Jordan Healy	13379226
Hazel Egan	12439798
Rob McGrane	12428288
Cianan Simpson	13433252
Group	AF

We the undersigned declare that the project material, which we now submit, is our own work. Any assistance received by way of borrowing from the work of others has been cited and acknowledged within the work. We make this declaration in the knowledge that a breach of the rules pertaining to project submission may carry serious consequences.

Signed: _____

Comparing the Software Development Process in Two Different Situational Contexts

Jordan Healy[1], Hazel Egan[1] Robert McGrane[1], and Cianan Simpson[1]

Dublin City University, Dublin, Ireland,
jordan.healy36@mail.dcu.ie, hazel.egan3@mail.dcu.ie,
robert.mcgrane2@mail.dcu.ie, cianan.simpson5@mail.dcu.ie

Abstract. This paper looks at two software development companies, FINEOS and NearForm. These organisations are comparable in a number of ways. It examines their chosen software development process along with their situational context. The goal is to examine, contrast and compare both the their chosen process in conjunction with their corresponding situational context. This is in order to create a discussion regarding the complexity and importance of selecting a software development process.

Keywords: Software Development Process · Situational Context · Agile · Waterfall · Continuous Software Development

1 Introduction

Software development methodologies have seen a series of adaptations over the years, continuously focusing strongly on specific aspects of the development process. Each different approach claims to have a unique perspective or to improve from its predecessor. Due to the existence of so many software development approaches we can assume that there is a relatively high level of complexity that exists within software development. This assumption cannot be too farfetched if we consider the act of creating a product from a mere idea [1]. Today, there are a large number of software processes that have been introduced and studied at a great lengths, and after much debate we still have no "gold standard". Since no one process is perfect for one workforce, we assume that there must be, to some extent some amount of adaptation and tailoring. A software development process consists of a set of methods, practices, activities, and ultimately transformations that are used to develop and maintain software and its corresponding products [2]. A more appropriate definition may be; the sequence of steps required to develop or maintain software [1].

The determination of a relevant and suited development process is dependant on the context of the work environment. There is a catalogue of factors that must be taken into consideration, including; nature of the application under development, team size, requirements redesign, product size, project duration, project complexity and personnel experience [3][4].

A situational context is unavoidable to continuous change and should not be seen as one-off event [1]. As one's situational context changes so will its

chosen process. This continuous change in context is causing a rapid growth in development processes. As the large consumer base demands the most new and innovative technology, it forces the development teams to strive towards a more efficient process. It is vital that the project's context be clearly defined before a process is chosen.

The remainder of this paper will focus on two case studies of software development companies with detailed description of their process and corresponding situational contexts. Following this will be a comparison and contrasting discussion of each case study, and finally a conclusion from this.

2 FINEOS

FINEOS Corporation is a market leading provider of core systems for Life, Accident and Health insurance. The company was established in 1993, delivers innovative solutions to a global market and now has customers, employees and established bases in North America, Europe and the Asia Pacific markets. They have over 500 employees [5]. In 2014 FINEOS invested 9 million in a research and development investment plan and now have over 200 working on research and development globally [5][6]. They are a profitable private company which is continuing to grow and is financially strong [5]. FINEOS are rated number 3 in the Financial Services Industry in The Irish Times ranking of top 1000 Irish Business [7].

FINEOS focus on the triple constraint issue of trying to juggle cost, scope and time in the right measures. Working in the insurance industry they must ensure their software is accurate and reliable. Similarly, security and privacy are incredibly important with their software as customers personal information will be stored on the systems [5]. FINEOS rely on customer advocacy, meaning they focus on what is best for the customer when working on a project [5][8]. They are constantly striving to delivery projects as quickly as possible but as their projects are generally largely code based they must ensure everything runs to a high standard before handing it over to the customer. FINEOS produce modern and innovative applications which are flexible and configurable. They strive for suite coherence, meaning all apps look and feel the same to the user. FINEOS have a large scale international customer base globally with over 25,000 claims users in 9 countries. Each year, on average, they manage up to 6 million new claim case and $7 billion in claims payments. Despite the large number of claims they have a 100% delivery success track record with high customer satisfaction [5]. FINEOS were ranked as the top 55th best technology company by customers in The Irish Times Top 1000 Irish Businesses ranking [7].

The company has come a long way in terms of their approach to development since the late 90s when the private company was first set up. The company started with a Waterfall approach with not much planning to an Iterative (RUP) approach then to Scrum and finally they changed to the current scaled agile approach in 2014 [5]. Scrum became very popular among software development teams and FINEOS still practice scrum as part of their development process [9].

Agile Software Development emerged as an umbrella term in 2001 for a set of lightweight methods that were considered an alternative approach to the overly bureaucratic and ineffective sequential, plan-driven, waterfall process that were dominant at the time [10]. FINEOS currently use the Scaled Agile Framework (SAFe) for their application development, and employ a range of different agile practices [5].

FINEOS work mainly with Java Enterprise Edition and for their mobile devices they code in HTML5. FINEOS use a method of continuous integration for their development, they run a full suite of code checks, build, deploy and test automation several times a day. This matches the pace of their product delivery as they are constantly producing new implementations for customers. This shows that this approach promotes efficiency and keeps up their standard of being fast to market. They maintain quality using triple constraints, large amount of work is done every day but the quality doesnt suffer.

FINEOS cut their release cycle from months to weeks by switching to an agile approach [5]. This shows that their current software development process is working well with their situational context in terms of their hopes to be fast to the market with their latest software developments.

Agile adoption also suits the company well because better mechanisms for change management is important for technology companies as it is a fast evolving area.

SAFe was established by Scaled Agile Inc. The first release was published in 2011 to the current SAFe website and the latest release (Version 4.0) was released in January 2016. SAFe was developed in the field with the idea that is flexible enough for each company to apply only what works the best for their situational context. SAFe is based on a number of underlying Lean and Agile principles [11]. The nine principles are;

1. **Take an economic view:** If the completed project doesnt suit the customers or the developing companies economic goals then the long-term viability of the system is in question [11].
2. **Apply systems thinking:** There are three primary aspects of systems thinking
   (a) **The Solution is a System:** Understanding what the project is and how it interacts with the environment and systems around it is fundamental.
   (b) **The Company Building the System is a System Itself:** Companies must understand also that the environment in which the builders of the system work is a system itself and must be optimised for those workers.
   (c) **Optimising the Full Value Stream:** Leaders of the company are continuously optimising the full value stream, especially as it goes from the technical side to the organisational side of the project [11].
3. **Assume variability:** preserve options: SAFe principle 3 suggests using a Set-Based Design approach. This process leaves design options open as long as possible, instead of designers deciding on one solution in the beginning of development, only to realise too late that a different approach may have been more suitable [11].

4. **Build incrementally with fast, integrated learning cycles:** After following principle 3, SAFe principle 4 suggests the use of a Plan-Do-Check-Adjust cycle which encourages constant checking both technically and with the customer, that the solution is heading in the right direction. Frequent checkpoints throughout production give a great insight into whether adjustments to scope, technical approach, cost or delivery timing is required [11].
5. **Base milestones on objective evaluation of working systems:** Principle 5 is based around the fact that Phase-Gate Milestones generally do not work in the long run. Instead SAFe suggests basing milestones on objective evidence viewed at each integration point (as laid out in the previous principle) [11].
6. **Visualize and limit WIP, reduce batch sizes, and manage queue lengths:**
   (a) **Visualize and Limit WIP:** Balancing the work in progress (WIP) against the available development capacity will speed up the process in the long run by not overloading workers with too much work.
   (b) **Reduce Batch Size:** Small batches move through the system faster, with less variability, which encourages faster learning.
   (c) **Manage Queue Lengths:** Reducing queue lengths decreases delays, reduces waste and increases predictability of outcomes [11].
7. **Apply cadence (timing), synchronize with cross-domain planning:**
   (a) **Cadence:** Cadence transforms unpredictable events into predictable ones which makes waiting times predictable, facilitates planning and lowers the transaction costs of key events.
   (b) **Synchronisation:** Synchronisation pulls the disparate assets of a system together to assess solution-level viability, align the development teams and business to a common mission and finally, integrate the customers into the development process.
   (c) **Synchronisation with cross-domain planning:** The Release (PI) Planning event in SAFe gives true knowledge of the current state and has three primary purposes.
      i. Assessment of the current state of the solution.
      ii. Realign all stakeholders to a common technical and business vision.
      iii. Plan and commit to the next program increment [11].
8. **Unlock the intrinsic motivation of knowledge workers:** Understand and acknowledge that the knowledge workers generally know more about the project they have worked closely on than the higher ups of the company who are more business focused [11].
9. **Decentralize decision-making:** Escalating all decisions to higher levels of authority leads to delays in production. Allow for technical workers to make minor decisions themselves so as to speed the process and allow for the best decision to be made promptly [11].

It is unclear as to whether FINEOS use all of the principles listed above although many of these principles would be suitable for their situational context. As previously mentioned, since implementing the SAFe process in the company

they have dramatically increased the speed of production of their projects. This could be due to many of the principles above such as principle 3, which allows for variability or principle 9 which allows workers to make minor decisions themselves, meaning they are not delayed waiting on a response from their superior. Principle 4 introduces regular checkpoints throughout production, which corresponds to the regular tests FINEOS perform daily, giving them a good insight into where they are with the project at all times. Principle 1 reflects how much money might be relying on the success of the software FINEOS produces, as they are working in the insurance industry and is likely the first step FINEOS take in their development process. Due to the clear success of the implementation of SAFe into the FINEOS company it is clear that it is a suitable software process for their current situational context.

3 NearForm

NearForm Ltd. are a software organization who have a great presence with both US and European customers. As the first Node.js consultancy agency established nearly 6 years ago, NearForm strive to deliver high quality software in regular deployment intervals [12]. There is a growing trend in the increase in release cycles and iteration times [13]. Shorter release cycles gives the developers user feedback earlier which will lead to happier customers [13], and NearForm aspire towards this. The choice of Node.js enables rapid code development by using one platform throughout their entire system [14]. NearForm are an expanding company, with a growth of to 135 employees in under four years [15].

NearForms practice is to combine large computing power along side established frameworks & tools and an accelerated development process & release cycles [12]. This is to follow the footsteps from already successful companies such as; Airbnb Inc., Netflix and Uber [12]. This is not to say that many successful companies do not have inhibitors within their development process. Such inhibitors may be a cause of internal factors (which could be eventually be rectified) some of which are; language or platform bias or even political context [16].

NearForms emphasis on micro services stems from its characteristics that every service is responsible for only one single part of the functionality of the entire software (Single responsibility principle). Micro services can be seen as distributed systems which allows the entire system to be broken down into a series of discrete cooperating systems [14]. And its advantage is that each service will have around 100 lines of code. This technology reduces risk and assists in their continuous delivery [12]. Combining micro services with NearForms other technology, Docker [1], individual services can be deployed without affecting the whole system [14]. Along with these technologies, tooling can be as important. NearForm use GitHub as a version control system to quality commit and review

[1] Docker is a container platform to run and manage apps side-by-side in isolated containers to get better compute density

code [14]. They also use Travis CI, a continuous integration platform to assure that the system passes build test [14].

It is known that there are a large number of different software development processes and that it is assumed that each organisation tailors processes to suit their situational context [17]. NearForm claim that they have no process, per se but do admit that several different processes contribute to their methodology [12]. Smaller teams can be more lenient with their adopted process and now that NearForms team is growing, it is more integral than ever that they define their process. NearForm have adopted two popular methodologies in order to address the potential disadvantages of each process. The Waterfall process is broken up into seven stages, with each stage's output is fed into its following stage. The emphasis here is on validation at early stages in the process chain [18]. The disadvantage of Waterfall is that if there is a mismatch between the clients requirements and the designed requirements of the system, the client will only see this at a very late stage [18]. NearForm aim to counteract this by implementing reparations of integrations through aspects of continuous software development. Continuous software engineering (CSE) focuses on three continuous aspects in order to increase the efficiency of the software engineering process [19].

1. **Continuous integration:** to test features as soon as theyre implemented.
2. **Continuous delivery:** to delivery new versions of the system early.
3. **Continuous deployment:** to deploy the new version as quick as possible.

With the help of tooling, organisations can closely represent the newest version. This CSE process has emphasis on automation. The advantage of using CSE along side Waterfall is that it solves risk of issues not being found early, which is likely to happen when solely using Waterfall. The main disadvantage towards using CSE is that there is often a lack of focus on the project's overall design [20]. This drawback can be overcome in the initial design stages of Waterfall. With the combination of Waterfalls emphasis on validation and CSEs emphasis on automation, NearForm feel that they have created the best possible process for their situational context. This is proved when you look deeper into what is involved with continuous delivery. When looking at a continuous delivery method we can see that it is made up of 4 elements, Automation, Traceability, Independent Artifacts and Pipeline [13].

1. **Automation:** This is what allows continuous delivery to work. Building, testing and deployment are repeated with each change in code. This automation is put under version control and executed with every change. This ensures deployment is always working and corresponds properly to the newest software and infrastructure to reduce errors.
2. **Traceability:** When a change is made to a project that uses another library, when one is changed both should be rebuilt instead of just the one that was changed. This makes sure that changes in the library that change results for the main project are easily identified.
3. **Artifacts:** Artifacts created during the process should be independent of the target environment. There should only be one artifact created for the

production and testing environment. With this one artifact set up it is always ensured that the version on production is the correct one that was tested.

4. **Pipeline:** The pipeline models the process from check-in to release. This process consists of quality gates and stages. Quality gates could be passing unit tests or having valid binaries. If a change is made to the source and configuration repository, then unit tests are executed and if they pass they are uploaded to the package repository. After all tests are passed the artifact from the package repository can be deployed. The use of the package repository makes sure the same build is used throughout the process.

We can see from these stages that using a method of continuous checks and quality assurances that this is the best process for NearForm in their current situation. As they rely on constant releases and changes in their working environment continuous delivery seeks to automate the process of testing, library updates and version control to make sure that every change is properly catalogued which is extremely efficient in a company that are constantly releasing new builds of software for their customers.

4 Comparison of Situational Contexts

FINEOS and NearForm are two strongly differing companies. While FINEOS is a well established company, running for over 20 years since first being established in 1993, NearForm is a relatively new company only having been established 6 years ago [5][12]. Likely partially due to this gap in establishment, FINEOS has a much larger employee base of over 500 people, with both employees and customers in North America, Europe and the Asia Pacific markets [5]. In contrast, NearForm has only 135 employees with customers in the US and European countries [12].

NearForm uses Node.js so as to enable rapid code development by using one platform throughout their entire system [12].FINEOS works with the Java Enterprise Edition as they claim it is a rock solid enterprise platform. FINEOS also use HTML 5 when developing software for mobile devices [5].

NearForm is a consultancy agency for Node.js, some of their success can be attributed to the fact that they were the first Node.js consultancy agency and are still the largest as well. Another contributing factor to their success is their contributions to the open source community, through their funding and contribution for open source projects they strive for innovation and are part of a large community of programs producing many different projects [21].

FINEOS is a market leading provider of core systems for Life, Accident and Health insurance. Much of their success can be attributed to their attitude towards their products. Their digital platform strives to have the claims, policy and billing aspects of insurance all done within one suite. They also strive for coherence in their sweets so all their programs function and look the same.

If we contrast FINEOS and Nearform we can see that their two companies function in different ways. NearForm which is a smaller and newer company has many new ways of thinking involved in how they treat business. In NearForm there is a focus on employees individual contributions to both the company and

open source. FINEOS the older and larger company has a more commonplace business model with a focus on product and customer support.

Currently the two companies also have different risks involved in the type of work they do, with FINEOS being one of the leading insurance software solutions in the world a mistake on their end could end up costing companies or people cumulative millions in many different countries. While if we look at NearForm as a consultancy agency a mistake on their end would usually end up with a much lower risk of catastrophe as it would only affect the client they are currently consulting.

5 Comparison of Software Development Processes

As NearForm Ltd. are a relatively younger and smaller company (6 years old with 100 employees) therefore it can be easier for them to decide upon and adapt a software process [12]. While FINEOS are a considerably older and larger organisation (24 years old with over 500 employees), choosing a software process is more challenging. These challenges can arise from the inhibitors mentioned in a previous section. These inhibitors have an exponential growth due to the larger number of employees, particularly within the factor of political context [22]. With NearForm being an older company they have had more time to grow through different processes to find one (or many) that suit their situational context.

Neither company uses the same software process, FINEOS uses SAFe while NearForm uses a combination of Waterfall and CSE, although FINEOS did use Waterfall in their inception [5]. FINEOS use SAFe for their software development, and employ a range of different agile practices. Moving from Waterfall allows them to cut their release cycle to weeks rather than months [23][24].

Both NearForm and FINEOS have two different focuses on their products. NearForm focuses to deliver high quality software in regular deployment intervals [12]. While FINEOS do this also, they have a higher importance on ensuring that their software is accurate and reliable in both security and privacy [5]. This is due to the industry in which they develop software in.

Both FINEOS and NearForm attribute a large amount of their success towards the use of frameworks & tools. Both organisations admit to using tools such as Git and TravisCI for version control and continuous integration [5][14]. These tools allow these teams to focus on continuous integration, delivery and deployment which would be almost unachievable without them. Although this continuous ideology is not a contributing attribute of SAFe, FINEOS still put heavy emphasis on it [5]. This continuous ideology can also be seen through NearForms choice of CSE. Both of these organisations have an attention to test features early, deliver new versions quickly and to deploy as soon as possible [19].

Both companies also use methods of continuous improvement. This is beneficial to companies who use cloud computing (especially for frequent releases), continuous delivery and integration which are all ways in which both FINEOS and NearForm would like to be releasing their software [5].

FINEOS and NearForm both believe that the use of micro services gives them the best possible method for this continuous ideology. Although, FINEOS also use these in conjunction with robust languages like Java [5].

FINEOS and NearForm are similar in terms of the fact that they both have a short release time for their software, although FINEOS have a more complex plans using SAFe compared to NearForm who are still using a Waterfall approach in some respects. This growth comes down to the fact that FINEOS are a more mature company and have grown greatly since using a Waterfall approach. Unlike NearForm, FINEOS are aggressively implementing SAFe aspects continuously [5][12].

Since FINEOS adopted an agile approach to their software development they have now got a faster return on investment and they can deal with changes that have to be made using better mechanisms.

Similar to NearForm, FINEOS employ continuous integration in their development processes. FINEOS perform continuous checks on quality such as code checks and both companies also run automated tests regularly.

FINEOS also make use of smart backlog prioritisation to help continuously improve their product. By using these practices FINEOS has cut their lead to sale time from 20-25 weeks to 4 weeks (depending on the size of the feature) [5].

FINEOS make use of inspection and adaptation techniques to also in order to fine tune their processes. FINEOS use a target operating model [2] which can be used to help with linking between information technology and strategy. FINEOS have now got a better custom and market fit through dynamic backlog prioritisation and feedback cycles, so they can create the right product for each customer [5].

From the comparison of these organisations development processes, we can see an increase in the trend of continuous methods through micro services [25]. These methods of integration, delivery and deployment can be garnered from an array of processes. It is evident that FINEOS use a process that do not specifically offer these ideas, but they do still strive towards it. Although, comparing only two businesses leaves much to be desired, it can be seen that both companies have followed the movement towards accepting an adaptation of one or many popular software development processes [1].

6 Conclusion

This paper has discussed how situational context and software development processes work interdependently. It is clear that there is never one development process perfectly suited to one situational context [1]. To be able to decide what development process is most suitable for a company, one must look closely at that company's situational context. Situational context, in this case, refers to the various factors in a company's environment that may have an affect on the development process [1]. When examining situational contexts, it is clear that

[2] Target operating model is a description of the desired state of the operating model of an organisation.

each situational context is different and that it is in fact possible that each possible context may be unique to that organisations, making it almost impossible to easily identify a perfectly matched development process to that context. In most cases, organisations will find the development process that is most closely suited to their situation and will use only the parts that work for them. They will often select different parts from different defined development process to create a process that works perfectly for them.

After examining the relationship between situational contexts and their suited development process, it is clear that their is a great complexity of the selection process. The complexity derives from many factors which all need to be satisfied in order to run an organisation to the highest possible standard. Finding this optimal development process can be quite the challenge. Often there may even be difficulties in implementing a new chosen process to an already established organisation. As there are so many factors involved in the running of an organisation it may be hard to get all of them to smoothly transfer to a new process of operations. Through this, it can be seen that even when the selection of a development process is complete, there is still more work to do in converting to the new process. There is also a strong need to continuously review if all parts of a chosen development process still work for that organization. As time goes on, companies continue to develop and change, with the possibility of many changes in situational factors such as the progression of technology, changes in company procedures or even company growth can lead to the need for the review of the process. What may once have worked or been relevant to a situational context, could change rapidly to being unnecessary and call the need for changes to the development process. Organisations must be aware of this ever growing need for continuous review if they wish to keep their process running as best as possible. The decisions leading to a chosen process are quite detailed and layered, maybe even more so than described in this paper.

The authors acknowledge that the importance between process and context is not being contested but hope to emphasize the complexity of this relationship. Many who have come face to face with this issue certainly understand the complexity involved and the authors believe this complexity may even be underrated by educators and professionals alike [1].

The goal of this paper was to demonstrate, with the aid of our two case studies, the importance of choosing a software development process in conjunction with a situational context. The authors of this paper aim to encourage further discussion of this topic with perhaps even further insight into the relationship between situational context and its development process.

References

1. Clarke P, O'Connor R, Leavy B (2016) A Complexity Theory viewpoint on the Software Development Process and Situational Context. 2016 IEEE/ACM International Conference on Software and System Processes (ICSSP) 1:86-89. doi: 10.1109/ICSSP.2016.019

2. Qasaimeh M, Mehrfard H, Hamou-Lhadj A (2008) Comparing Agile Software Processes Based on the Software Development Project Requirements. 2008 International Conference on Computational Intelligence for Modelling Control & Automation 1:49. doi: 10.1109/cimca.2008.54
3. O'Connor R, Elger P, Clarke P (2016) Exploring the Impact of Situational Context A Case Study of a Software Development Process for a Microservices Architecture. 2016 IEEE/ACM International Conference on Software and System Processes (ICSSP) 1:6-7. doi: 10.1109/ICSSP.2016.009
4. Ortega D, Silvestre L, Bastarrica M, Ochoa S (2012) A Tool for Modeling Software Development Contexts in Small Software Organizations. 2012 31st International Conference of the Chilean Computer Science Society 1:29-30. doi: 10.1109/sccc.2012.11
5. Solan, D. (2017) FINEOS - Process and Agile Journey. Dublin City University. 23 February 2017
6. Taylor C (2014) Tech firm Fineos to create 50 new jobs for Dublin. In: The Irish Times. http://www.irishtimes.com/business/tech-firm-fineos-to-create-50-new-jobs-for-dublin-1.1956026. Accessed 27 Mar 2017
7. Turnover (2017) Fineos on Top1000.ie. In: Top 1000. http://www.top1000.ie/fineos. Accessed 27 Mar 2017
8. Craig Bailey, Kurt Jensen(2006) http://www.customercentricity.biz/PDFs/Customer_Advocacy.pdf. Accessed 2 April 2017
9. Brenner R, Wunder S (2015) Scaled Agile Framework: Presentation and real world example. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 1. doi: 10.1109/icstw.2015.7107411
10. Power K (2016) Sensemaking and Complexity in Large-Scale Lean-Agile Transformation: A Case Study from Cisco. 2016 49th Hawaii International Conference on System Sciences (HICSS) 1. doi: 10.1109/hicss.2016.669
11. (2017) SAFe Lean-Agile Principles Scaled Agile Framework. In: Scaledagileframework.com. http://www.scaledagileframework.com/safe-lean-agile-principles/. Accessed 29 Mar 2017
12. Elger, P. (2017) Software, processes and origami. Dublin City University. 6 March 2017
13. Hansen P, Hacks S (2017) Continuous Delivery for Enterprise Architecture Maintenance. Full-scale Software Engineering/The Art of Software Testing 1:56
14. Clarke P, Elger P, O'Connor R (2016) Technology Enabled Continuous Software Development. 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED) 1:48. doi: 10.1109/CSED.2016.017
15. Kennedy J (2015) Waterford software start-up NearForm to create 100 new jobs. In: Silicon Republic. https://www.siliconrepublic.com/jobs/waterford-software-start-up-nearform-to-create-100-new-jobs. Accessed 29 Mar 2017
16. Salleh N, Al-Kautsar E, Hoda R, Asmawi A (2014) A Window into the Emergence of Agile Software Development Landscape in Indonesia. Journal of Advances in Software Computing and Its Application 6:11
17. Vijayasarathy L, Butler C (2016) Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?. IEEE Software 33:86-94. doi: 10.1109/ms.2015.26
18. Chandra V (2015) Comparison between Various Software Development Methodologies. International Journal of Computer Applications 131:7-10. doi: 10.5120/ijca2015907294

19. Ameller D, Farre C, Franch X, Valerio D, Cassarino A (2017) Towards continuous software release planning. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) 1:402-406. doi: 10.1109/saner.2017.7884642
20. Clarke P (2017) DevOps & CSE. Dublin City University. 16 February 2017
21. (2017) About Us. In: NearForm. http://www.nearform.com/about-us/. Accessed 1 Apr 2017
22. Arvidsson S (2010) Communication of Corporate Social Responsibility: A Study of the Views of Management Teams in Large Companies. Journal of Business Ethics 96:342. doi: 10.1007/s10551-010-0469-2
23. Petersen K, Wohlin C, Baca D (2009) The Waterfall Model in Large-Scale Development. Lecture Notes in Business Information Processing 32:397-398. doi: 10.1007/978-3-642-02152-7_29
24. Huo M, Verner J, Zhu L, Babar M (2004) Software quality and agile methods. Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004 COMPSAC 2004 1:1-2. doi: 10.1109/cmpsac.2004.1342889
25. Zhang H, Kitchenham B, Pfahl D (2008) Software Process Simulation Modeling: Facts, Trends and Directions. 2008 15th Asia-Pacific Software Engineering Conference 1:65. doi: 10.1109/apsec.2008.50