The impact of situational context on the software development process – A case study of a highly innovative start-up organization

Gerard Marks[1], Rory V. O'Connor[2, 3], Paul M. Clarke [2, 3]

[1] Optimality Technologies, Invent, DCU, Dublin, Ireland
[2] Dublin City University, Dublin, Ireland
[3] Lero, the Irish Software Engineering Research Centre, Ireland
Gerard.Marks@optimalitytech.com, Rory.OConnor@dcu.ie, Paul.M.Clarke@dcu.ie

Abstract. Over the past six years, we have examined the impact of situational context of the software development process. Our early work involved the systematic development of a comprehensive situational factors reference framework. More recently, our efforts have focused on the application of this reference framework to different types of situational context. In this latest in a series of case studies, we examine the case of a small start-up organization, exploring in detail the process adopted. We also undertook a detailed evaluation of the situational context, carefully identifying the situational factors of greatest importance and how these factors have influenced the process design. The outcome of our case study confirms our earlier finding that a software development process is highly dependent on the organizational context. We also discovered some interesting new themes in this start-up environment, including the difficulty associated with prioritizing situational factors and the complexity that surrounds software process design. The role of organizational learning and feedback into improved development processes is also presented as a critical feature.

Keywords: Software Development Process; Software Development Context; Agile; Lean; Process Selection

1 Introduction

Although many valuable software development models, methods and standards have been created, it remains the case that attempts to identify a universally optimal approach to software development have been frustrated by the variation that presents in software development contexts [1]. Added to the challenge introduced by this this variation, we also find that no situational context is unchanging [2], with the result that process adaptation is inevitably required. These observations in relation to the software development process seem likely to meet with the agreement of seasoned software development researchers and practitioners. However, while agreement on the position may arise, the authors have suggested that the solution to the problem of harmonizing a process with a context is highly complex, in fact it would appear to be an instance of a complex adaptive system [3]. In pursuit of an improved understanding of this complex interplay between a software development process and its situational context, we have assigned high importance to the evaluation of individual situational contexts and corresponding processes [4]. Accordingly, some of our related work has examined the problem in a high-growth small to medium sized organization applying a microservices architecture for rapid product evolution [5, 17], and also in safety critical software development environments, including medical device and nuclear power domains [6].

In the case study reported upon in this paper, we focus our examination on a further development setting. This time, we examine the software development process adopted by a high potential growth firm that operates in the specialized database performance and interoperability niche. This firm has grappled with the challenge of satisfying the predictability demands of mission-critical data-intensive systems while simultaneously battling the survival concerns which are all too often a reality of small start-up organizations. Through examining the situational context and software development process in the case study organization, we identify the key constraints that have focused the software development process enactment. Together with earlier findings, this knowledge is helpful in building up a portfolio of context-to-process relationships.

While our work has proven to be iterative and slow in nature, it has a number of important benefits. Firstly, it can help us to better appreciate the relationships and dimensions that comprise this complex challenge. Individual organizations seeking an objective reflection on their software development process can tap this resource as an aid to self-evaluation. Secondly, over time, the development of a suite of case studies can identify commonalities and differences in different types of settings (and the impact this has on the development process), thereby collectively helping to reduce the process-to-situational-context harmonization challenge.

Section 2 outlines the situational factors framework; Section 3 presents an overview of the company studied, including its software development process; Section 4 examines the role of situational context; and finally, Section 5 presents a discussion and conclusion.

2 Situational Factors

Since at least 1992 (and probably much earlier) the importance of situational context as an informant of the software development process has been acknowledged [7]. Although published resources advocate that an “organization’s processes operate in a business context that should be understood” [8] and that a “life cycle model… [should be] appropriate for the project's scope, magnitude, complexity, changing needs and opportunities” [9], we suggest that there remains a significant lack of guidance on exactly how companies might adapt their process to their (changing) situational context. Software development necessarily occurs in a development context, which includes a large number of concerns and factors [10], [11] with this context being pivotal in understanding what works for whom, where, when, and why [12]. In support of the importance of understanding the instructional function of situational contexts, authors such as Dyba [13] highlight that the dependence on a potentially large number of situational factors is of itself an important reason for why software engineering is so hard

Despite the various references to the importance of situational context in the literature, it was the lack of a comprehensive situational factors framework for software development that led two of the authors to produce and publish an initial reference framework [14], itself an amalgamation of earlier important contributions, from multiple areas such as software risk estimation, cost models for software engineering, capability maturity frameworks, etc.

The framework incorporates 44 individual factors (refer to Figure 1) classified under 8 categories (refer to Table 1), which are further elaborated as 170 underlying sub-factors. A sample listing of the sub-factors in the *Personnel* classification is presented in Table 2, with comprehensive details of the framework available in previously published material [14].

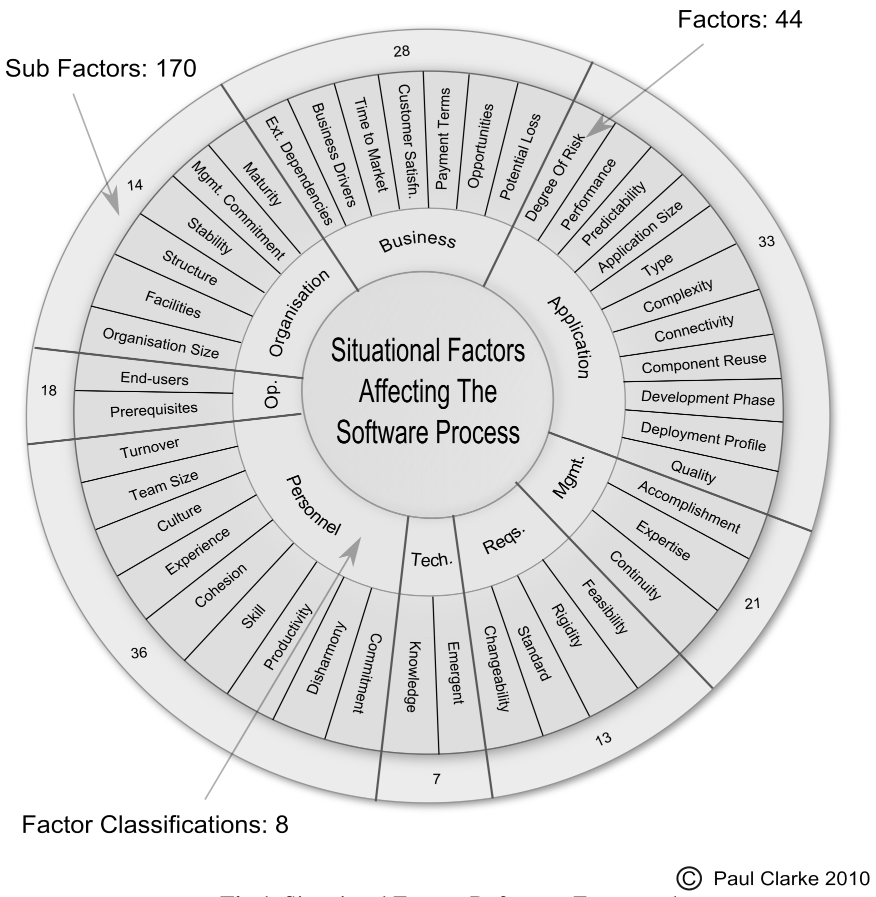

Fig 1. Situational Factors Reference Framework

The authors believe that the Situational Factors Reference Framework presented above is currently the most comprehensive reference framework of the situational factors that affect the software development process. As such, the framework holds significant value for researchers and practitioners, as researchers will have access to a broad, systematically developed initial framework (replete with 170 subfactors), which can be used as a reference for the situational factors affecting the software development process. Further this situational factors reference framework can be used as a stepping stone towards greater understanding of the complexity of software development settings, and the systematic approach adopted in its creation from a rich and detailed set of sources has given rise to a framework that we consider to outline a broadly informed reference for the software development community [4].

Table 1. Situational Factors Classification

Classification	**Description**
Personnel	Constitution and characteristics of the non-managerial personnel involved in the software development efforts.
Application	Characteristics of the application(s).

Technology	Profile of the technology being used for the software development effort.
Organization	Profile of the organization.
Operation	Operational considerations and constraints.
Management	Constitution and characteristics of the development management team.
Business	Strategic / tactical business considerations.
Application	Characteristics of the application(s).

Using the framework, the situational factors affecting the software process were investigated in practice in the case study start-up organization, details of which are presented in the following sections.

Table 2. Personnel Factors & Sub-Factors

Factor	**Sub-Factor**
Turnover	Turnover of personnel
Team size	(Relative) team size
Culture	Team culture/resistance to change
Experience	General team experience / diversity/ ability to understand the human implications of a new information system/team ability to work with management/application experience/analyst experience/programmer experience/tester experience/experience with development methodology / platform experience.
Cohesion	General cohesion/team members who have not worked for you/team not having worked together in the past/team ability to successfully complete a task/team ability to work with undefined elements and uncertain objectives / overdependence on team members / distributed team/ team geographically distant.
Skill	Operational knowledge/team expertise (task) / team ability to work with undefined elements and uncertain objectives/training development.
Productivity	Team ability to carry out tasks productively.
Commitment	Commitment to project among team members.
Disharmony	Interpersonal conflicts.
Changeability	Scope creep/continually changing system requirements/ill-defined project goals / gold plating/unclear system requirements.

3 Case Study Company

Optimality Technologies is a spin-out company from Dublin City University (DCU), founded in July 2015. Prior to the formation of Optimality, a sustained research effort into XML query performance (focusing largely on data indexing and retrieval techniques and data processing algorithms) was achieved through the DCU-based Database Performance & Migration Group (DPMG). From its earliest roots as the DPMG, Optimality prioritized industrial engagement as a means to test assumptions and potential solutions.

During the lifetime of the DPMG, the sustained focus was the development of data-related products that reduced the complexity and risk associated with modernizing database infrastructure. Ultimately, a series of industrial and research engagements pointed to single, significant challenge: Could the database layer be optimized whilst preserving the application code base? If this challenge could be overcome in an economically-viable manner, there was a major immediate benefit for organizations: it would be possible to modernize the database infrastructure with zero application code rewrites. This became the focus of Optimality's efforts and over time, resulted in the development of a sophisticated tool set and associated interfacing software product, which has been applied to the task of database modernization. Perhaps the two most important advantages of this approach can be summarised as:

- **Reduced solution development time**. The result of zero application code rewrites is a significant reduction in the development effort.
- **Reduced risk of software failures**. Since the application-level test suites remain valid, it is possible to quickly detect any issues with the database modernization/migration effort.

These two advantages increase the viability of database modernization for organizations, and this attracted the attention of leading global financial software product and service providers. In some cases, these organizations have harnessed the Optimality tool set to evaluate alternative or emerging database technologies prior to making large investment decisions for production-grade products and services. No matter the size of the organization with which Optimality engages, the constant challenge is to harmonize the objectives of strict correctness (in data terms) and reliability (in terms of solution stability) with the need for high-paced innovation (sometimes involving emerging technology). To address this context, Optimality has adopted and refined a software development process that is particular to their organization

3.1 Process Overview

Figure 2 illustrates the process lifecycle from the initial customer engagement through to an iterative system elaboration process, with further details of the individual steps being as follows:

Initial Customer Engagement

- **Secure Contract**. New business acquisition.
- **High Level Requirements**. Evaluate the client's high level requirements and formulate a specification document along with projected milestones, deliverables and payment terms (which may be time and materials based, or fixed price). Since there is high variability in existing client systems and objectives related to innovation, it is not possible to fully elaborate requirements at this stage.
- **Customer Sign-off**. Once the customer has signed-off on the requirements and terms, work can begin.
- **Establish Initial Benchmark**. Performance considerations are key aspect of the work. Therefore, a specified benchmark system captures performance metrics prior to the implementation of any solution implementation effort.

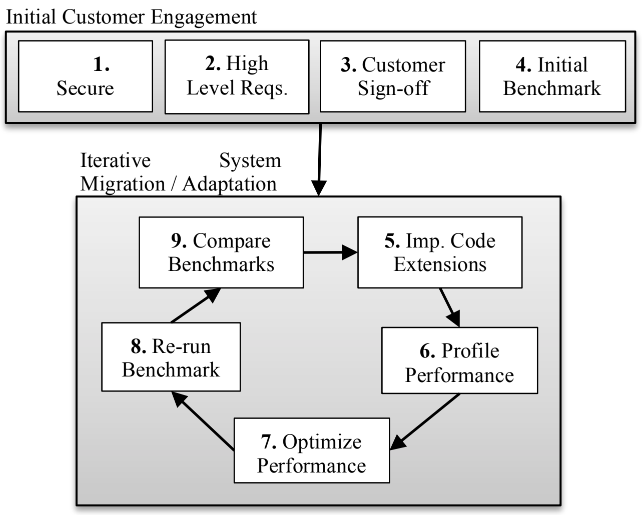

Fig 2. Optimality – High level Software Process Lifecycle

Iterative System Migration / Adaptation

- **Implement Code Extensions**. If required, extensions to the Optimality tool set are implemented to enable the migration process (e.g. providing coverage for a new query language).
- **Profile Performance**. Evaluate the performance constraints and targets. Where appropriate, identify the most attractive cost-benefit work packages.
- **Optimize Performance**. Involves tuning the target database and Optimality's processing engine to ensure that performance is maximized.
- **Re-run Benchmark**. Rerun the benchmark to examine impact on performance and if required, confirm (using application-level tests) that migration effort has been successful.
- **Compare Benchmarks**. Evaluate the results of the benchmark, and liaise with the customer to determine if subsequent migration / adaptation iterations would be beneficial.

Table 3 provides an overview of the typical durations for each step of the process. Note that there is variance for each step duration, which allows for some small rapid changes to be introduced into a formal evaluation cycle or for longer iterations where changes are either more certain in nature or larger in size.

Table 3. Estimated Process Duration Overview

Process Name	**Duration (Days)**
Implement Code Extensions	0 - 60
Profile Performance	2 - 10
Optimize Performance	2 - 60
Re-run Benchmark	1 - 5

Compare Benchmarks	1 - 3

3.2 Testing and Quality

In the earliest stages, the objective was to simply morph the Optimality tool set into whatever the client demanded. Gradually, this led to the development of an automated Extract, Transform, Load (ETL) process, whereby once a query is received, it is redirected to the new data model (or database) and will reconstruct the result set into the format expected by the application layer. However, in enabling this automated interaction, constraints in relation to coverage and quality must also be satisfied.

3.2.1 Coverage

Optimality provides an *SQL-to-X* service where *X* can be: (1) a new data model within the same database, (2) another relational database, or (3) a *NoSQL* or *NewSQL* database. As a result, multiple dialects of SQL (e.g. Oracle, SQL Server, MySQL) must be supported which necessitates the need for a dual/hybrid database.

In the dual database scenario, a runtime *query routing* service enables a subset of tables to be migrated to the new database/model, while all others are routed to the original system, thereby allowing the fully functional software application to be redeployed in a very short time period. Since this approach can quickly isolate critical solution viability information, it has proven to be very effective in supporting the type of proof of concept required by many clients.

3.2.2 Quality

High data quality is a critical requirement for many database intensive systems, especially in sectors such as Finance. To satisfy this constraint, a number of quality related techniques were injected or emphasised in the software development process, including:

- Core algorithms formally verified at the theoretical level.
- Core functionality subject to robust unit testing.
- Continuous integration is adopted to protect against overall quality degradation.

Collectively, and although costly, the insistence on the adoption of these three techniques adequately addressed quality considerations. With very few exceptions, unit tests are written prior to the code itself being written. In the early stages, standalone unit tests were written for each core piece of functionality (a query transformation, for example). However, it became apparent that continuous integration (whereby test data is re-generated each time and queries are tested against each of the supported databases) was required.

3.2.3 Automating Continuous Integration

As a final degree of integrity checking, an automated *Integrity Checker* was developed. Given that the dual database approach was adopted to allow for iterative migration lifecycles, it is possible to execute the 'original' query against the 'original' database and the 'translated' query against the 'new' database and byte-compare the results at runtime (i.e. the process is entirely automated). Therefore, the Integrity Checker provides (1) a way for end users to validate the correctness of the system against multiple sources of *test data*, and (2) a means for end users validate the system against actual *production data* (at runtime). Together with other innovations such as the automated ETL process, the Integrity Checker effectively automates the creation of continuous integration tests. Were it not for this advance form of automation, it would not be possible to sustain the pace of development while also satisfying the quality constraints.

4 Applying the Situational Factors Reference Framework

Two researchers in association with the Managing Director from Optimality analyzed the company's situational factors, the outcome of which was a listing of the dominant contextual factors affecting Optimality's software development process (refer to Table 4).

Table 4. Situational Factors Identified in Case Study

Category	**Factors Identified in Case Study**
Personnel	**Skill:** Given the very high application and programming skill of both primary engineers, the team had a high velocity while also maintaining high quality – plus the start-up cost in terms of personnel on-boarding was low.
Requirements	**Changeability**: Many requirements would only became clear through a sustained prototyping-type effort. Therefore, an agile / rapid prototyping approach was well suited to the nature of requirements.
Application	**Quality**: There is a strict requirement for accuracy (i.e. high quality) of query-related tasks. This factor was a motivator for adopting test driven development (TDD) and continuous integration (CI);
Application	**Performance**: There was a significant requirement for very high performance from the Optimality software and as a result, regular investments in refactoring were needed in order to streamline performance;
Application	**Complexity**: The high volume and complexity of data queries raised the complexity of the application overall. TDD and CI were instrumental in raising confidence that the complexity did not compromise the application quality;
Application	**Predictability**: Given the sometimes rapid pace of functional deliveries, a lean / agile software development philosophy was adopted. As the extent of recent changes could be high, the need for a process offering both robust refactoring and TDD/CI was very high;
Application	**Type**: A low tolerance for data inaccuracy influenced the decision to implement a robust TDD and CI infrastructure. The factor also had a direct impact on the software architecture. To permit 3rd parties to address different

	aspects of overall system functionality, parallelization allowed other systems to handle certain concerns.
Operational	**End-Users**: End-users in this case were expecting responsiveness from their software supplier in pursuit of competitive advantages in a fast moving market. This fact is key in shaping much of the process design – which is capable of addressing rapidly changing requirements.
Technical	**Emergent**: Aspects of the technology stack were emergent (e.g. the Datomic and MongoDB databases). A responsive / agile software process was desirable.
Organizational	**Size**: Given that the organization comprised (on a full time basis) of between one and two highly specialized, post-Doctoral and close-working engineers, the need for documentation as a means for internal communication was very low.
Business	**Business Drivers**: Being a small start-up organization, the pressure to manage finances and minimize costs was high. As a result, the use of technology solutions for quality (e.g. TDD and CI) was preferred to human solutions (which also serviced the demand to quickly deliver high quality software on a continual basis);
Business	**Payment Arrangements**: In many cases, fixed price contracts were secured with the result that the motivation to adopt a minimal scope delivery was increased;
Business	**Magnitude of Potential Loss**: Since inaccurate queries can result in inaccurate calculations and information, the magnitude of potential loss for low quality software was potentially financially very high. To address this factor, large investments in TDD / CI. Plus, the architectural decision to adopt a dual/hybrid database solution had a major impact in de-risking potential software issues;
Business	**Customer Satisfaction**: Given the profile of clients as large financial services IT provided, the quality of the application had to consistently very high. TDD and CI in the software process contributed to realizing this confidence and quality.

5 Discussion

In the case of Optimality, we witness a common theme in software development process decision making: a complex set of dominant and sometimes interrelated situational factors need to be accommodated in the software development process, thus any individual software development process decision may (and perhaps ideally should) deliver positive benefits for multiple situational constraints (ref. to Table 3). This type of approach can be considered favorable in the context of complex adaptive systems, wherein a cocktail of interrelated concerns continually interact and we seek to derive the optimal holistic outcome across all concerns when adapting a process. As we have argued in the past, the relationship between a software development process and its situational context would appear to be an instance of a complex adaptive system [3] and therefore, discovering the type of process thinking that we have revealed in Optimality further supports this observation.

Evidence in support of the role of learning in buttressing improved process adaption was observed in the Optimality case study. While the company had a tendency for aggressive product innovation with an ultra-lean approach to feature delivery, the experience gained with this approach demonstrated that in practice, the cost of refactoring (which was an absolute necessity given the product quality and cost-base constraints) mounted over time. Accordingly, the company had to adapt their process so as to improve their ability to implement better product architecture and design early in development cycles so as to strike an improved economic balance. This is interesting as it represents a regression from lean/agile thinking back to more traditional approaches.

With the variability in iteration durations quite high and the need to reduce risk in relation to the impact of low quality, it is perhaps the case that the Optimality process, while being agile, is also resonant with Boehm's spiral model [15]. And while Optimality may notional consider their process to be agile / lean, the significant variation in iteration durations (from a total of 5 days to >130 days) and the heavy burden that can be placed on a small team in a start-up environment, may run contrary to the agile principle: "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely" [16]. Clearly, a constant pace of development is difficult to establish where there is not a constant pace of requirements identification, and while a sustainable pace is a worthy goal, there remain segments of the software development community who continue endure long and unpredictable working hours. Perhaps when economics and human nature collide with worthy ideals, there will always be a battle to be waged.

Acknowledgments: This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 to Lero, the Irish Software Engineering Research Centre (www.lero.ie) and Enterprise Ireland grant CF20133605.

References

1. 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.
2. R. V. O'Connor and P. Clarke. "Software process reflexivity and business performance: Initial results from an empirical study," Proceedings of the 2015 International Conference on Software and System Process, pp. 142-146, 2015.
3. 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.
4. P. Clarke and R. V. O'Connor, "Changing situational contexts present a constant challenge to software developers," Proceedings of the 22nd European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2015), CCIS (Vol. 543), pp. 100-111, 2015.
5. R. V. O'Connor, P. Elger and P. Clarke. "Exploring the impact of situational context: A case study of a software development process for a microservices architecture," Proceedings of the International Conference on Software and Systems Process (ICSSP '16), pp. 6-10, 2016.

6. R. Nevalainen, P. Clarke, F. McCaffery, R. V. O'Connor and T. Varkoi. "Situational factors in safety critical software development," Proceedings of the 23rd European Conference on Systems, Software and Services Process Improvement, EuroSPI 2016, Graz, Austria, September 14-16, 2016, pp. 132-147, 2016.
7. P. Feiler and W. Humphrey, Software Process Development and Enactment: Concepts and Definitions. CMU/SEI-92-TR-004. Pittsburgh, Pennsylvania, USA: Software Engineering Institute, Carnegie Mellon University, 1992.
8. SEI, CMMI for Development, Version 1.3. CMU/SEI-2006-TR-008. Pittsburgh, PA, USA: Software Engineering Institute, 2010.
9. P. Clarke, R. V. O'Connor and M. Yilmaz, "A hierarchy of SPI activities for software SMEs: Results from ISO/IEC 12207-based SPI assessments," Proceedings of the 12th International Conference on Software Process Improvement and Capability dEtermination (SPICE 2012), pp. 62-74, 2012.
10. L. McLeod and S. MacDonell. "Factors that affect software systems development project outcomes: A survey of research," ACM Comput.Surv., vol. 43, no. 4, pp. 24:1-24:56, 2011.
11. W. J. Orlikowski and J. J. Baroudi. "Studying Information Technology in Organizations: Research Approaches and Assumptions." Information Systems Research, vol. 2, no. 1, pp. 1-28, 1991.
12. T. Dyba. "Contextualizing empirical evidence," IEEE Software, vol. 30, no. 1, pp. 81-83, 2013.
13. T. Dyba, D. I. K. Sjoberg and D. S. Cruzes. "What works for whom, where, when, and why?: On the role of context in empirical software engineering," Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 19-28, 2012.
14. 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.
15. B. Boehm. "A spiral model of software development and enhancement," IEEE Computer, vol. 21, no. 5, pp. 61-72, 1988.
16. M. Fowler and J. Highsmith, "The Agile Manifesto," Software Development, pp. 28-32, 2001.
17. R. V. O'Connor, P. Elger and P. Clarke, "Continuous software engineering—A microservices architecture perspective." *Journal of Software: Evolution and Process* (2017).