SYSTEM DYNAMICS MODELING OF AGILE CONTINUOUS DELIVERY PROCESS

Olumide Akerele
School of Computing and Creative Technologies
Leeds Metropolitan University
Leeds, UK
O.Akerele@leedsmet.ac.uk

Muthu Ramachandran
School of Computing and Creative Technologies
Leeds Metropolitan University
Leeds, UK
M.Ramachndran@leedsmet.ac.uk

Mark Dixon
School of Computing and Creative Technologies
Leeds Metropolitan University
Leeds, UK
M.Dixon@leedsmet.ac.uk

Abstract - The popularization of agile development as well as the recent prevalence of virtualization and cloud computing has revolutionized the software delivery process- making it faster and affordable for businesses to release their software continuously. Hence, the need for a reliable and predictable delivery process for software applications. The aim of this paper is to develop a System Dynamics (SD) model to achieve a repetitive, risk-free and effortless Continuous Delivery process to reduce the perils of delayed delivery, delivery cost overrun and poor quality delivered software.

Keywords - Agile software development, Continuous Delivery, Delivery Pipeline, System Dynamics

I. INTRODUCTION

The most prioritized principle of the Agile Manifesto explicitly emphasizes on the frequent delivery of working software: "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software" [1].

Software only offers value to customers when they are deployed into production and provide the necessary functionalities to the end user. As Humble et al points out: "It's hard enough for software developers to write code that works on their machine. But even when that's done, there's a long journey from there to software that's producing value - since software only produces value when it's in production"[2]. It is therefore vital to ensure that a developed software ends up being deliverable- as that is the utmost goal..

Software delivery suffers as a result of many post-development issues: Configuration management problems, lack of testing in a clone of the production environment and insufficient collaboration between the development teams and the deployment team (operations) are the major problems that cause software rejection at this stage [2]. An example is the complete failure of the software in the production environment due to assumptions built into previous test environments - which are different from the specifications of the production environment. Another example of such problems that stem from these factors is lateness by the operations team to realize they can't support a version of developed software due to the incompatibility of the software architecture with their available infrastructure. The end result: delivery failure.

These problems are the rationale for the need of the expedient continuous deployment of software features into a sandbox, staging or production environment and ensuring the system behaves as required before being classified as "accepted stories" or *"release candidates"*. This practice is called *Continuous Delivery* (CD) or agile delivery [2].

Several measures have been investigated to enhance the CD process: Tests automation, intense team collaboration, configuration management, deployment automation and good team culture[2][10]. However, these factors are not a surety to a smooth CD process; while there have been testimonies of overwhelming success with these practices - as experienced by Flickr and IMVU, with up to 50 deployments a day [4] - there have also been instances of failures [2][19]. This shouldn't be surprising: as is the case with most processes in software projects, there are many limiting factors to the predictability of a software process [5][6].

Various interacting and interconnected factors are present in software projects and these are accountable for the inconsistencies in the software project results [7]. According to Brooks, "no one thing seems to cause the difficulty (in software projects)...but the accumulation of simultaneous and interacting factors..." [7]; hence the adoption of CD. It is the goal of this research to develop a System Dynamics [8] model to study the dynamic effects of these investigated variables within the delivery lifecycle and their relative impact on the success of the CD process. SD stands out as the best approach as it provides the leeway to alter the constituting variables and seeing the behavior and evolvement of software projects under these conditions. Vensim [9], a free SD software is used for this research work.

A. Problem Statement

Factors such as continuous integration, automated tests, good culture and strong collaboration of teams have been identified as the pre-requisites for a successful CD process [2][3][4][10][19]. However, software projects are multifaceted with interrelated problems which make the project outcomes unreliable [5][6]. The success of software delivery is impacted by a host of non-exhaustive factors that interact in a continuous manner - creating revolving loops within software projects.

For example, agile practices like refactoring, pairing, surrogate customers and face-to-face communication impact the cost, quality and schedule of the CD process through the factors described above [2][10]. Refactoring of a test suite, as a practical example, has a dynamic effect on CD process: As the project progresses and acceptance test automation script increases (in size) in direct proportion to the functionalities developed, the test suite complexity, brittleness, as well as coupling increases - gradually introducing *test smell* into the test suite[11].

This is further exacerbated by the influence of schedule pressure and developers respond to such by taking shortcuts, thereby, ignoring the test coding standards and ideals [6]. The test smell effect is a major detriment to the design of the test cases. The directly proportional relationship between these two has a ripple effect on the *build time* to run the acceptance test suite- hence putting the project at risk of late delivery at an increased expense [10]. Another corollary is that a software project with bad designed test suite will perform significantly poorer when judging with the same CD success variables i.e. cost, quality and schedule. However, after refactoring the test suite, there is a significant drop in the test suite maintenance effort due to the improved design of the test scripts [2]. Refactoring, of course comes at a cost of extra effort.

These behaviors as described in the above examples as well as their significance on achieving CD have not been studied in any context - making it difficult for organizations to anticipate the influence of their actions on a CD process. Also, without such understanding of how these factors affect software delivery, it would be difficult for teams to maintain a steady delivery process and optimizing their available resources. A rigorous study of these variables, their dynamic effects and their measurable impact within the *CD system* is therefore vital to ensure repeatability and predictability of an efficient CD process.

B. Related Work

The works related to this research are extensive and span across both empirical and simulation based approaches to studying processes in software development. Kajko-Mattson [12] developed a preliminary process model incorporating the two parts of release management: the vendor and user side. Van Der Hoek et al [13] identified the problems of releasing software from a component-based software engineering approach. The authors developed a tool to solve the identified problems. Krishnan [14] develops an economic model to optimize the delivery cycle of delivering good quality software. Lahtela et al [3] presented the challenges in the delivery of software by performing a full case study. The authors identified 7 different challenges in the release of software.

These works are based on the context of a "big-bang" approach to delivery and not a repetitive delivery process This casts a major doubt on the relevance of the results to agile software projects. More so, these works are empirical based and not simulation based which may indicate to a high degree there is limitation on the control over the identified factors. Though these works give an insight into some of the problems with delivering software, these problems are wholly generic and highly aggregated.

Abdel-Ahmed [5] pioneered the application of SD for software process simulations. He investigated the effect of various management policies on development cycle time, quality and effort were presented. His works however adopt the waterfall methodology approach which limits their applicability in recent software project and more importantly, does not focus on the actual delivery process of software. Melis et al [16] developed a SD model to investigate the impact of TDD and PP on the cycle time, effort and quality of software projects. Cao [21] investigated the dynamics of agile software development and the impact of agile practices on cycle time and customer satisfaction using SD. Though some of these works are contemporary and consider agile practices, their scope is limited to the activities on the developer site and do not consider the actual delivery process.

Hitherto, to the best of the author's knowledge, there hasn't been any published work investigating the dynamics and efficiency of the CD process.

II. Research Focus

The CD phase is initiated when the development team makes a commit to the repository and culminates when the push button for deployment into the production environment is pushed – usually by the operations team. A push button in this context doesn't literally mean a physical button; it is an adopted metaphor in the industry that describes an achievable action by a simple click. The scope of this research thus goes beyond the on-site acceptance testing stage; it protrudes to the actual delivery of features in the production environment.

The aim of this research is to develop a SD model that would be leveraged to achieve a repetitive and predictable delivery process of software while enabling users to have complete control over the delivery risk factors such as cost and schedule flaws. To achieve this, full investigation is to be carried out to determine the pertinent factors that have an effect on the outcome of the CD practices described in section 1. In particular, full investigation of the direct and indirect effects of agile practices on these advocated CD practices within the *delivery pipeline* [17] are carried out.

The *delivery pipeline* [17] encapsulates a series of four stages depending on the extent of testing required by the application. We investigate a generic 4-tier deployment pipeline in our research which could be further modified to fit each varying project requirements: Commit stage, Automated Acceptance Testing (AAT), Manual Testing and Release to Production. The corresponding deliverables of each stage are represented in figure 1 below.

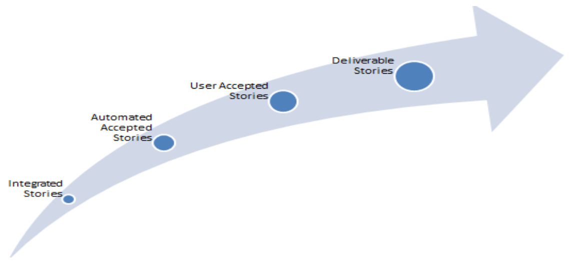

Fig 1: The Delivery Pipeline Artifacts

A. Research Goal

CD lays a strong emphasis on automation of the build pipeline. Humble et al [2] asserts that without automation, the delivery process is not repeatable and predictable. This is somewhat true. However, facts have shown that even with automation and adoption of other "prerequisites" for a smooth CD process, the results are not always a success and performance in different contexts vary significantly[18][19].

Numerous factors which need to be considered and evaluated are responsible for the variation of results in software projects. Madachy[6] presents over 200 active factors that cause instability on outputs of software projects. This is where this research work comes in: to ensure predictability and total control over the CD activities.

The goal of this work is to develop a SD model to act as a tool for the delivery pipeline to ensure a repetitive, predictable and risk-free CD activity for software projects.SD is chosen as the best approach to realizing the goal of this paper as it enables the full design, analysis, and simulation of the software process – fully representing the complexity of the system. The model will ensure a fully controllable delivery environment and help management anticipate the results of their deliberate actions. This model will act as an invaluable tool to project managers, release managers and senior management of software development organizations interested in the frequent release of their software to customers.

B. Research Questions

This paper work is aimed at answering the following major Research Questions (RQ):

RQ1: What are the variables in software projects that have a significant impact on the frequent delivery of software features? These are the environmental, human and technological factors that alter the stability of the process. These variables will be primarily sourced by having interviews with experienced project managers of development teams. Responses will be limited to the experience in context medium sized projects (20-50)kloc . Other methods of finding these variables are highlighted in section III

RQ2: What are the agile practices that have an impact on the CD process? What are the measurable impacts (positive and negative) of these practices on the delivery process?

RQ3: As automation of tests and builds is a major constituent of CD and the build duration is dependent on the number and complexity of tests [10], what are the responses of developers to the duration of the build? Does their behavior influence the number of errors detected?

C. Research Objectives

To realize the goal of this research work, the following objectives have been identified:

- Investigate all the factors that have an impact on the success determining practices of continuous delivery.
- Study the full dynamics of these factors and relevant agile practices on the continuous delivery process.
- Design the system dynamic model for continuous delivery to provide a high level insight into the "actions and reactions" within the CD context.
- Run simulation and compare results for validation.
- Model experimentation for sensitivity analysis.

III. RESEARCH PLAN

This section describes how the objectives of the research work are planned to be achieved. The space constraints of this paper makes it impossible to discuss in full depth each proposed approach.

A. Methodology

i. Data Sources

- Interview: This is the primary technique used to source the variables used to achieve the objectives of this work. Semi-structured interviews will be conducted with experienced agile consultants, project managers and developers. Following interview sessions will be carried out to narrow down the variables to the most relevant factors using a specific approach.
- Literature review: Pertinent empirical findings from literature within the research topic domain will be used to develop and calibrate the model. Search strings such as "continuous delivery (modeling)", "release management (simulation)" and "software system dynamics" will be used in digital libraries.
- Questionnaire: Questionnaires will be developed and sent to a host of agile practitioners. This will include a host of questions that are related to the present practices of the delivery process. The responses will then be analyzed systematically
- Author's discretionary assumption: Where necessary, author's assumptions are used in the development of the model. Such assumptions will be ratified by experienced agile practitioners via interviews and questionnaire.

ii. Simulation

As this is carried out on real life projects, it will be impracticable to have the necessary leeway on the project variables using an empirical approach.

Simulations help to overcome the shortcomings of empirical analysis: cost, flexibility and time consumption [5]. It provides the computerized prototype of an actual system run over time (iteratively) to improve project understanding and knowledge base of project stakeholders.

SD is the modeling and simulation platform used for this work. SD models are used to visualize the complexity of a system in feedback loops to study how the system behaves over a specified period of time [6].

B. Validation

The validation process is quite extensive and is sub-divided into two main stages, as described by Richardson et al: structural validation and behavioral validation [20]. Structural validation involves the inspection of the variables within the model, their calibrations and the designed inter-relationships between them. This is to ensure the structure of the model is “true” and capable of replicating real life project behaviors. Reviewers responsible for validation at this stage will be experienced project managers, consultants and developers. This is expected to be an iterative process incorporating feedback changes into the model.

Behavioral validation checks the model actually produces results that are similar to real life projects. Ideally, the entire model will be validated against data from the delivery history of a specific real life project with similar characteristics. The developed model will be expected to produce similar project behavioral outputs when compared to the real life project .The identified project data will be for medium sized agile projects with the company attaining a minimum of CMMI level 3 accreditation.

REFERENCES

[1] K. Beck et al, "Manifesto for Agile Software Development. Agile Alliance", http://agilemanifesto.org/ (Retrieved 14 Jan 2012).

[2] J. Humble and D.Farley, "Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation", Addison Wesley, 2010.

[3] A. Lahtela and M. Jantti, “Challenges and problems in release management process: A case study,” in 2011 IEEE 2nd International Conference on Software Engineering and Service Science (ICSESS), 2011, pp. 10 –13.

[4] T.Fitz, "Continuous Deployment at IMVU: Doing the impossible fifty times a day", http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/ (Published on 2nd October, 2009, Date accessed 2nd Jan, 2013.)

[5] T. Abdel-Hamid and S.Madnick, "Software Project Dynamics: An Integrated Approach", Prentice Hall, 1991.

[6] R.J. Madachy, "Software Process Dynamics", Wiley-IEEE Press, 2008.

[7] F.P. Brooks, "The Mythical Man Month and Other Essays on Software Engineering", Addison Wesley, 1995.

[8] K. Ogata, "System Dynamics", Prentice Hall, 2003.

[9] Ventana Systems Inc, http://vensim.com/ ,2012

[10] G.Gruver, M.Young and P.Fulghum, "A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware", Addison Wesley, 2012.

[11] R.Borg and M.Kropp, "Automated Acceptance Test refactoring", Proceedings of the 4th Workshop on Refactoring Tools, ACM (2011), 15–21.

[12] M. Kajko-Mattsson and F. Yulong, “Outlining a Model of a Release Management Process,” J. Integr. Des. Process Sci., vol. 9, no. 4, pp. 13–25, Oct. 2005.

[13] A. van der Hoek and A. L. Wolf, “Software release management for component-based software,” Softw. Pract. Exper., vol. 33, no. 1, pp. 77–98, Jan. 2003.

[14] M. S. Krishnan, “Software release management: a business perspective,” in Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research, 1994, p. 36.

[15] Madachy, R.J. System dynamics modeling of an inspection-based process. , Proceedings of the 18th International Conference on Software Engineering, 1996, (1996), 376 –386.

[16] M.Melis, I.Turnu, A.Cau and g.Concas, "Evaluating the impact of test-first programming and pair programming through software process simulation. Software Process: Improvement and Practice 11, 4 (2006), 345–360.

[17] J.Humble, C. Read, and D.North, "The deployment production line", Agile Conference, 2006, (2006), 6 pp. –118.

[18] M.W. Mantle and R.Lichty," Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams", Addison-Wesley Professional, 2012.

[19] L. Klosterboer, "Implementing ITIL Change and Release Management", 1st ed. IBM Press, 2008

[20] G. P. Richardson and A. L. P. III, "Introduction to System Dynamics Modeling", Pegasus Communications, 1981.

[21] L.Cao, B.Ramesh, and T.Abdel-Hamid," Modeling dynamics in agile software development". ACM Trans. Manage. Inf. Syst. 1, 1 (2010), 5:1–5:26.

[22] M. C. Paulk, "The capability maturity model: guidelines for improving the software process", Addison-Wesley Pub. Co., 1995.