Cambridge: Cambridge University Press, 2000.
[7] P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta, Agile Software Development Methods - Review and Analysis. VTT Publications Number 478. Finland: VTT Technical Research Centre of Finland, 2002.
[8] P. Clarke, A. L. Mesquida Calafat, D. Ekert, J. Ekstrom, T. Gornostaja, M. Jovanovic, J. Johansen, A. Mas, R. Messnarz, B. Nájera Villar, A. O'Connor, R. V. O'Connor, M. Reiner, G. Sauberer, K. D. Schmitz and M. Yilmaz, "An investigation of software development process terminology," Proceedings of the 16th International SPICE Conference, pp. 351-361, 2016.
[9] G. Sauberer et al., "Do we speak the same language? terminology strategies for (software) engineering environments based on the elcat," Systems, Software and Services Process Improvement. EuroSPI 2017. Communications in Computer and Information Science, Vol 748, pp. 653-666, 2017.
[10] W. S. Humphrey, A Discipline for Software Engineering. Reading, Massachusetts, USA: Addison-Wesley, 1995.
[11] M. Paulk, "Agile Methodologies and Process Discipline," Crosstalk – the Journal of Defense Software Engineering, vol. October, pp. 15-18, 2002.
[12] T. Dyba and T. Dingsoyr. "Empirical studies of agile software development," Information and Software Technology, vol. 50, no. 9-10, pp. 833-859, 2008.
[13] J. A. Highsmith, Agile Software Development Ecosystems. Boston: Addison-Wesley, 2002.
[14] J. Rumbaugh, "What is a method?" Journal of Object Oriented Programming, vol. 8, no. 6, pp. 10-16, 1995.
[15] ISO/IEC, ISO/IEC 12207:1995 Information Technology – Software Life-Cycles Processes. Geneva, Switzerland: ISO, 1995.
[16] J. J. Connelly and Y. R. Osajima, Management Report: Controlling Production of Complex Software. Technical Memorandum TM-LO-810/101/00. Santa Monica, CA: System Development Corporation, 1963.
[17] J. Highsmith, "What is agile software development?" Crosstalk – the Journal of Defense Software Engineering, vol. 15, no. 10, pp. 4-9, 2002.
[18] B. Boehm and R. Turner, Balancing Agility and Discipline - A Guide for the Perplexed. Boston, Massachusetts, USA: Pearson Education Limited, 2003.
[19] D. Leffingwell, "Scaled agile framework," http://www.scaledagileframework.com/.
[20] S. Ambler, "The disciplined agile (DA) framework," http://www.disciplinedagiledelivery.com/blog/.
[21] Hybrid software and system development in practice: Waterfall, scrum, and beyond, "Kuhrmann, M.; diebold, P.; munch, J.; tell, P.; garousi, V.; felderer, M.;McCaffrey, F.; trektere,
K.; linssen, O.; hanser, E.; prause, C.R." Proceedings if the International Conference on Software and Systems Processes, pp. 30-39, 2017.
[22] T. Dingsøyr, S. Nerur, V. Balijepally and N. B. Moe. "A decade of agile methodologies: Towards explaining agile software development," J. Syst. Software, vol. 85, no. 6, pp. 1213-1221, 2012.
[23] P. Johnson, M. Ekstedt and I. Jacobson. "Where's the Theory for Software Engineering?" IEEE Software, vol. 29, no. 5, pp. 96-96, 2012.
[24] K. J. Stol and B. Fitzgerald. "Uncovering theories in software engineering," 2nd SEMAT Workshop on a General Theory of Software Engineering, pp. 5-14, 2013.
[25] N. Abbas, A. M. Gravell and G. B. Wills. "Historical roots of agile methods" Agile Processes in Software Engineering and Extreme Programming, pp. 94-103, 2008.
[26] J. Highsmith, Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York, USA: Dorset House Publishing, 2000.
[27] S. R. Palmer and J. Felsing, A Practical Guide to Feature-Driven Development. Upper Saddle River, New Jersey, USA: Prentice Hall, 2002.
[28] K. Schwaber and M. Beedle, Agile Software Development with SCRUM. Upper Saddle River, New Jersey, USA: Prentice Hall, 2002.
[29] S. Stavru. "A critical examination of recent industrial surveys on agile method usage," J. Syst. Software, vol. 94, pp. 87-97, 2014.
[30] P. Clarke and R. V. O'Connor. "The situational factors that affect the software development process: Towards a comprehensive reference framework," Journal of Information and Software Technology, vol. 54, no. 5, pp. 433-447, 2012.
[31] P. Diebold and M. Dahlem. "Agile practices in practice: A mapping study," Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1-10, 2014.
[32] K. Schwaber, "SCRUM development process," 10th Annual Conference on Object-Oriented Programming Systems, Languages and Applications, 1995.
[33] W. Royce, "Managing the development of large software systems: Concepts and techniques," Western Electric show and Convention Technical Papers, 1970.
[34] B. Boehm. "A spiral model of software development and enhancement," IEEE Computer, vol. 21, no. 5, pp. 61-72, 1988.
[35] K. Schwaber and J. Sutherland, The Scrum Guide™. Scruminc, 2013.
[36] W. Edwards Deming, Elementary Principles of the Statistical Contrl of Qualiry. Tokyo, Japan: Nippon Kagaku Gijutsu Renmei, 1950.
[37] W. F. Bauer and E. K. Campbell, Advanced Naval Tactical Command and Control Study (Informatics Report TR-65-58-2). Prepared for Advanced Warfare Systems Division, Naval Analysis Group, Office of Naval Research by Informatic Inc., 1965.
[38] D. Kulak and E. Guiney, Use Cases: Requirements in Context. Boston, MA: Addison-Wesley, 2004.
[39] P. Deemer, B. Vodde, C. Larman and G. Benefield, "Scrum primer: A lightweight guide to the theory and practice of scrum," http://www.scrumprimer.com/;. 2015.
[40] C. Ladas, Scrumban: Essays on Kanban Systems for Lean Software Development. Seattle, WA: Modus Cooperandi Press, 2008.
[41] M. Cohn, User Stories Applied: For Agile Software Development. Boston, Mass.: Addison-Wesley, 2004.
[42] R. Davies, The Power of Stories. Practitioners Report / Poster Presentation at the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2001). 2001.
[43] R. Jeffries, "Essential XP: Card, Conversation, and Confirmation," XP Magazine, vol. August 30, 2001.
[44] D. Leffingwell, Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. NJ: Pearson Education, 2010.
[45] F. Paetsch, A. Eberlein and F. Maurer. "Requirements engineering and agile software development," Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.
[46] K. Schwaber, Agile Project Management with Scrum. WP Publishers & Distributors Pvt Limited, 2004.
[47] M. Cristal, D. Wildt and R. Prikladnicki. "Usage of SCRUM practices within a global company," IEEE International Conference on Global Software Engineering, 2008, pp. 222-226, 2008.
[48] N. B. Moe, T. Dingsoyr and T. Dyba. "Overcoming barriers to self-management in software teams," IEEE Software, vol. 26, no. 6, pp. 20-26, 2009.
[49] M. M. Jones and E. McLean, "Management problems in large-scale software development projects," Industrial Management Review, vol. 11, pp. 1-15, 1970.
[50] R. Conradi, C. Fernström, A. Fuggetta and R. Snowdon. "Towards a reference framework for process concepts," Software Process Technology. Proceedings of the Second European Workshop, EWSPT '92, pp. 1-17, 1992.
[51] C. Larman, Agile and Iterative Development: A Manager's Guide. Boston: Addison-Wesley, 2004.
[52] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process. Reading, Massachusetts: Addison Wesley Longman, Inc., 1999.
[53] R. V. O'Connor, P. Elger and P. Clarke, "Continuous Software Engineering - A Microservices Architecture Perspective," Journal of Software: Evolution and Process, vol. 29, no. 11, pp. 1-12, 2017.
[54] P. Clarke, R. O'Connor, B. Leavy and M. Yilmaz. "Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance," IEEE Transactions on Software Engineering, vol. 41, no. 12, pp. 1169-1183, 2015.
[55] Clarke, P., O'Connor, R. V., Leavy, B., "A complexity theory viewpoint on the software development process and situational context," Proceedings of the 2016 International Conference on Software and System Process (ICSSP 2016), 2016.