Marks, G., O'Connor, R., Yilmaz, M. and Clarke, P., An ISO/IEC 12207 Perspective on Software Development Process Adaptation, Software Quality Professional, Vol. 20, No. 2, 2018.

An ISO/IEC 12207 perspective on software development process adaptation

Gerard Marks, Rory V. O'Connor, Murat Yilmaz, Paul M. Clarke

Abstract
n their earlier work, the authors had a sustained engagement with situational factors affecting software development, particularly how these factors affect the software development process. Part of this previous engagement involved the development of a situational factors reference framework. As part of an ongoing industrial engagement, the authors are currently examining situational factors and software development processes in a series of case studies. This latest case study is concerned with a small start-up organization. They start by identifying the software development process in this organization. Thereafter, the authors examine the situational context of the company, leading to an analysis of the relationship between the process and the situational context. Their general findings are consistent with their previous related work, supporting the case that a software development process is dependent on the organizational context, perhaps in a highly complex manner. In this particular case study, the authors also find that the role of organizational learning and process adaption is considered to be central to organizational survival.

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

1 Introduction

While various software development models, methods, and standards have been advocated, attempts to identify a universally optimal approach to software development have been thwarted by the variation that presents in software development contexts (Clarke et al. 2015). Added to the challenge introduced by this variation, the authors also and that situational contexts are volatile (O'Connor and Clarke 2015), with the result that process adaptation is inevitably required. These observations in relation to the software development process may meet with the agreement of experienced software development researchers and practitioners. However, the authors have suggested that 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 (Clarke, O'Connor, and Leavy 2016). In pursuit of a better understanding of this complex interplay between a software development process and its situational context, the authors assign high importance to the evaluation of situational contexts and their corresponding processes (Clarke and O'Connor 2015). Accordingly, some of their related work has examined the problem in a high-growth small to medium-sized organization applying a microservices architecture for rapid product evolution (O'Connor, Elger, and Clarke 2016), and also in a safety-critical software development environments, including medical device and nuclear power domains (Nevalainen et al. 2016).

In the case study reported upon herein, we focus our investigation on a new development setting. This time, we examine the software development process in a high potential growth organisation that operates in the specialized database performance and interoperability domain. This firm has worked with the challenge of satisfying the predictability demands of mission-critical data-intensive systems while concurrently battling with 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 this organization, we identify the key factors that have influenced the software development process implementation. Together with earlier studies, this knowledge is helpful in building up a portfolio of context-to-process relationships.

While our work has proven to be time consuming, it has a number of important benefits. Firstly, it can help us to better understand the relationships and dimensions that comprise this complex challenge. Organizations seeking an objective reflection on their software development process can reference this resource as an aid to self-evaluation. Secondly, the development of a suite of case studies can identify similarities and differences in different settings (and the impact this has on the development

process), thereby collectively holding the potential to reduce the process-to-situational-context harmonization challenge.

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 (Feiler & Humphrey, 1992). Although published resources advocate that an "organization's processes operate in a business context that should be understood" (SEI, 2010) and that a "life cycle model… [should be] appropriate for the project's scope, magnitude, complexity, changing needs and opportunities" (P. Clarke, O'Connor, & Yilmaz, 2012), 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 (McLeod & MacDonell, 2011; Orlikowski & Baroudi, 1991) with this context being pivotal in understanding what works for whom, where, when, and why (Dyba, 2013). In support of the importance of understanding the instructional function of situational contexts, authors such as Dyba (Dyba, Sjoberg, & Cruzes, 2012) 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 (P. Clarke & O'Connor, 2012), 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 (P. Clarke & O'Connor, 2012).

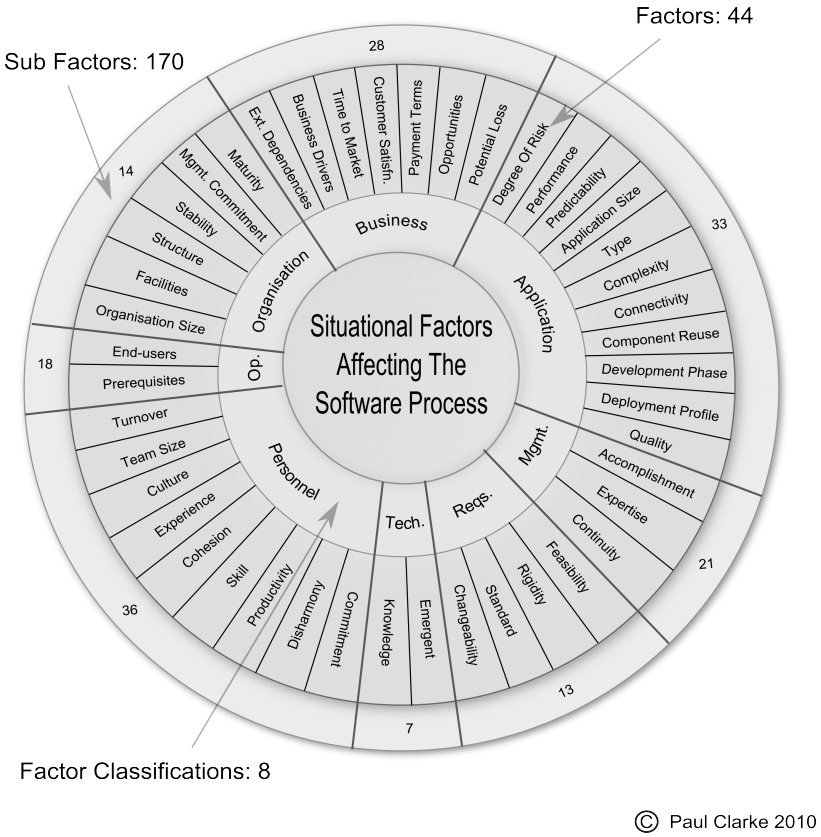

Fig 1. Situational Factors Reference Framework

The authors consider the situational factors reference framework to be 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 (P. Clarke & O'Connor, 2015). 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 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.
Requirements	Characteristics of the requirements.

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.

Case Study Company

Optimality is a company that delivers user-friendly interfaces to SQL and NoSQL databases. For example, Optimality's SQL interface to MongoDB reduces the complexity of accessing data in MongoDB (similar to the way that Hive reduces the complexity of implementing MapReduce jobs). That said, Optimality originally started out with the goal of retrospectively optimizing the performance of software applications with zero code rewrites; that is, optimizing the performance of software applications in a cost-effective and safe way.

The main challenge that is encountered when one is developing a product that aims to optimize software applications at the data layer whilst keeping the existing code is that access to the application's bespoke business logic is restricted. This means that automation, while effective in many cases, can only be effective up to a point; after which human input is necessary in order to realise the full scope of knowledge of the business logic (to achieve maximum performance gains). This leads to a major difficulty in the productizing process as it is difficult to scale a product if a human is required (at any point).

In an effort to overcome this human impediment to the development of an off-the-shelf product, Optimality provided a powerful plug-in framework that would allow end-end users to apply their own business logic knowledge directly to the query transformation process at runtime, thus maximizing the performance benefits while still preserving the existing application code. This powerful ability enables Optimality users to make changes to the underlying database schema above and beyond those that could be made if there was reliance on automation alone. However, customer engagements quickly highlighted the fact that it is unlikely that the end-users would be able to use the plug-in framework alone and ultimately, it would require Optimality consultants to understand the existing business logic in each case as a prerequisite to implementing the plugins as part of the optimization process.

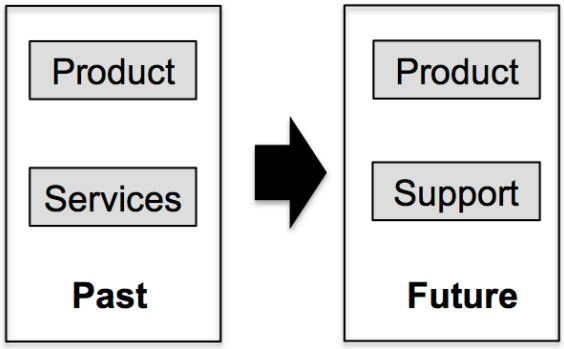

Fig 2 - Towards a Scalable Business Model

Exasperated by the perpetual reliance on the services that were required as part of the sales proposition, Optimality decided to shift its focus to a business model that was not dependent on a services element. To achieve this, it was decided that Optimality needed to become a product that users could download and trial completely independently of Optimality consultants, as illustrated in Figure 2. That is, to develop a scalable product that could be sold off-the-shelf.

To achieve the goal of developing a scalable business model, Optimality reverted to an idea that it prototyped many years before. That is, to provide a user-interface to existing databases, but this time the interface could be used during development; in contrast to their middleware solution for optimizing applications. In doing so, it would be possible to reduce the scope of the offering so that it could be sold as a standalone product while still providing significant value to customers; later, the feature set could be extended over time adding more value for clients. This led to the initial standalone product offering: The MongoJDBC Driver.

Original Process Overview

In this section, an introduction of the original (product with services lifecycle) is provided. In contrast, later, in this section the process lifecycle of the new scalable business model is introduced. Figure 3 illustrates the original 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.

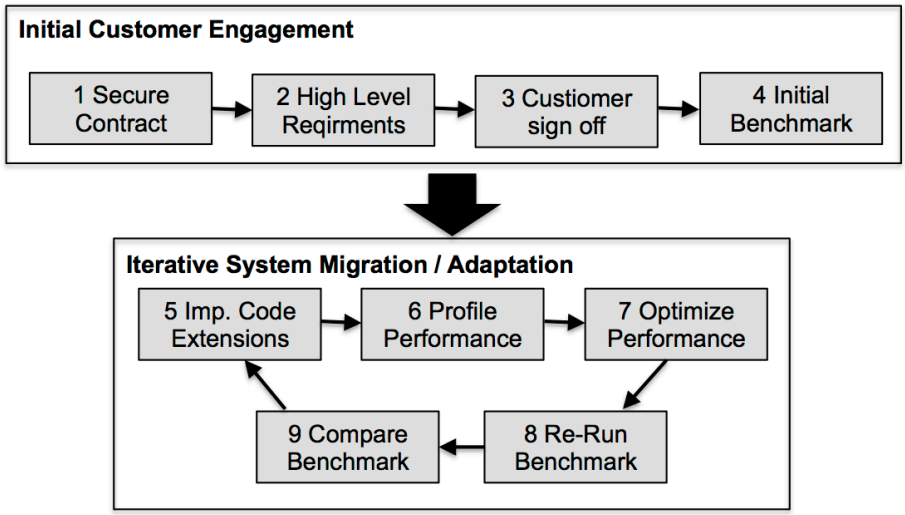

Fig 3. Optimality – High level Software Process Lifecycle

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 if required.

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

Challenges Associated with Current Process

The process described in the previous section provides some significant challenges for a small company. In particular:

1. **Employee Acquisition**. In a small company such as Optimality, it is difficult to hire staff that are capable of working in the highly specialized performance domain; mainly due to cost considerations.

2. **Customer Acquisition Lag**. There is a significant lag between the first meeting with the customer and the time at which the contract is signed, and in particular, the time at which payment is received. This can lead to cash flow issues.
3. **Product Scalability**. As discussed earlier, due to the services (human) element of the offering, it is difficult for a company like Optimality to scale; i.e. the challenges of point 1 and 2 are exacerbated with each new, potential, customer acquisition.

Thus, to remove these impediments to achieving business goals, Optimality are in the process of moving to a model whereby customers download and use their products without the need for assistance; in effect replacing the services component of the business model with a minimalistic support service (incl. documentation).

ISO/IEC 12207 Perspective

While perhaps not central to the core software engineering tasks associated with product development, other processes that surround the general delivery of software require adaptation in order to enable the business to evolve and overcome the challenges identified above. Specifically, it is the ISO/IEC 12207:2008 (ISO/IEC, 2008) system context processes that are currently under review in Optimality at this time: Supply, Installation and System Qualification Testing process. It is interesting that the case study organisation, while unaware of the existence of ISO/IEC 12207:2008, could readily relate to the system context processes at this stage in their evolution. This demonstrates the potential universal appeal of ISO/IEC 12207, while at the same time emphasizing the need to take a more complete end-to-end delivery focus as the organization strives to transition from a small consultancy-led operation into a product-focused company.

Perhaps it is the case that start up organisations must necessarily focus initially on the software centric processes as they go about the business of innovation and customer/idea identification. However, when the time for expansion arises and assuming that a product-focused strategy is preferred, the process focus may need to shift to the system context (having first consolidated a functioning set of software centric processes in the initial start-up phase). A good example here is that a Software Installation procedure may need to be improved in order that it will work in the mass marketplace, plus there may be implications for formal end-user licensing arrangements that were not of major concern in earlier phases of company establishment. A further example which is in evidence in the case study organisation relates to the strategic decision to offer a free trial to allow users to see the benefits without having to pay for the product. The free trial will contain a minimal feature set and is limited to a short time period. If a user is satisfied that the trial has demonstrated that product has utility for them, they may purchase the full version of the product.

Software Process Implications of the Scalable Model: In contrast to the previous model whereby Optimality employees are available to deal with any issues that arise during the adoption process, the downloadable offering has to work out of the box. This meant that Optimality needed to reduce the feature set initially to ensure that they could focus on a robust product that anyone familiar with database access mechanisms (such as JDBC) can use. This eliminated the requirement of hiring Optimality consultants to facilitate the adoption process.

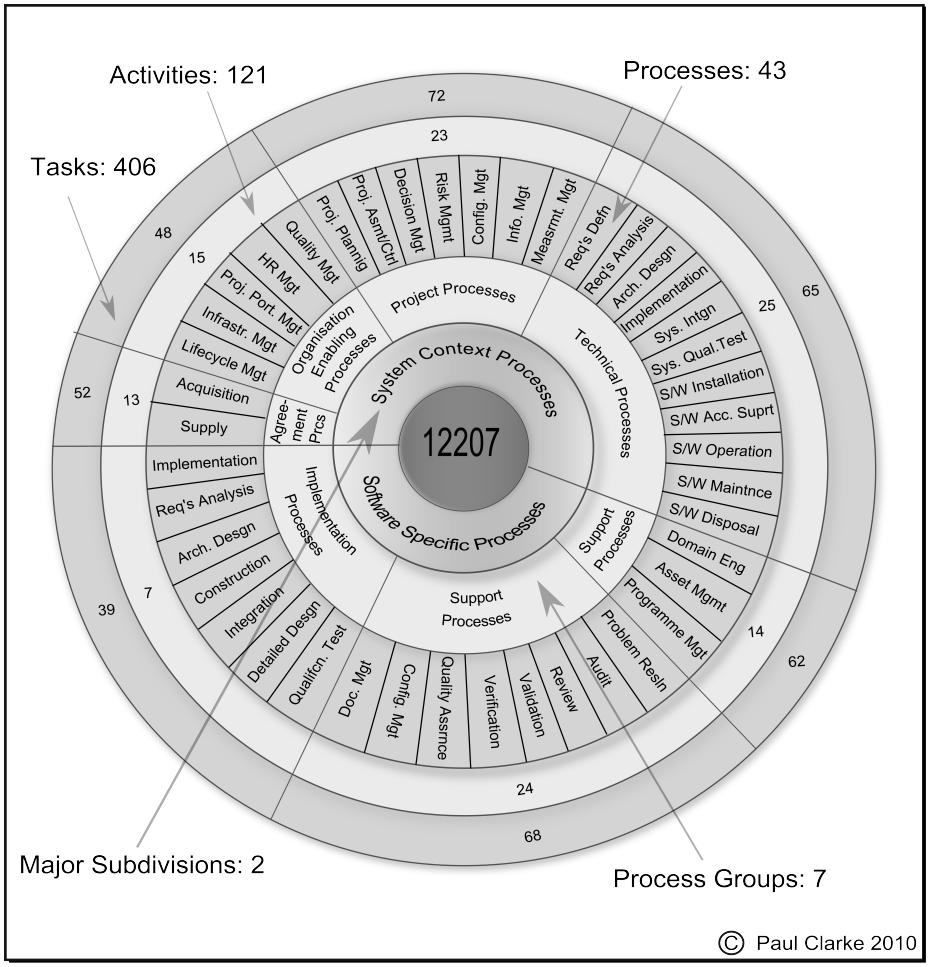

Fig 4. ISO/IEC12207:2008 Software Processes

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 (for example, from a user interface), 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.

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.

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.

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 advanced form of automation, it would not be possible to sustain the pace of development while also satisfying the quality constraints.

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.

Discussion

In Optimality, we have observed what we believe to be a common theme in software development process decision making: a complex set interrelated situational factors need to be addressed in the software development process, thus any individual software development process decision might deliver benefits for various situational constraints (ref. to Table 4). Such adaptive mechanisms can be considered favorable in the context of complex adaptive systems, wherein interrelated concerns continually interact. As we have advocated 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 would appear to offer support for this observation. However, Optimality did not conduct their process adaptation through application of the situational factors framework utilised in this retrospective study, rather they modified whatever aspect of the process they felt justified change at any point in time (and only to the extent that it was economically feasible to do so). There is a suggestion that perhaps each SME adopts a different process, and perhaps it is routinely more finely tuned than is widely appreciated (especially given the proliferation of various tool sets that now exist, each with different functionality). And this fine tuning is not just an SME concern; in a related study into the role of situational context in SMEs and medium sized organisations, we have witnessed some considerable variability in reported process enactment (P. Clarke et al., 2017).

Evidence in support of the role of organizational learning as a catalyst enabling process adaption was observed in the Optimality case study. While the company pursued an aggressive product innovation strategy with a generally lean development approach to feature delivery, it was discovered that in practice, the cost of refactoring (which was an absolute necessity given the product quality and cost-base constraints) had a strong tendency to grow, sometimes quite quickly. As a result, the company had to adapt their process in order to implement better product architecture and design early in development iterations 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. Or perhaps it is the case that in order to find a cost-effective process formula (especially in a product based environment), agile and lean based processes must ultimately focus increased attention on refactoring as a fundamental concern. To some extent, this is an intuitively appealing concept: as to retain unneeded and poorly designed software components will only inhibit product expansion, and it also holds the potential to introduce additional costs in various respects including software testing and maintenance.

Given that the variation in iteration durations is quite high and the basic operational demand for relatively high quality levels, it may be the case that the Optimality process, while being agile, also shares some common ground with Boehm's spiral model (Boehm, 1988). We also see evidence in Optimality of increased automation in software development, a phenomenon which we have witnessed in other case studies (P. Clarke, Elger, & O'Connor, 2016; O'Connor, Elger, & Clarke, 2017). And while Optimality may consider their process to be agile or lean, the significant variation in iteration durations (ranging from 5 days to more than 130 days) and the 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" (Fowler & Highsmith, 2001). 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.

Looking to the future, we see that Optimality are now starting to focus their energies on some of the ISO/IEC 12207:2008 system centric process. For example, they have decided to adapt their Supply process in order that they can deliver an out-of-the-box product. Furthermore, this decision has implications for a host of other system context processes, including system qualification testing and software installation. It would appear that Optimality has a vibrant approach to process evolution – the process can be seen to be changing all of the time, and this adaptation is considered to be informed by situational context and vital for business performance. It can also be seen from our case study that ISO/IEC 12207 holds relevance for small companies, even if these organisations might not be familiar with the standard at the current time. Furthermore, were ISO/IEC 12207 to meet with more widespread adoption in smaller software companies, current issues in relation to software development terminological confusion (P. Clarke, Mesquida Calafat, Ekert, Ekstrom et al., 2016a; P. Clarke, Mesquida Calafat, Ekert, Ekstrom et al., 2016b; Sauberer et al., 2017) might be reduced which could represent a significant positive improvement for the broader community. In earlier related work, the authors have introduced gaming mechanisms as a means to teaching ISO/IEC 12207 in an educational setting (Aydan, Yilmaz, Clarke, & O'Connor, 2017), which we hope can also assist in reducing the terminology problem. Perhaps it is the case that if ISO/IEC 12207 was to be published in a reduced or consolidated format, then many smaller companies might be able to embrace it as a useful process reference. In its current format which runs to a sizeable volume, it may be the case that smaller companies might feel that they have insufficient resources to start using ISO/IEC 12207 in their business.

Perhaps the most fundamental learnings from our case studies to date are that it would appear that small companies engage in process fine tuning on a regular basis, that process change is heavily influenced by their perceived situational context, and that certain aspects of process must necessarily be fluid in these settings. A good example of this can be seen in the lack of certainty surrounding iteration duration and iteration interruption – both this case study and an earlier study in another small company demonstrate that iteration durations may vary and that sometimes, iterations may be interrupted. This observation would appear to run contrary to the constant pace of development once advocated in earlier agile software development approaches.

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

Aydan, U., Yilmaz, M., Clarke, P., & O'Connor, R. V. (2017). Teaching ISO/IEC 12207 software lifecycle processes: A serious game approach. *Computer Standards & Interfaces, 54*(3), 129-138.

Boehm, B. (1988). A spiral model of software development and enhancement. *IEEE Computer, 21*(5), 61-72. doi:10.1109/2.59

Clarke, P., O'Connor, R. V., Leavy, B. (2016). 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),* Austin, TX.

Clarke, P., Mesquida Calafat, A. L., Ekert, D., Ekstrom, J., Gornostaja, T., Jovanovic, M., . . . Yilmaz, M. (2016a). An investigation of software development process terminology. *Proceedings of the 16th International SPICE Conference,* Dublin, Ireland. *, CCIS 609* 351-361.

Clarke, P., Mesquida Calafat, A. L., Ekert, D., Ekstrom, J. J., Gornostaja, T., Jovanovic, M., . . . Yilmaz, M. (2016b). Refactoring software development process terminology through the use of ontology. *Proceedings of the 23rd European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2016),* 47-57.

Clarke, P., & O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Journal of Information and Software Technology, 54*(5), 433-447.

Clarke, P., & O'Connor, R. V. (2015). 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),* Ankara, Turkey. 100-111.

Clarke, P., O'Connor, R. V., Solan, D., Elger, P., Yilmaz, M., Ennis, A., . . . Treanor, R. (2017). Exploring software process variation arising from differences in situational context. *Proceedings of the 24th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2017),* Ostrava, Czech Republic. 29-42.

Clarke, P., O'Connor, R. V., & Yilmaz, M. (2012). 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),* 62-74.

Clarke, P., Elger, P., & O'Connor, R. V. (2016). Technology enabled continuous software development. Paper presented at the *Proceedings of the International Workshop on Continuous Software Evolution and Delivery,* Austin, Texas. 48-48. doi:10.1145/2896941.2896943

Clarke, P., O'Connor, R., Leavy, B., & Yilmaz, M. (2015). Exploring the relationship between software process adaptive capability and organisational performance. *IEEE Transactions on Software Engineering, 41*(12), 1169-1183. doi:10.1109/TSE.2015.2467388

Dyba, T., Sjoberg, D. I. K., & Cruzes, D. S. (2012). What works for whom, where, when, and why?: On the role of context in empirical software engineering. Paper presented at the *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement,* Lund, Sweden. 19-28. doi:10.1145/2372251.2372256

Dyba, T. (2013). Contextualizing empirical evidence. *IEEE Software, 30*(1), 81-83. doi:10.1109/MS.2013.4

Feiler, P., & Humphrey, W. (1992). *Software process development and enactment: Concepts and definitions*. CMU/SEI-92-TR-004. Pittsburgh, Pennsylvania, USA: Software Engineering Institute, Carnegie Mellon University.

Fowler, M., & Highsmith, J. (2001, August). The agile manifesto. *Software Development,* , 28-32. Retrieved from http://hristov.com/andrey/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf; http://www.ddj.com/architect/184414755

ISO/IEC. (2008). *ISO/IEC 12207-2008 - systems and software engineering – software life cycle processes*. Geneva, Switzerland: ISO.

McLeod, L., & MacDonell, S. (2011). Factors that affect software systems development project outcomes: A survey of research. *ACM Comput.Surv., 43*(4), 24:1-24:56. doi:http://doi.acm.org/10.1145/1978802.1978803

Nevalainen, R., Clarke, P., McCaffery, F., O'Connor, R. V., & Varkoi, T. (2016). Situational factors in safety critical software development. Paper presented at the *Proceedings of the 23rd European*

Conference on Systems, Software and Services Process Improvement, EuroSPI 2016, Graz, Austria, September 14-16, 2016, 132-147. doi:10.1007/978-3-319-44817-6_11"

O'Connor, R. V., & Clarke, P. (2015). Software process reflexivity and business performance: Initial results from an empirical study. Paper presented at the *Proceedings of the 2015 International Conference on Software and System Process,* Tallinn, Estonia. 142-146. doi:10.1145/2785592.2785607

O'Connor, R. V., Elger, P., & Clarke, P. (2017). Continuous software engineering - A microservices architecture perspective. *Journal of Software: Evolution and Process, 29*(11), 1-12.

O'Connor, R. V., Elger, P., & Clarke, P. (2016). Exploring the impact of situational context: A case study of a software development process for a microservices architecture. Paper presented at the *Proceedings of the International Conference on Software and Systems Process (ICSSP '16),* Austin, Texas. 6-10. doi:10.1145/2904354.2904368

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research, 2*(1), 1-28. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=4431364&site=ehost-live

Sauberer, G., Villar, B. N., Dreßler, J. R., Schmitz, K., Clarke, P. M., & O'Connor, R. V. (2017). Do we speak the same language? terminology strategies for (software) engineering environments based on the elcat model - innovative terminology e-learning for the automotive industry. *Systems, Software and Services Process Improvement. EuroSPI 2017. Communications in Computer and Information Science, Vol 748,* 653-666.

SEI. (2010). *CMMI for development, version 1.3*. CMU/SEI-2006-TR-008. Pittsburgh, PA, USA: Software Engineering Institute.

Biographies

Gerard Marks is a database performance consultant who specializes in query processing and indexing. After graduating with a PhD from Dublin City University (DCU) in 2011, Gerard successfully commercialized the technology that he was working on in DCU and founded the Database Performance & Migration Group (DPMG). DPMG worked closely with a number of industrial partners to develop a tool set that could retrospectively optimize existing software applications (with zero code rewrites). In 2015, Gerard established Optimality Technologies; a company that specializes in enhancing software applications with a unique brand of database access products and services. He can be reached by email at Gerard.Marks@optimalitytech.com.

Rory V. O'Connor is a professor of software engineering at Dublin City University (Ireland), where he is currently serving as the Head of the School of Computing. He is also a senior researcher with Lero, the Irish Software Research Centre, and is Ireland's Head of Delegation to the ISO/IEC JCT1/SC7 standardization body. His research interests are centered on the processes and standards whereby software-intensive systems are designed, implemented, and managed. He is currently the editor in chief of the journal Computer Standards and Interfaces. He can be reached by email at Rory.OConnor@dcu.ie.

Murat Yilmaz is a lecturer and a researcher at Çankaya University (Turkey). He holds a Masters' from the University of Minnesota and a PhD from Dublin City University. Dr. Yilmaz has worked for 12 years as a software developer, software architect, technical lead, systems engineer, and project coordinator. His research interests include empirical/experimental software engineering, method engineering, game theory, and mechanism design, serious gaming, software team dynamics, agile project management, and gamification. He can be reached by email at myilmaz@cankaya.edu.tr.

Paul M. Clarke is an assistant professor at Dublin City University (Ireland) and has active research engagements in the areas of software process, process adaptive capability, situational factors affecting software development processes, complexity theory in software engineering and continuous software engineering. He is a researcher with Lero, the Irish Software Research Centre, and is a nominated national delegate to the ISO/IEC JCT1/SC7 standardization body. He can be reached by email at Paul.M.Clarke@dcu.ie.