ISBN 978-93-84468-80-4
International Conference on Recent Trends in Engineering and Technology (RTET-2016))
Pattaya (Thailand) Dec. 14-16, 2016

Smart Continuous Delivery Framework for Software Releases

Anwin Varghese [1], Dr. Rohini V [2] and Dr. Prabu P [3]

[1] Research Scholar, Christ University, Bengaluru, India
[2] Associate Professor, Christ University, Bengaluru, India
[3] Assistant Professor, Christ University, Bengaluru, India

Abstract--- *In today`s IT world, there is an increasing demand for quality software products that helps business grow, which can in turn help reduce the effort in time & money and also be highly dependable. The challenge of having product releases frequently is not quite an easy task as it sounds. To meet this challenge of producing reliable software products, the IT managers & leads need a team of dedicated developers, system programmers, testers along with a highly efficient process in place. Continuous Delivery (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. CD is attracting increasing attention and recognition.*
The continuous delivery mechanism already has certain frameworks such as agile framework, Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), Composable Fault Tolerance Framework (CFT), Test Orchestration Framework & Large-scale Scrum Framework (LeSS). These existing frameworks do not have any approach to determine or predict the futuristic state of the continuous delivery pipeline. Detecting problems early in development require a new CD approach that speeds up testing & eventually successful releases.
This research work will propose a new framework that can provide error & problem prediction analysis & determine a desired futuristic state in software release lifecycle. The research proposes a new continuous delivery framework with features of early defect recognition through machine learning and that is smart to take informed decisions. This new framework can help the adoption of CD as a practice across IT organizations helping in effective handling of scenarios when the existing frameworks fail. The intent is to generate and maintain a data store to help in decision making for continuous delivery life cycle. The data will be collected & stored & will be mined to be analyzed for success & failure points & concluding on reasons & factors towards the outcomes. The data store will constitute the backbone of the machine learning that will be mined or possible outcome scenarios hence assisting in early detection of problems & help with its resolution in the continuous delivery lifecycle. Extending the adoption of the continuous delivery framework across organizations requires cloud based deployments as most organizations depend on cloud services to host the services. There are a plethora of cloud enablers available & the framework aims to provide the cloud enablement feature to help in its increased adoption.

Keywords: *Continuous Delivery, software, framework, release lifecycle.*

1. Introduction

In today`s IT world, there is an increasing demand for quality software products that helps business grow, which can in turn help reduce the effort in time & money and also be highly dependable. The various different industries such as the automotive, industrial, defence, medical, agriculture are now making use of varied software products & tools that help them in some aspect. This increasing dependency on software products helps the IT industry thrive, grow & be sought after. However, it also puts up an equal challenge for it to be shipping real good quality products quickly to meet the expectations of its end customers. The challenge of having product releases frequently is not quite an easy task as it sounds. To meet this challenge of producing reliable

software products, the IT managers & leads need a team of dedicated developers, system programmers, testers along with a highly efficient process in place. Continuous Delivery (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. CD is attracting increasing attention and recognition.

Also we have the implementation frameworks based on one of these available. Play Framework, Robot Framework and Microsoft Bot Framework are few of the implementation continuous delivery frameworks used in the industry.

We find that the existing frameworks & its implementations do not have any approach to determine or predict the futuristic state of the continuous delivery pipeline. Detecting problems early in development require a new CD approach that speeds up testing & eventually successful releases. The problems in continuous delivery are due to technical gaps in the software development lifecycle, quality gaps in the testing strategy for the software and process gaps while accepting and propagating a flawed software artifact to adjacent environment. These problems will grow with more and more software releases which will eventually cost in the long run.

This research work will propose a new hybrid framework that makes use of artificial intelligence & data mining techniques to get error & problem prediction analysis & determine a desired futuristic state in software release lifecycle. The research hopes to propose a new algorithm that will be the backbone of this proposed new continuous delivery framework. The research will propose a new continuous delivery framework with features of early defect recognition through machine learning and that is smart to take informed decisions. This new framework can help the adoption of CD as a practice across the different IT organizations. It aims to capture the challenges, complexities & benefits of the existing framework implementations & shows how adoption of the new framework helps in handling scenarios when the existing frameworks fail.

The intent is to generate a decision for continuous delivery life cycle by making use of the analysis of data collected over the life time of the software requiring it. The data will be collected & stored to be analysed for success & failure points & concluding on reasons & factors for the outcomes. This data store will constitute the backbone of the machine learning that will be mined for decisions to be made for the continuous delivery lifecycle. Additionally, extending the adoption of the continuous delivery framework across organizations requires cloud based deployments as most organizations depend on cloud to host the services. There are a plethora of cloud enablers available & the framework aims to provide the cloud enablement feature to help in its increased adoption.

2. Literature Survey

IT organizations are moving towards continuous integration and deployment which complements the agile environment. As per Helena Holmstrom Olsson, Hiva Alahyari and Jan Bosch, CD is the ability of a software lifecycle to deliver features to the customers frequently [2]. The continuous delivery mechanism already has certain frameworks that are used in the implementation & to achieve it.

2.1. Waterfall Framework

Waterfall framework is one in which the progress of the delivery cycle of a feature release is taken step by step. The framework follows the same principle of the waterfall software development life cycle. The decision of moving forward with the next step of the delivery for feature release depends on the result of the current step. In the framework time taken for releasing the feature is more. Also, the teams or resources involved will have idle time that translates to resource wastage. This framework fell apart with evolution of parallel development tracks & introduction of multi project shared resources.

2.2. SAFe Framework

The Scaled Agile Framework has been getting a lot of attention lately. SAFe is an interactive knowledge base for implementing agile practices at enterprise scale. At the Team level, SAFe looks a lot like Scrum,

including of extreme programming practices. Not every sprint necessarily produces a potentially shippable increment, but this should happen frequently, possibly after a hardening sprint. At the Program Level, the efforts the agile teams are aligned and integrated to serve the needs of the enterprise and its stakeholders. SAFe provides a fair amount of detail on how to do this. The Portfolio level provides similar product and goal alignment between the investment levels and the operational levels of the organization. Lean thinking, the Principles of Product Development Flow and the extensive benefits that agile development (Agile Manifesto, Scrum, XP technical practices, Kanban) all play important roles in defining the principles and practices of SAFe framework. The framework is one in which the progress of the delivery cycle of a feature release is taken step by step [9].

2.3. DAD Framework

When moving forward with the idea of agile, an excellent framework is required if the project is large scaled. Agility is easy to implement with small development procedure, however, when agility is to be implemented at large scale, projects becomes difficult to handle. Alan. W. Brown, Scott Ambler and Walker Royce discussed that how economic governance, measured improvement and disciplined delivery [3] construct a framework for large scale agile projects. Disciplined Agile Delivery (DAD) is a framework which ensures scalability of agile process. The framework has various characteristics such as: people first, explicit scaling support, goal driven, enterprise awareness, risk and value driven, delivery focused, IT solution focused, agile, hybrid and learning oriented. All these characteristics if embedded properly in the product development procedure will ensure its reliability. Question arises that where does deployment issues are concerned in the DAD framework. DAD framework is the second generation of agile framework; therefore it is an amalgamation of all the excellent features of different agile methodologies. Goal-driven characteristic of DAD, make it aware of the issues that is associated with each goal. This awareness of issues leads the team to look for solution beforehand. Hence at the time of deployment, the realized goals must be associated with the underlying issues. And these issues could be addressed to overcome complications arising due to them. Risk and value-driven characteristic followed by DAD lets the team to identify the common risks and deliver solutions in short span of time. The delivery focused characteristic, manages the post-delivery activities, which is necessary to handle the issues occurring during deployment. Hence, DAD framework's approach vaguely ensures less production issues [9].

2.4. Composable Fault Tolerance Framework

The CFT framework integrates previously constructed components and also has techniques to tolerate fault. The errors in continuous delivery could be minimized through automation. CFT framework is based on workflow model composed of various execution entities. The prime aim of this framework is to uncover errors and make the system fault tolerant. The idea is increment validation, which follows failure identification, automatic fault injection, integration of fault detection/recovery with protected operation and validation of fault tolerance. After increment validation, user provided control data is verified, followed by design of automated testing environment. Having all the future failures sorted out beforehand keeps the team aware of the deployment issues. [1]

2.5. Test Orchestration Framework

Test orchestration as per Nikhil Rathod and Anil Surve is a technique for automated testing and deployment of software work products [5]. It analyses the codes, selects the tests to be conducted, schedules the tests, prepare the environment, executes the tests, analyse the results and finally deploy. Primarily, its aim is to make software reliable and bug free. All the steps involved in the process are automated. This is helpful in identification of bottlenecks early. The visualization of deploying the builds in pre-production environment first before actual deployment is a path that when followed will minimize the production issues. The design of test orchestration framework consists of components such as build automation, test automation, reporting and deployment automation. Build automation leads to a quality software because of reusing components for all

builds. In testing automation, a Test Driven Development (TDD) procedure is followed which tests the work product at every stage. And testing always makes software product reliable. Reporting is a necessary component of test orchestration framework, because previous results are helpful to develop and deploy a build with its help. The deployment automation of the framework makes use of continuous integration as well as continuous deployment, joining them to form a pipeline for life cycle. The framework characteristics that are useful for handling production issues are future prediction and testing the product throughout the development. Future predictions with the help of repository assist in minding the loopholes which could become issues during deployment. Effective testing model a one way, through which bugs in the codes could be revealed, hence a better model will probably minimize production issues. [1]

2.6. LeSS Framework

Large-scale Scrum is regular Scrum applied to large-scale development in very large organizations. The basic roles are unchanged, but some of the meetings are changed and some are replicated at the-cross team level. LeSS involves adding an additional role, the Area Product Owner, who assumes product Ownership of a major section of the product. At this point, an Overall Sprint Review and Retrospective is also added to ensure overall product consistency and process improvement. Sprint Planning may be held with representatives of each team, rather than all members of all teams. Similarly, a cross team retrospective with representatives of each team facilitates overall improvement. Teams are organized as Feature-Teams. Other inter-team coordination meetings may be added, in the form of Scrum of Scrums or Open Space meetings [9].

3. Proposed Framework

The proposed framework is intended to help all the differently sized software teams with the continuous delivery lifecycle. The requirements of a small software team will focus on more agile principles with less effort spent on processes. However, a large team will be more into processes yet being quick with their releases. A medium sized team will have a mix of the two in the sense they will want to have fast paced continuous delivery with good process also in place. The proposed framework aims to assist all the varied needs of these teams. This is achieved by generated a number, we name it the ‘devkata factor’. The framework will define an algorithm to compute this ‘devkata factor’. The ‘devkata factor’ is generated with computation of complexity risk management, bugs in the software release & the time impact of the software release.

Risk Management in agile environment, as quoted by Aalaa Albadarneh, Israa Albadarneh and Abdallah Qusef, is focused minimally by researchers in recent years [6]. RM consists of identification, assessment and prioritization of risks which leads organization to manage events which could turn the project upside down. However, while working in an agile environment, various risks associated with the project fall back. In other words agile reduces risks associated in various areas of project development. Various agile models implements RM in its own specific way. However, DSDM and scrum provides better ways of managing risks than extreme Programming (XP). The effectiveness of RM in agile model could be handful if implemented properly. It could address deployment issues beforehand and provide the team with solutions [1]. From a software program management perspective risk factor depends on the process followed, the structure of team & effectiveness of industry prescribed development standards followed by the software team. The risk factor as we can understand is important as it quantifies the standards followed & gives an idea of the process conformity to industry standards [10].

The continuous testing phase will help discover the metrics with the number of bugs in the software release. Features of test orchestration framework can be used to arrive at the bug factor of the software release. The severity of the bug pays a very important role. It can be safely assumed that high impact, high severe bugs will arrive at a high bug factor which will be detrimental negatively with the going ahead of the release. This also helps the testing folks to always strive for the least bug factor possible. This in turn will lead to better test suits & help software reach a state closer to being bug free.

The time of arrival in market of any software is extremely important for the businesses. If the software is going to miss the time & will take more time then this will be economical disaster to the management. Hence the time factor is the factor that will help management convey the need of required pace of progress with the software release at any point in time.

Depending on the size of the software team, the threshold of these three factors can be set at an optimum required value. The research work will propose an optimum value for the different factors & the eventual 'devkata factor' which will be the final value based on which decisions on continuous delivery lifecycle will be taken.

A data store will be introduced that can be used to store the various factors contributing to the calculation of the three important factors – risk factor, bug factor & time factor which will then give the overall 'devkata factor'. The data store will be the main engine for calculation purpose and will be the main differentiator compared to other existing frameworks. The data store will need data that will contribute to these different factors. The research will arrive at these data points using analysis of the existing metrics in the continuous delivery lifecycle. Once the different data points are identified, arrangement of data in the data store is the challenge that will be solved in the research. The data points need to be stored using a data structure that can help easy storage of data points & also aid in faster calculations of the factors. The calculation of the intermediate factors & the eventual devkata factor will be taking place many times during the continuous delivery lifecycle at the important junctions of the releases. This factor will aid in determining the error analysis, risk analysis & time analysis at the given point & can be an eye opener for deciding the future course of actions.

The aim is to propose a new smart framework that makes use of artificial intelligence & data mining techniques on the data store to understand how to achieve and what it takes to achieve a desired futuristic state in a software release lifecycle. The research proposes a new algorithm that computes the devkata factor that can determine the continuous delivery lifecycle.

4. Conclusion

Continuous delivery of a software in an industry requires to be fine-tuned with all the factors contributing to it required to be measured quantitatively. It needs to be evaluated for the standards followed during the different phases of the lifecycle. The risk factor will help with this aspect and in long run help the teams follow industry standard by default. This will be beneficial with the frequent changing human resources of today`s times. Also, there are additional benefits with metrics that can be generated with following an industry accepted standards for different lifecycle phases. These metrics can be used for improvement in these phases and can a potential way of achieving the required quality state. Similarly, testing the software for potential bugs is an important aspect & bug factor aims to achieve to numerically quantify the testing effort. Also to detect problems early in development there is need for the new CD approach that speedup testing & releases. This research work aims to propose a new hybrid framework that makes use of risk management with artificial intelligence & data mining techniques to get error prediction analysis & determine a desired futuristic state in software release lifecycle. Time to market is very important for market facing software & quantifying the requirement will help management throw the required focus, resources & focus in achieving the economical results. With the time factor we achieve to get this aspect of software release. With these factors quantified, we calculate the devkata factor. The data store which will be the engine to calculate these different quantifiable factors will be the crux of the framework. The arrangement of the data in data store and the various variables and metrics that will be a part of the metrics will be listed in this research. The framework will make use of the data store to arrive at the devkata factor to help teams understand exactly how to achieve and what it takes to achieve a desired futuristic state in a software release lifecycle.

The research will focus on the framework to be host independent of the software releases. Framework can be used in cases of the continuous delivery lifecycle in cloud based providers and its use is not limited to

software teams using traditional premise based resources. The research will propose the formula to arrive at the devkata factor and the new algorithm that will be the backbone of the new continuous delivery framework.

5. References

[1] Aishwarya Vatsa and Shiv Kumar "Software Production Issues and Mitigation Techniques: A review", International Journal of Recent Research Aspects ISSN: 2349-7688, Vol. 3, Issue 2, June 2016, pp. 6-9

[2] Helena Holmstrom Olsson, Hiva Alahyari and Jan Bosch; "Climbing the "Stairway to Heaven"", 38thEuromicro Conference on Software Engineering and Advanced Applications, IEEE, 2012, pp.392- 39.

[3] Alan W. Brown, Scott Ambler and Walker Royce; "Agility at Scale: Economic Governance, Measured Improvement, and Disciplined Delivery", ICSE, IEEE, 2013, pp. 873-881.

[4] Mika V. Mantyla and Jari Vanhanen; "Software Deployment Activities and Challenges – A Case Study of Four Software Product Companies", 15th European Conference on Software Maintenance and Reengineering, IEEE, 2011, pp. 131-139.

https://doi.org/10.1109/csmr.2011.19

[5] Nikhil Rathod and Anil Surve; "Test Orchestration", International Conference on Pervasive Computing, IEEE, 2015.

[6] Aalaa Albadarneh, Israa Albadarneh and Abdallah Qusef; "Risk Management in Agile Software: a Comparative Study", Jordan Conference on Applied Electrical Engineering and Computing Technologies, IEEE, 2015.

https://doi.org/10.1109/aeect.2015.7360573

[7] Humble, Jez & Farley, David. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison Wesley; 1 edition, 27 July 2010

[8] D. Leffingwell and Asd, "SAFe: Scaled Agile Framework," 2014. [Online]. Available: http://www.scaledagileframework.com/.

[9] Peter, "scaling-scrum-safe-dad-or-less," 2013. [Online]. Available: http://www.scrum-breakfast.com/2013/10/scaling-scrum-safe-dad-or-less.html

[10] Bryon Root. "Release Management – Build, Deploy, Test, and Continuous Improvement," 2013. [Online]. Available: http://blog.nwcadence.com/release-management-build-deploy-test-and-continuous-improvement/