

Software Release Management Evolution - Comparative Analysis across Agile and DevOps Continuous Delivery

Samer I. Mohamed

Modern Science and Arts University, Faculty of Engineering, Electrical and communication department, Egypt

Abstract—Software release management is the process of managing, planning, scheduling and controlling a software build through different stages and environments; including testing and deploying software releases. Traditional approaches like ad-hoc and incremental/iterative approaches prove not to satisfy the current demanding clients or IT business. Thus a need for new techniques arise like agile software development, DevOps continuous delivery. DevOps and Agile complement each other to deploy working functionality into production faster. The main goal of Continuous Delivery and DevOps is to release more reliable applications faster and more frequently to satisfy the client and business needs. This paper sheds a light on the evolution of the software release management starting from traditional techniques towards agile and continuous delivery via DevOps. Analytical case study will prove how new software release managements techniques succeeded to bridge the gap of traditional techniques both in time to market and quality efficiency to fulfil the IT business needs.

Keywords—Continuous delivery, Operational excellence, Software Release management, Agile approach, DevOps.

I. INTRODUCTION

Software delivery evolves over the past years to fit for the objective of satisfying the end clients and IT industry needs. The ability of IT organizations and their products, systems, and services to compete, adapt, and survive within the current market depends increasingly on software delivery. Mobility, cloud computing and virtualization all put high pressure on IT organizations, and R&D to innovate new approaches/methodologies for software delivery to satisfy the high demand from customers [16]. Time to market, quality, reliability, productivity and customer satisfaction become critical for IT organizations to survive and able to compete within current IT market.

The fundamental agile principle of releasing frequently tends to get overlooked or ignored by organizations that approach agile transformations by scaling teams. It has

been overlooked by these organizations that new practices called DevOps and Continuous Delivery (CD) have begun to emerge to address this gap. In DevOps, the objective is to blur the lines between Development and Operations teams so that new capabilities flow easier from Development into Production. On a small scale, blurring the lines between Development and Operations at the team level improves the flow. In large organizations, this tends to require more structured approaches like CD [15]. Applying these concepts at scale is typically the source of the biggest breakthroughs in improving the efficiency and effectiveness of software development in large organizations, and it should be a key focus of any large-scale transformation.

Software release management process for future releases is considered a complex process since not all requirements can usually be met with available time and resource constraints in one software release. This process allows the product stakeholders to receive portions of their requirements in the product releases based on each release constraints. This type of software development called incremental software development [10]. There are many challenges for the release planning process which make it one of the most complex process in software requirements engineering [11], I will summarize some of these difficulties as follows.

- Requirements are not well specified and understood because there is usually no formal way to describe the requirements. Non-standard format of requirement specification often leads to incomplete descriptions and makes it harder for stakeholders to properly understand and evaluate the requirements.
- Uncertainty of data due to meaningful data for release planning are hard to gather and/or uncertain. Specifically, estimates of the available effort, dependencies of requirements, and definition of preferences from the perspective of involved stakeholders are difficult to gauge.
- Constraints exist while planning the releases needs to be taken into account by the product manager while allocating the requirements to

various releases. Most frequently, these constraints are related to resources, schedule, budget or effort and hard to determine as shown in figure 1.

- Unclear objectives from various stakeholders and the facility to define “Good” release plans are hard at the beginning. There are competing objectives such as cost and benefit, time and quality, and it is unclear which target level should be achieved [19].
- Release planning is typically done ad hoc, not based on sound data, models, experience and methodology. This is even the case when planning for several hundreds of features. As a consequence, the created plans do not create the maximum value achievable from the resulting products.

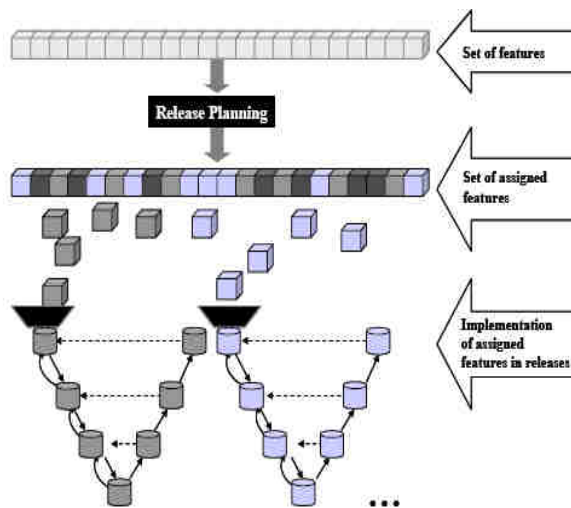


Fig.1: Planning and development process

Agile basically means an ability to harness change for competitive advantage. An agile software delivery has the ability to respond to and create change in a way that allows it to react to and gain advantage over its traditional counterparts of software development approaches. Agile businesses can implement concepts quickly (speed-to-market). They are able to quickly recognize, capture, and respond to new and emerging markets. Agile methodologies are the normal evolution of the traditional approaches of software development like water fall, incremental delivery, and/or iterative software delivery. Agile methods offer a viable solution when the software to be developed has fuzzy or changing requirements, being able to cope with changing requirements throughout the life cycle of a project [2]. Agile methods have proved to have a far higher agility and flexibility than the traditional software development [3] and are used to produce higher quality software in a shorter period of time [4]. Adoption of agile software development methods enables a

software developer to be more flexible and responsive to the changing environments and customer demands.

DevOps and Continuous Delivery (CD) is another subset of agile which the team keeps its software ready for release at all times during development. It is different from “traditional” agile in that it does not involve stopping and making a special effort to create a releasable build. CD is a group of practices and methodologies in software development that are designed to improve the process of software delivery aspects and ensure reliable software releases. Ultimately, it enables the systematic, repeatable, and more frequent release with high quality software to end clients[5].

The paper is organized as follows: section II gives a background for the evolution of the software release management starting from the traditional approaches towards new approaches of continuous delivery and DevOps; section III illustrates the proposed E2E framework and how IT services can be delivered in seamless strategy under proposed framework. Section IV introduces the proposed Proof of Concept (PoC) model; where PoC description, details, results, and recommendations are detailed; section V is the conclusion of this study.

II. EVOLUTION OF SOFTWARE RELEASE MANAGEMENT/DELIVERY BACKGROUND

There are many models exist in the literature for the software life cycle and release management which describe the series of steps the system goes through starting from realization of need, through construction, maintenance and retirement. Brief description for some of these models will be mentioned in the following sections.

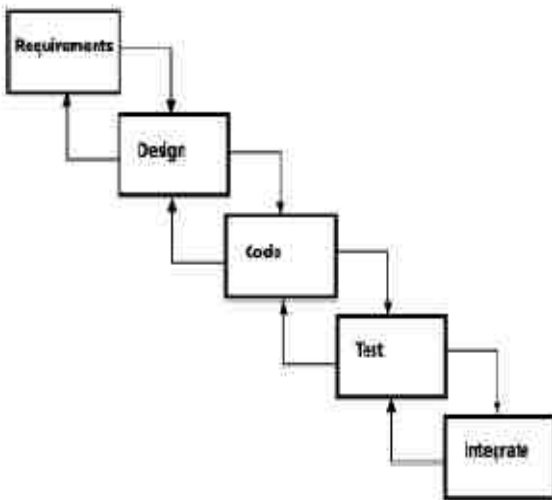
2.1. Ad-hoc methodology

This methodology focus only on planning the contents of the next direct release using manual approach. Ad hoc methods are used to determine solution plans but are far from objective demands. Many organizations have an ad hoc plan that relies solely on the judgment of the product manager [11]. An ad hoc approach may be suitable for relatively small in-house projects involving few tens of requirements and relaxed constraints.

2.2. Incremental methodology

Incremental software development is the process in which software product is developed in incremental manner such that additive components and/or faults correction are produced through the sequential product releases. This will enable the end customers to receive parts of the system early to get higher business value and gain early feedbacks. Release planning methodology for incremental software

development incorporates set of decisions about which software requirements to be implemented during which release. This will be a critical and challenging process especially with stakeholders conflicting perspectives, competing targets and different types of resources and financials constraints [5]. Thus the objective from the release planning process is to maximize the business value gained while balancing the stakeholder's objectives and meeting the resources, costs, schedule, and mitigate risks constraints.



2.3. Agile methodology

Agile approaches/methodologies guide software developer engineers to break down their software requirements down into small releases known as 'User stories' to accelerate the feedback and response from the client. This will facilitate aligning the software product features/requirements to fit for the business needs. This agile guiding principles centered to help small development teams to better deliver smarter and more efficient. Adopting this, software developers are able to produce their code in shorter iterations slots to satisfy the client and market needs. But the issue is raised when it comes to the interlocks with the other teams down the stream like operations, infrastructure teams due to difference in culture, working approach, scope of work, business processes, thus open the door for a need to another approach to resolve both the communication aspects between the interconnecting teams besides the process and execution aspect towards the end goal of satisfying the end client needs. DevOps and Continuous delivery approaches designed to fix this agile drawbacks from E2E perspective [9].

2.4. DevOps methodology

DevOps is a philosophy under which the business teams, software development teams, and the operations teams

collaborate on a continuous basis to make sure that IT solutions are available to business on time as per expectations and that they run without disruption. It calls for automation, collaboration, cultural change, process adaptation, and an organizational structure that is less complex and is easy to navigate. It addresses the people, process, and tools, as well as the technology dimensions needed to secure this collaboration and sync up the different stakeholders to move functionality to production faster. Both DevOps and agile in sync to release the value and benefits of the software products towards the business units. Besides it facilitate open channels and continuous communications between the development and operations team starting from the early stages of SDLC (Software Development Life Cycle) to understand the business vision and release planning aspects. The edge of DevOps is pushing towards full automation SDLC towards the clients especially for those apps require more than one release/day. Currently there are massive set of tooling towards this full automation SDLC [15].

2.5. Continuous delivery methodology

Continuous Delivery (CD) is built on the agile principles to resolve some of the agile drawbacks as detailed in the previous section like communication, processes, and tooling aspects. CD is composed of set of methodologies and practices within software delivery domain that are designed to improve the process of software delivery to ensure reliable software releases within shorter time [10]. It facilitates realizing the business value of software products to the customer in shorter time or by other means in continuous manner by making the software code deployable at any point of time through the development life cycle. Some of the added values of CD like:

- Accelerate time to market
- Ability to build the right product
- Improved productivity and efficiency
- Reliable releases
- Improved product quality.
- Improved customer satisfaction

III. PROPOSED PROOF OF CONCEPT DEVOPS CONTINUOUS DELIVERY FRAMEWORK

The previous sections show how the software release management approaches evolve over time to satisfy the end user demand and drawback of the Agile which only addresses the software requirements through software development and doesn't address rapid delivery of software to production systems. To address the rapid delivery to production and disconnect between development and operation teams via DevOps which addresses the collaboration, and automation between

software development and operation teams. The proposed framework is an approach to agile development, continuous integration, continuous testing, and continuous delivery through the use of automated tools, and streamlined processes. With the main objective to fill gap of current traditional approaches like waterfalls, incremental and evolutional approaches and even with agile methodology as described earlier in the previous sections. It helps with the new emerge trends in mobility, information optimization and converged clouds to easily deliver as per end-user rapid needs for social and mobile applications as shown in figure 2. The framework delivers incremental development continuously to production, which reduces defects, eliminates excess cycle time, provides continuous feedback and eliminates outage windows when deploying to production. Automation is a critical component of a successful DevOps continuous Delivery approach, and the tools in this space continue to rapidly advance.

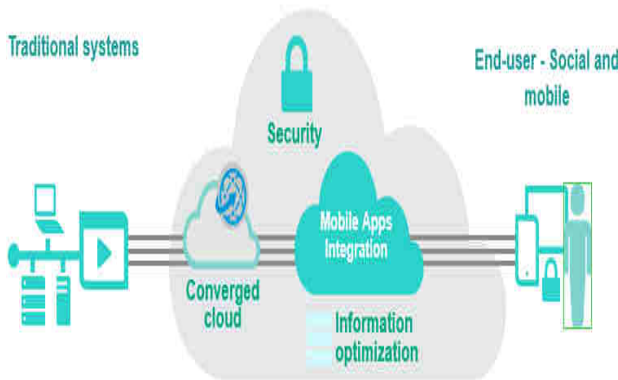


Fig.2: Proposed DevOps continuous delivery framework value

The proposed DevOps continuous delivery framework is agnostic to a particular toolset as shown in figure 3, and is customizable based on customer preference. The key steps for automation that enable the proposed DevOps continuous delivery framework include:

1. **Daily Code Commit.** Developers check-in code into a central source code repository on a daily basis.
2. **Automated Builds.** A Continuous Integration (CI) server is continually polling the source repository for changes, and when a change occurs the code is checked out of the repository and built. The built software is stored in a repository manager by the CI server.
3. **Automated Testing.** The code is automatically unit tested, code quality tested, smoke and UI tested, and performance tested.
4. **Automated Delivery.** The built version is deployed using provisioning tools that treat infrastructure as code.

The proposed DevOps continuous delivery framework introduced in [29] in more details as shown in figure 3. This detail the set of toolset used through the framework to facilitate End to End (E2E) continuous delivery starting from provisioning the infrastructure towards deployments of the code into production via continuous development, build, integration and testing.

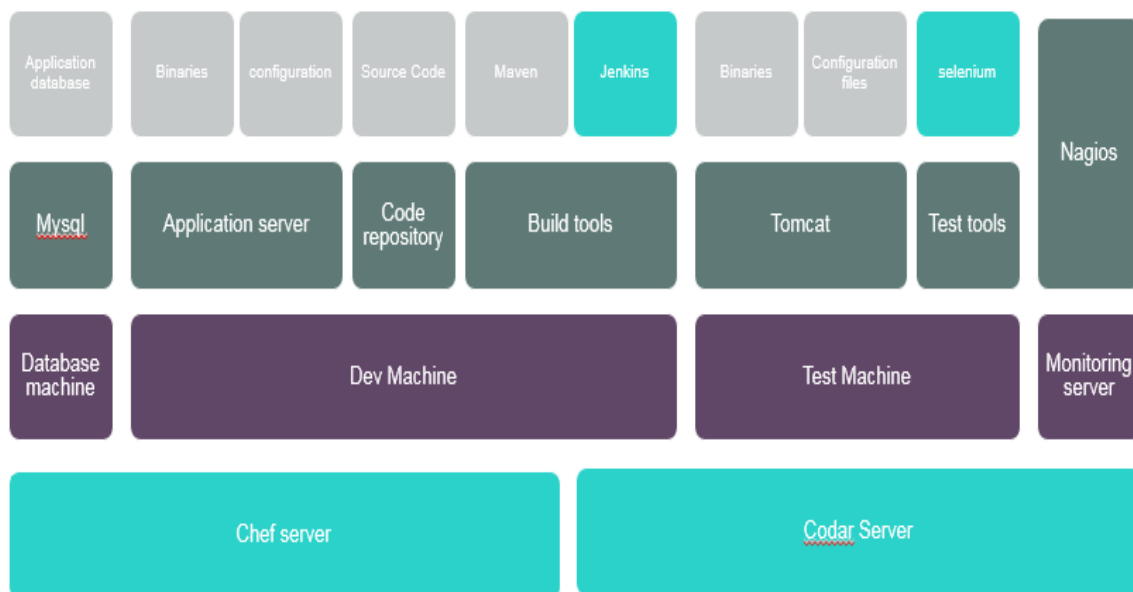


Fig.3: Proposed DevOps continuous delivery framework toolset

Continuous Integration (CI) and Continuous Delivery (CD) approach within the proposed framework as shown in figure 4 is designed to create an automation environment for the entire end-to-end release process so that every change to the application results in a releasable version that is built automatically. Software applications are built using this framework in the development process on every change checked in by the developers, thus make the code always deployable at any point of time. This

effectively eliminates the need for integration testing because the code is incrementally being integrated on a daily basis which removes the cost associated with developers spending time on this phase. The feature of continuous deployment, ability to have frequent incremental builds and mandating a comprehensive automated testing process allows developers to detect problems early and as a result, ensure higher quality.

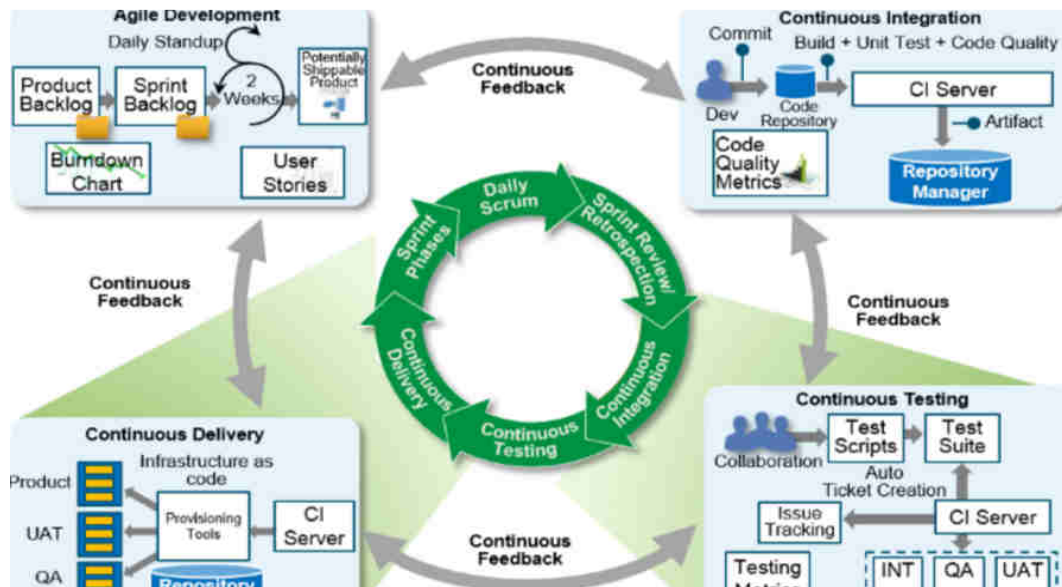


Fig.4: Proposed DevOps continuous delivery framework architecture

The main edge for this framework to support rapid deployment and release is the automation via set of tools as shown in figure 3 that allow the DevOps team to automate provisioning of infrastructure resources and platforms. The server configuration, packages installed, relationships with other servers are modeled with code, and is automated and has predictable outcomes, removing error-prone manual steps. The framework also introduce automated configured toolset that fits with the project/application scope needs and configure the required tools into the end-to-end application environment or infrastructure as infrastructure as a code (IaaC). It uses software development best practices for the infrastructure code and stores the code in a Code Repository with tags and branches, and releases the code just as if it were applications software. This infrastructure code is continuously integrated, tested and deployed right along the application software and is treated no differently.

The Continuous Integration (CI) server is configured with build steps to check for coding style, coding standards, and other features using tools such as Chef or Codar [33]. After continuous build for the application package using Jenkins [30], the framework runs the set of unit tests regression from the CI server and deploy the code to the

development integration environment and execute additional functional test scripts. Tools like Selenium [35] is used for smoke and UI testing. Using CI server, project teams still have the ability to get the output testing results and artifacts of the build, unit testing, and deployment and functional testing along with the source version used for the build for better and continuous improvement. The proposed automated deployment provides a continuous delivery pipeline that automates deployments to development, staging and production environments. This approach significantly reduces the manual intensive tasks, resource lag time and errors prone from manual repetition. This is done via E2E automated deployment tools and processes that aim of reducing deployment risk, and giving the option of deploying code multiple times per day without any degradation in service. The outcomes releases are small in size to first reduce the risk for system instabilities and customer user experience issues, quickly realize the value of the new features to the business more quickly, make the application code change is easier to roll back and easier to test because the number of changes per release is very small.

The proposed DevOps Continuous delivery framework close the loop by integrating the operations aspect as well. This is done via set of tools where operations and

infrastructure teams are using once application package is deployed to the production environment. These set of tools like Codar [32] and Service Management (SM) [33] are used to troubleshoot incidents and continuous monitor across all phases of the application development, testing, and deployment which is crucial for a successful DevOps Continuous delivery implementation. This will facilitate minimizing the costs of errors and changes by providing continuous feedback throughout each phase of the lifecycle. Tools like Splunk[36] are adopted by the proposed framework for log analysis for developers and tools like New Relic to monitor the performance of the applications from the user's perspective such as database-transactions, and systems monitoring to focus on CPU load, memory utilization, and disk space. These tools allow project teams to better understand issues and metrics, and ensures that we are optimizing resources to reduce operational expenditures.

The main benefits realized from the proposed framework can be summarized as follows:

- Design an E2E innovative framework to overcome current legacy approaches issues and drawbacks.
- Adapt new IT trends like converged infrastructure, information, services and delivery approaches to satisfy the market and client needs.
- Enable services flexibility and portability.
- Articulate seamless and lean delivery approaches.
- Utilize available resources/tools to maximize value towards clients.
- Industrialized delivery model to sustain quality while reducing cost
- Innovative approach to align services to the business.
- Narrow down overhead communication between teams and build on collaboration.
- Provides reliability, predictability, and efficiency to ultimately get the most from the applications portfolio.
- Build on project maturity through innovative maturity calculator tool.
- Utilize outcomes from calculator to draft action plan via CSFs/KPIs.
- Automated environment setup toolkit based on push button approach.
- Facilitate smooth/seamless delivery model with all interlocked teams.
- Plugin/customize the tools/resources to fit for project purpose.

- Create new outcomes/value for the clients by composing tools, asset, resources and IT experiences.
- Develop real time instant insights for continuous improvements, innovation.
- Support growth strategy with min time to market.
- Link service offering with business outcomes and client's needs.
- Adopt the 'Smarter rather than harder' theme

IV. FRAMEWORK VALIDATION (CASE STUDY)

To measure how the proposed DevOps continuous delivery framework would help faster releases, case study has been developed and set of measures/metrics are used/proposed according to literature and industry recommendations as follows. There are hard, quantifiable technical and financial metrics we can measure, such as:

- Number and frequency of software releases
- Volume of defects
- Time/cost per release
- Change lead time
- Change failure rate Mean Time Between Failure (MTBF)
- Mean Time To Repair (MTTR)
- Number and frequency of outages / performance issues

The main objective from the case study is to show via quantifiable figures, how the proposed framework over achieve client needs compared to other traditional approaches using the above set of metrics. The specification of the testbed or system under test is based on the following assumptions as:

- Four servers (Dev, Test, Nagios, Database)
- One target is done over a java application.
- Average no of code changes is 4 changes per week
- Average no of environmental changes is 5 changes per week
- 3 ESXI servers with 6 virtual machines with Centos OS
- Installed chef server for provisioning of all servers with development of cookbooks for each machine creation to
- make the creation automated
- Codar topologies are used to configure the automation scripts

With assumptions, the following setup and configuration steps are implemented on the proposed infrastructure to test the outcomes from the proposed framework.

- Build set of virtual m/c to host the new integrated system/applications under the NSIT framework (6 m/c of the following technical configurations).
- 6 Virtual m/c is basically to simulate three different environments (Development, staging, Production).
- Configure and setup the different framework tools (Configuration Management (CM), Jenkins, Git, Codar, SM, Chef).
- Build the application server (front end and backend servers including the Database (DB), Load Balancers (LBs).
- Integrate the application servers (frontend and DBs) hosted on the virtual m/cs.
- Deploy and configure the DB on the new build DB server.
- Deploy the application on the new build application server and ensure the connectivity between the systems is as expected.
- Build the CI/CD (Continuous Integration/Continuous Deployment) servers and link with 'Jenkins' and ensure they are linked with the integrated system of application and DB.
- Build the testing server and deploy set of test cases using 'Selenium'.

Table.1: Proposed DevOps continuous delivery comparison results

Criteria	Waterfall/ad-hoc	Agile	Proposed DevOps continuous delivery
# Releases/month	1 every 6 months	1 every 3 weeks	X per day
#Defects	10	8	2
Change lead time	Months	Days	Mins
MTBF	6 months	16 hours	4 hours
MTTR	6 months	24 hours	6 hours
Outages time	X	5X shorter	10X shorter
Resources Productive time	X	3X	7X
#Changes	X	5X	14X
Change success rate	X	80% X	99.5% X

As shown from the data in table 1, comparing the different release management methodologies starting from legacy/traditional methodologies like waterfall through agile and proposed DevOps continuous delivery, we see huge variance and value where on average 7x times more productive than their non-high performing peers. It produces 14x more changes, with one-half the change failure rate with 4x higher first fix rates, and 10x shorter Severity 1 outages times. Highest deploy rate produced from the framework on the tested application/package was approximately 600 production changes per week, with a change success rate of 99.5%.

V. CONCLUSION

The proposed Continuous delivery DevOps framework goes even beyond the entire SDLC by incorporating the stages after package deployment to production. This paper sheds light and shows how the proposed framework through implementation of automation tools and business processes, releases are being continuously delivered to production systems without outage, higher quality, and unnecessary manual processes. The proposed approach

reduces costs by providing environments that are fully automated thus removing the need for staff to spend time with manual processes. The delivery processes are simple, repeatable and automated to allow for more frequent and less error-prone releases. The proposed framework case study proved the value gained against other traditional approaches especially with current market and business increasing demand via set of benchmark metrics. Which leads to increased efficiencies through improved development and operational processes, minimized and better communication/collaboration between teams, transparency via continuous monitoring and feedback, improved quality from continuous integration and testing, and less risk due to an automated environment?

REFERENCES

- [1] S. Bang, S. Chung, Y. Choh, and M. Dupuis. A grounded theory analysis of modern web applications: Knowledge, skills, and abilities for devops. In RIIT 2013 - Proceedings of the 2nd Annual Conference on Research in Information Technology, pages 61-62, 2013.

- [2] L. Bass, R. Je_ery, H. Wada, I. Weber, and L. Zhu. Eliciting operations requirements for applications. In 2013 1st International Workshop on Release Engineering, RELENG 2013 - Proceedings, pages 5{8, San Francisco, CA, 2013.
- [3] D. Cukier. Devops patterns to scale web applications using cloud services. In Proceedings - SPLASH '13, pages 143{152, Indianapolis, Indiana, USA, 2013.
- [4] P. Debois. Opening statement. Cutter IT Journal, 24(8):3{5, 2011.
- [5] A. Schaefer, M. Reichenbach, and D. Fey. Continuous integration and automation for devops. Lecture Notes in Electrical Engineering, 170 LNEE:345{358, 2013.
- [6] W. Shang. Bridging the divide between software developers and operators using logs. In Proceedings - International Conference on Software Engineering, pages 1583{1586, 2012.
- [7] S. Stuckenberg, E. Fielt, and T. Loser. The impact of software-as-a-service on business models of leading software vendors: Experiences from three exploratory case studies. In PACIS 2011 - 15thPaci_c Asia Conference on Information Systems: Quality Research in Pacic, 2011.
- [8] B. Tessem and J. Iden. Cooperation between developers and operations in software engineering projects. In Proceedings - International Conference on Software Engineering, pages 105{108, 2008.
- [9] M. Walls. Building a DevOps Culture. O'Reilly Media, Sebastopol, CA, 2013.
- [10] J. Webster and R. T. Watson. Analyzing the past to prepare for the future: Writing a literature review. MIS Q., 26(2):xiii{xxiii, June 2002.
- [11] D. DeGrandis. Devops: So you say you want a revolution? Cutter IT Journal, 24(8):34{39, 2011.
- [12] D. Feitelson, E. Frachtenberg, and K. Beck. Development and deployment at facebook. IEEE Internet Computing, 17(4):8{17, 2013.
- [13] L. Fitzpatrick and M. Dillon. The business case for devops: A five-year retrospective. Cutter IT Journal, 24(8):19{27, 2011.
- [14] S. Hosono and Y. Shimomura. Application lifecycle kit for mass customization on PaaS platforms. In Proceedings - 2012 IEEE 8th World Congress on Services, SERVICES 2012, pages 397{398, Honolulu, HI, 2012.
- [15] J. Humble and J. Molesky. Why enterprises must adopt devops to enable continuous delivery. Cutter IT Journal, 24(8):6{12, 2011.
- [16] B. Keyworth. Where is it operations within devops? Cutter IT Journal, 24(12):12{17, 2011.
- [17] B. Kitchenham. Procedures for performing systematic reviews, 2004.
- [18] O. Akerele, M. Ramachandran, and M. Dixon. System dynamics modeling of agile continuous delivery process. In Proceedings - AGILE 2013, pages 60{63, 2013.
- [19] A. Le-Quoc. Metrics-drivendevops. Cutter IT Journal, 24(12):24{29, 2011.
- [20] M. Loukides. What is DevOps? O'Reilly Media, Sebastopol, CA, 2012.
- [21] S. Neely and S. Stolt. Continuous delivery? easy! Just change everything (well, maybe it is not that easy). In Proceedings - AGILE 2013, pages 121{128, 2013.
- [22] B. Phifer. Next-generation process integration: CMMI and ITIL do devops. Cutter IT Journal, 24(8):28{33, 2011.
- [23] H. Pruijt. Multiple personalities: the case of business process reengineering. Journal of Organizational Change Management, 11(3):260{268, Jan. 1998.
- [24] J. Roche. Adopting devops practices in quality assurance. Communications of the ACM, 56(11):38{43, 2013.
- [25] S. Mohamed: DevOps Maturity Calculator DOMC - Value oriented approach, International Journal of Engineering Science and Research, Vol 2, Issue 2, PP 25-35.
- [26] S. Mohamed: DevOps shifting software engineering strategy-value based perspective, International Journal of Computer Engineering, Vol 17, Issue 2, and PP 51-57.
- [27] S. Mohamed: GOAL ORIENTED DEVOPS TRANSFORMATION FRAMEWORK – METRIC PHASED APPROACH, International Journal of Current Research Vol 8, Issue 3, PP 28307-28313.
- [28] S. Mohamed: New style of software lifecycle strategies – Use Case perspective, International Journal of Management, Information Technology and Engineering, Vol 4, Issue 3, and PP 99-114.
- [29] S. Mohamed: Innovative software delivery framework towards software application modernization, International Journal of Research in Engineering & Technology, Vol 4, Issue 5, and PP 77-98.
- [30] <https://wiki.jenkins-ci.org>, May 2016.
- [31] <http://www.tutorialspoint.com/git>, Jun 2016.
- [32] <https://www.youtube.com/watch?v=fVUoWqmuYJM>, Jun 2016.
- [33] https://docs.chef.io/install_server.html, Apr 2016.
- [34] <http://www.tutorialspoint.com/ant/>, May 2016.
- [35] <http://www.tutorialspoint.com/selenium/>, Jun 2016
- [36] <http://www.splunk.com/view/SP-CAAHSM>, May 2016.