Evaluating the Impact of Critical Factors in Agile Continuous Delivery Process: A System Dynamics Approach

Olumide Akerele
School of Computing and Creative Technologies
Leeds Metropolitan University
Leeds, UK

Muthu Ramachandran, Mark Dixon
School of Computing and Creative Technologies
Leeds Metropolitan University
Leeds, UK

Abstract—**Continuous Delivery is aimed at the frequent delivery of good quality software in a speedy, reliable and efficient fashion – with strong emphasis on automation and team collaboration. However, even with this new paradigm, repeatability of project outcome is still not guaranteed: project performance varies due to the various interacting and inter-related factors in the Continuous Delivery 'system'. This paper presents results from the investigation of various factors, in particular agile practices, on the quality of the developed software in the Continuous Delivery process. Results show that customer involvement and the cognitive ability of the QA have the most significant individual effects on the quality of software in continuous delivery.**

***Keywords**—Agile software development; Continuous Delivery; Delivery Pipeline; System Dynamics*

I. Introduction

The Agile Manifesto places a high importance on the need for the frequent delivery of working software: “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software” [1]. This subtle principle indicates that not all developed software is actually made available to the customer for use where it actually adds value to the customer's business. 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].

Software delivery is inhibited by a number of post-development issues: Configuration management problems, insufficient testing in production-like environment and poor collaboration among the various 'silos' in software projects are the major problems that cause software rejection at this Stage [2]. A practical example of such problem is the 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. This is strictly owed to the lack of involvement and collaboration of the operations team in the development process, thus, resulting in delivery failure. Such post-development problems are the motivation for the *Continuous Delivery* (CD) initiative [2][4][10].

Tests automation, strong team collaboration, effective configuration management, deployment automation and good team culture [2][10] are the major practices advocated in CD to boost the effectiveness of a frequent delivery process . However, these factors are not a surety to a smooth CD process; while there have been overwhelming testimonies of success with these practices ,most notably by Flickr and IMVU – with up to 50 deployments a day [4], there have also been numerous instances of failures [2][19]. This shouldn’t be surprising: project outcomes in software projects is faced by many limiting factors [5][6].

Various interacting and interconnected factors are present in software projects and these are accountable for the inconsistencies in the quality of 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].

The primary goal of this work is to investigate the dynamic causal relationships of the variables within the CD 'system' and develop a *System Dynamics* (SD) [8] model to evaluate the impact of these pertinent factors on the quality of software projects adopting CD. This can be used as a tool to evaluate various managerial decisions and introduce reliability, predictability and risk aversion in the CD process. Vensim [9], free SD software is used for this research work.

A. Problem

Continuous integration, tests automation, good culture and strong collaboration have been identified as the "pre-requisites" for a successful CD process [2][3][4][10][19]. However, software projects are daunted with several interrelated problems which make the project outcomes unreliable [5][6] – even with the adoption of the aforementioned "CD success pre-requisites" [2]. 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 [5].

Refactoring of an automated acceptance test suite, as an example, is hypothesized to have a causal and dynamic effect on CD process: As the acceptance test automation script increases linearly with the project progress, the test suite complexity, brittleness, as well as coupling increases –

gradually introducing *test smell* into the automated acceptance test suite [11]. This is worsened by the presence of *schedule pressure;* developers take short cuts by ignoring the test coding standards and ideals in order to meet up with the estimated work [6]. The *test smell* effect has a negative ripple effect on the maintenance effort of writing automated acceptance tests [10]. However, after refactoring the test suite, there is a significant reduction in the test suite maintenance effort due to the improved design of the test scripts [2][10]. Refactoring, of course, comes at a cost of extra effort [12].

Such causal effects of various practices are the determinants of the failure and success of CD and there is a wide gap in academic research within this context [18]. Without the managerial proactiveness of the effects of various practices at various times in software projects, software delivery will continue to be uncontrollable, leading to many unpleasant surprises. A rigorous study of these variables, their dynamic effects and their impact on the quality of the developed software is vital to ensure repeatability and predictability of an efficient CD process.

B. Literature Review

CD is a relatively new paradigm; this explains the reason for the paucity of research work done in this field. At the time of writing this paper, there isn't any research work categorically done on CD. However, some works have been done within considerably similar context: Kajko-Mattson [12] developed a preliminary process model incorporating the two parts of release management: the vendor and user side. Lahtela et al [3] presented the challenges in the delivery of software by performing a full case study. The authors identified 7 different challenges encountered in the release of software. 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] developed an economic model to optimize the delivery cycle of delivering good quality software. These works adopt a big-bang traditional waterfall approach to delivery and not a repetitive delivery process – as is the case in an agile development. This casts a major doubt on the relevance of their findings to agile software projects. More so, these works are empirical based and not simulation based which indicates to a high degree that 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.

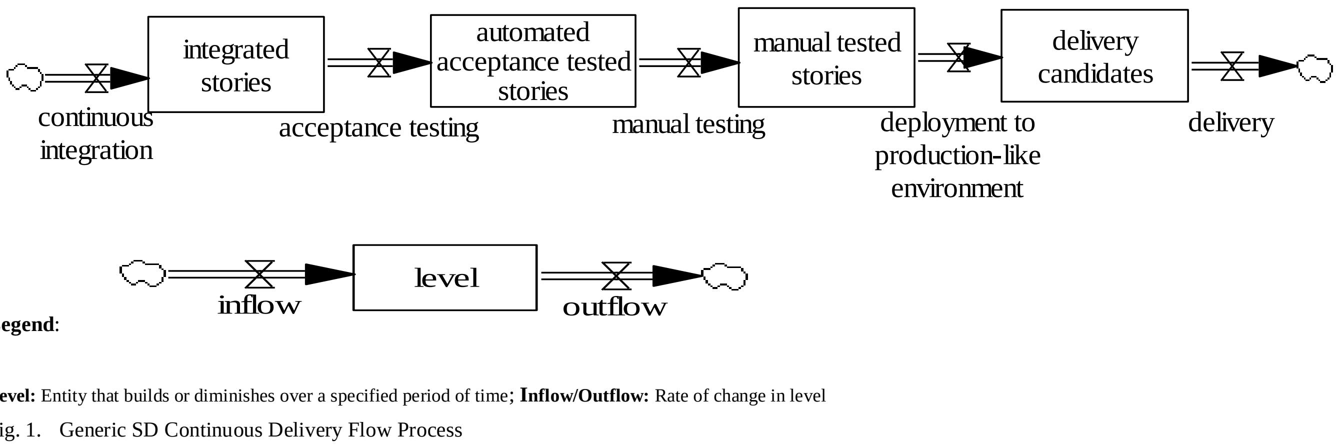
Abdel-Ahmed [5] was the first researcher to leverage SD in software process simulations. He investigated the effect of various management policies on development cycle time, quality and effort were presented. However, his work was based on the waterfall methodology approach which confines the applicability of the results to waterfall projects. The actual delivery process in software projects is also beyond the scope of his work. Melis et al [16] developed a SD model to investigate the impact of Test Driven Development (TDD) and Paired Programming (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. With credit to the impact of the work done by these authors in agile software development, their works do not consider any post-development activities relevant to software delivery. Furthermore, there is complete exemption of the impact of schedule pressure experienced by software project teams.

The authors of the paper assert that the successful conclusion of this research work is going to be a pioneering development in the field of CD and will create further insights in which new research interests can evolve.

II. Research Scope

This research aims to develop a SD model that delivery practitioners can adopt to have control over the delivery risk factors, particularly cost overrun and schedule flaws. Achieving this aim involves full investigation to determine the pertinent factors impacting the outcome of the CD practices described in section 1; the causal effects of agile practices on these advocated CD practices within the *delivery pipeline* [17] are also considered.

Fig.1 below presents an overview of the generic flow process of CD. The entire process line is known as the *delivery pipeline, deployment pipeline or build pipeline* [17]. The complexity of the pipeline created by teams will vary depending on the level of available resources, project risks involved and criticality of the developed software [35]. This research work is based on a standard 4-stage deployment pipeline as represented in Fig. 1.



Legend:

Level: Entity that builds or diminishes over a specified period of time; **Inflow/Outflow:** Rate of change in level

Fig. 1. Generic SD Continuous Delivery Flow Process

Mathematically, the legend above is exemplified by:

$$level\ (t)= level\ (t-dt) + inflow * (dt) \qquad (1)$$

Fig.1 above shows each activity and their corresponding artifacts. The success of each initial stage is a criteria for the commencement of the succeeding stage. Our work lies in this pipeline to determine the relevant factors affecting the efficiency of this 'journey' for frequent software delivery.

A. *Research Goal*

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. The model will ensure a fully controllable delivery environment and help management anticipate the results of their deliberate actions.

B. *Research Questions*

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

RQ1: What are the key factors (environmental, human and technological)in software projects that impacting the success of CD? What are the agile practices that have an impact on the quality of software projects in the CD process?
RQ2: What are the dynamic and causal effects of each of these factors on the software quality in CD?
RQ3: What is the impact of the agile practices such as on-site customer, TDD,PP and Pair Testing on software quality in CD? What is the impact of the ability of the Quality Assurance (QA) tester on the quality of the software

C. *Research Benefits*

A number of benefits would be achievable from the success of this research work: Firstly, it will help to maintain a total control of the available resources to achieve a stable, repeatable and predictable CD process. The lack of such tool has created a huge gap in the industry and made delivery stability a difficult task. The stability that is realizable with this tool will help organizations striving to achieve CMMI levels 4 and 5 [22] accreditation.

This model may also be used as a risk management tool of the delivery process. Since the impact of potential technological and strategic decisions on outcomes such as project completion dates and number of deliverable features is possible via simulations, potential risks can be anticipated and proactively planned against or avoided completely. Several software organizations depend mainly on SD models as their major risk management tools [5].

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.

In addition, the model can serve as a process improvement tool by helping to determine points for optimization of important variables like acceptance rate, build time, required effort, and etcetera.

III. Methodology

This section describes how the objectives of the research work are planned to be achieved.

A. *Data Sources*

- Interview: Primarily, semi-structured interviews will be conducted with experienced agile consultants, project managers and developers to elicit the major active variables to achieve the objectives of this research work. A formal approach will be adopted to narrow down these factors to the most relevant active factors.

- Questionnaire/Survey: This will be developed and sent to practitioners within the CD field who will give their responses based on the valuable experience in the area. The responses will then be analyzed systematically.

- Literature review: Keywords such as "continuous delivery (modeling)", "release management (simulation)" and "software system dynamics" will be used to search for related work in digital libraries. Significant findings from related work will not only help in identifying some factors but also help in the quantification of the impact the factors have on other variables in the project. The quantification of this impact will be vital in the calibration of the SD model for simulations.

- Author’s discretionary assumption: Where necessary, author’s assumptions are used in the development of the model. Such assumptions will be sanctioned and perhaps, moderated by experienced agile practitioners via interviews and questionnaire.

B. *Simulation*

Simulations provide the computerized prototype of an actual system run over a specified period of time. They are useful in software projects to improve project understanding and knowledge base of project stakeholders.

Simulations offer a more realistic and cost-effective approach to realizing the objectives of this work as opposed to the 'rigidity' offered by empirical methods. The flexibility provided by simulation techniques to alter the variables for system behavior analysis will be impractical to achieve if the conventional empirical methods are adopted [5][15].

SD, a continuous simulation technique, provides the full functionalities to achieve the goals and objectives of this research work, hence, its adoption for this work. SD facilitates the visualization of the complex inter-relationship between variables in a software project system and runs simulations to study how complex project systems behave over time [6]. A system dynamic model has a non-linear mathematical structure of first order differential equations expressed as:

$$y'(t)= f(y,m) \ (2),$$

Where y represents vector of levels, f is a non-linear function and m is a set of parameters.

IV. Continuous Delivery Model and Parameterization

The full CD model is designed into three sub-models for ease of analysis: The schedule pressure sub-model, the delivery pipeline sub-model, CD cost effectiveness sub-model. Due to space constraints of this paper, only the automation acceptance testing section of the delivery pipeline sub-model is presented in this paper. This section is responsible for estimating the *AAT Pass Multiplier*.

A. *The AAT PASS Multiplier*

This sub-model was designed to determine the impact of various policies on the quality of automated acceptance tests. Schedule pressure -- an occurrence triggered when actual time left to finish the development of the software exceeds the estimated time to finish the development of the software- plays a pivotal role in the level of adoption of process improvement practices [5]. The Figure below shows the dynamic modelling of factors responsible for the quality of the automated acceptance tests.

The authors have solely discussed the elicitation and calibration of the *tdd factor* only as space constraints of this paper makes it impossible to discuss all the active variables in the model

TDD as a development technique has been a core practice in agile software projects. TDD involves a sub-iterative and incremental 6-step process in the following order: *write failing unit test* - *run to ensure failure* - *write functionality code* - *re-run unit test to ensure success* - *refactor* - *proceed*. This iterative and incremental procedure instills a high degree of reliability into the developed software and reduces redundancy in the production code and test artifacts [23].

Some researchers have investigated the impact of tdd on the quality of automated acceptance tests: A recent research investigated the effects of TDD on external quality and productivity by using meta-analytical techniques [24]. Results of the analysis suggest that the TDD has a relatively small positive impact on the quality of software; however, the impact on productivity is non- conclusive.

George and Williams [25] carried out a controlled experiment on 24 professional pair programmers to evaluate the external code quality and speed of development of the TDD adopters vis-a-vis waterfall approach adopters. 3 experiments were performed on 8 person-group teams at 3 different companies to program Martin's bowling game task [26].

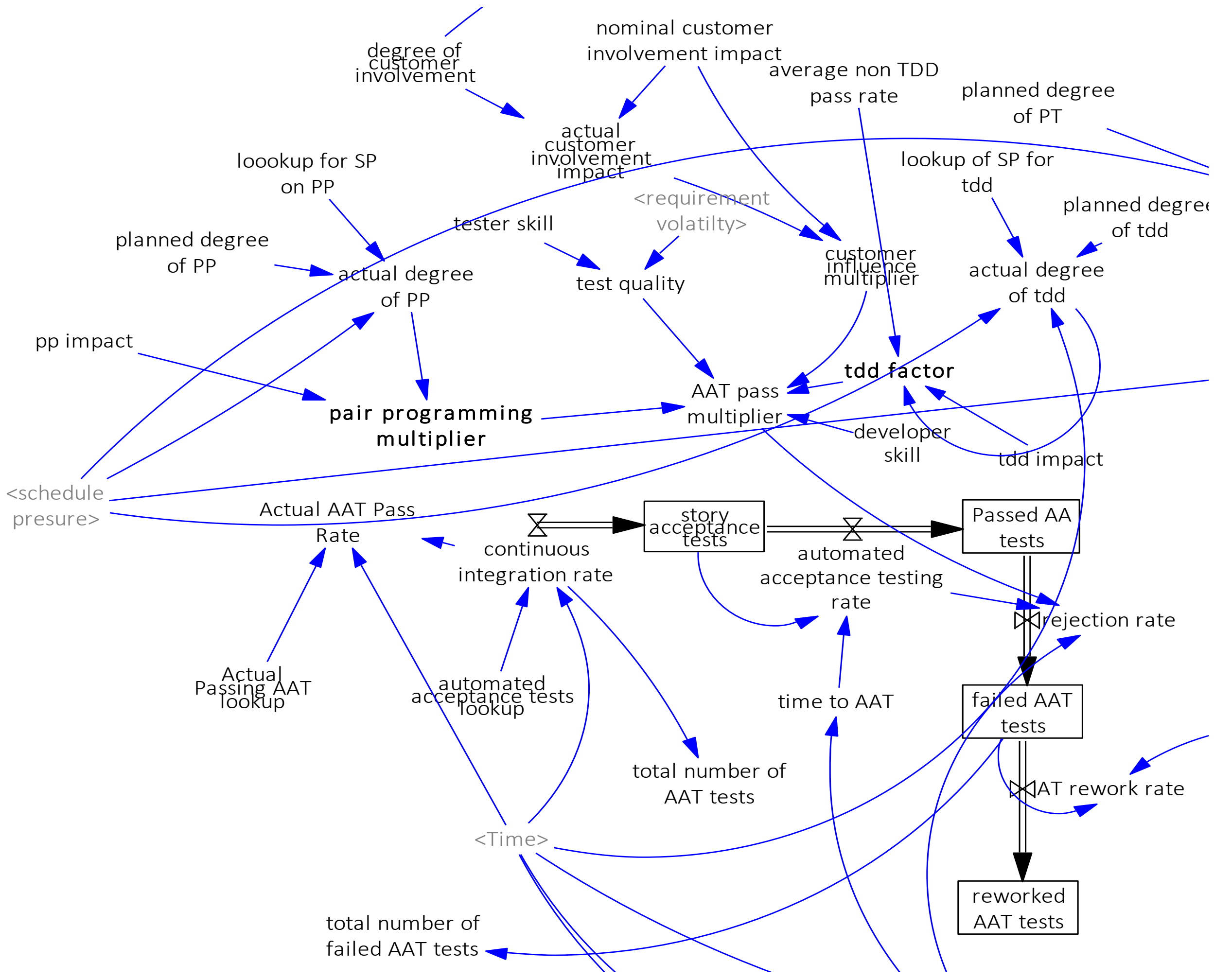


Fig. 2. Automated Acceptance Testing Section of the Delivery Pipeline Sub-Model

Results showed the tdd group passed roughly 18% more tests than the control group that adopted the waterfall approach. The results in the work are however may have been confounded by the effects of PP.

A more relevant experiment was carried out on by Yenduri [27]. A two 9-person groups of senior undergraduate students to evaluate the impact of tdd on software quality and productivity; one group used the test first approach and the other using the test-last approach. Results showed 55% improvement in the acceptance test pass rate of the tdd group. The task developed however was described as a "small" project which implies results could vary with larger projects. Also, the size of the subjects is relatively small.

A survey was conducted by the authors directed at experienced project managers and developers with over 5yrs experience in tdd usage to determine the impact of tdd on acceptance tests success. Analysis of the responses gave values ranging from 30% to 60% improvement, with the mean of approximately 54%. The authors assumed a modest value of 50% and this value was further supported by two interviewees. One of the interviewees (I1)said:

"The high level of granularity of functionality testing during TDD, when done effectively, guarantees the behavioural requirement of the system is fulfilled, severely limiting the causes of failures during functional testing to environmental factors or inaccuracy in the requirement elicitation. Conveniently, we achieve 50% better success during acceptance testing than when we used to adopt the test-last approach... . "

Baring other factors, the authors make a bold assumption that a full adoption of TDD should guarantee 100% success of acceptance tests. Hence, we estimate the success rate of acceptance tests pass rate without TDD to be 67% so that a full adoption of TDD will yield 100% success in our model. The estimated acceptance pass rate of the test –last approach is represented by the *average non-TDD pass rate* in the model. This value is quite close to values of test-last adopters (75%) in Williams' work [25].

The planned degree of tdd in the model represents the planned level of tdd adoption in the development of the software features for the project. No literature exists on the average level of tdd adoption. However, the standard degree of unit test coverage in the industry ranges from 80-90%. [28].Some platform providers maintain a strict level of unit test coverage before allowing promotion of software unto their platform. Sales force, a leading PaaS provider, insists on minimum unit test coverage of 75% before allowing promotion of customer's software unto their staging environment [29]. In our project case study, the planned level of tdd adoption is 100% for the project i.e. all features were planned to be developed by tdd approach.

For ease of analysis, we assume that the test suite offers complete coverage; implying all behavioral defects in the system are detected during development when tdd is fully adopted. This follows a similar assumption made by Williams et al in the development of their economic model [30].

The actual degree of tdd adoption is affected by schedule pressure [6][23]. Developers tend to "cut corners" when the team is behind schedule to try and catch up . When a team is behind schedule, the procedural steps f adopting tdd are easily bypassed to increase development speed. The *actual degree of tdd* is the effective percentage of features developed using the tdd approach in the project throughout the project. There is no published work on the estimated impact of schedule pressure on the degree of tdd adoption prompting the authors to derive simplistic mathematical model to estimate the impact of schedule pressure on the planned degree of tdd adoption. Effective and simple mathematical models can be developed by researchers when reliable data is not available for model parameterization. Forrester advised:

"A mathematical model should be based on the best information that is readily available, but the design of a model should not be postponed until all pertinent parameters have been accurately measured. That day will never come. Values should be estimated where necessary…." [31]

As the schedule pressure develops, the team responds to falling behind in schedule by working extra hours and cutting their slack time to try and meet up with the lost work [32]. This makes the initial effect of SP very minute, hence, the initial flatness in the curve. However, as the pressure mounts, the "threshold" is exceeded and the team responds by cutting corners and reducing their adoption of TDD steps, instead, following the test last approach. It then gets to a maximum point where a increase in SP doesn't have an effect anymore. This forms the tail end/flat end of the other extreme end of the graph. This relationship is built in the variable *lookup of SP for tdd.*

tdd factor, the variable representing the impact of tdd on the estimated pass rate of the automated acceptance tests has the formula : *average non TDD pass rate+(actual degree of tdd*tdd impact*average non TDD pass rate*)

where *actual degree of TDD = IF THEN ELSE(Time=11, 0, planned degree of tdd*lookup of SP for tdd(schedule presure)), TDD impact = 0.5 and average non TDD pass rate=0.67.*

V. Model Validaton

The model is validated in two folds, following the approach described by Richardson et el [20]: structural phase and behavioral phase. *Structural validation* is the examination of the structure of the entire model. This involves the studying of the inter-relationship and parameterization of the variables to ensure they are credible enough to produce replicate real-life scenarios. Experienced project managers, consultants and developers were sought for this process, with critical feedback used to rework the model in an iterative manner until the structure is approved by the reviewers. The model was also presented at two conferences and valuable feedback was incorporated to rework the model.

Behavioural validation aims to verify the model actually produces results that are similar to real-life project outputs. The model will be validated against data of output variables from a completed software project with similar characteristics that successfully implemented CD. Coherence in the results between the simulation outputs and actual completed project outputs prove the model is capable of producing real-life project results, hence, validating the model. Also Success at this stage is a critical prerequisite before the model could be subjected to *sensitivity analysis* to answer the remaining research questions outlined.

A. *Project Data*

Data was sourced from a complete project that adopted CD and agile practices from a sales software vendor. The developed software is part of a comprehensive software suite used for enhancing sales of products by manufacturers.

The project case study used for this project was the software development project for their sales modeling solution. Data from the project is presented in the table 1.

Table 2 presents the data used to simulate the pressure experienced by the team. The pressure influences the adoption of major practices in the model and consequently the outcome of the project [5][6]. Fig. 5 below shows the simulated graphical representation of the SP experienced by the team. Schedule pressure is determined across each iteration in the project by the formula:

(*Actual Work Left - Estimated Work Left)/ Estimated Work Left* (3).

TABLE I. GENERAL PROJECT INFORMATION

Programming Language	Java
Project Duration	220 working days
Development Duration	203 working days
Iteration duration	2weeks
Team Size	5
Team Velocity	50
Agile Methodology Used	XP/Scrum
Number of Stories	199
Version Control System	Subversion
CI Server	Go
Configuration Management Tool	Chef
Unit Test Framework	JUnit
Automated Acceptance Testing Framework	BDD
Automated Acceptance Testing Tool	JBehave
Team Experience Mix	Average of 9years software projects experience
Working hours/day	7.5

TABLE II. PROJECT DATA USED FOR SCHEDULE PRESSURE SIMULATION

Iteration #	Estimated No of Tasks Committed	Actual No of Tasks Committed	Actual # User Stories Completed	Actual Value of Work Completed(Points)	Production Code Size (LOC)
1	30	20	6	62	450
2	40	28	7	55	695
3	40	46	11	60	1123
4	40	44	11	51	1095
5	40	41	10	49	960
6	40	43	10	58	1145
7	40	38	9	62	967
8	40	34	8	41	888
9	40	27	6	47	620
10	40	25	7	50	540
11	20	0	0	59	0
12	40	53	14	61	1322
13	40	55	13	65	1485
14	40	48	12	64	1055
15	40	46	11	60	912
16	40	41	10	58	993
17	40	46	12	66	1211
18	15.6	53	15	69	1368
19	0	56.96	17	63	1420
Total	**705.6**	**744.96**	**199**	**1099**	**18249**

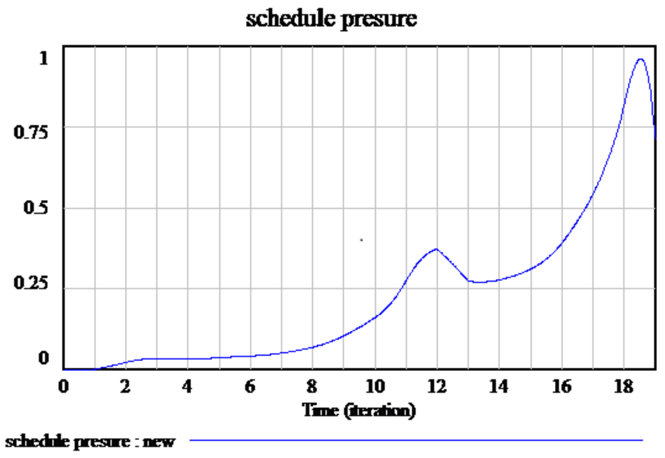


Fig. 3. Project Schedule Pressure

B. *Simulation Results*

The table below shows the extracted results from the simulation model. "AA" denotes “automated acceptance" in table 3.

TABLE III. RESULTS COMPARISON OF ACTUAL PROJECT OUTCOME AND SIMULATED PROJECT OUTCOME

Iteration	Actual # of Passing AA test cases	Actual AA Pass Rate	Simulated # of Passing AA Test Cases	Simulated AA Pass Rate	Actual # Passing UA test cases	Actual UA Test pass rate	Simulated # passing UA test cases	Simulated UAT pass rate
1	31	83.78	33.83	91.44	23	74.19	21.37	68.94
2	40	80	44.30	88.61	26	68.4	26.58	69.96
3	62	89.85	60.14	87.17	36	75	34.51	71.90
4	55	82.08	59.77	89.22	39	82.9	34.95	74.39
5	58	90.62	56.75	88.68	40	81.63	37.58	76.70
6	63	90	62.84	89.78	50	87.71	45.15	79.22
7	57	95	52.29	87.15	41	75.92	43.99	81.47
8	44	83.01	46.71	88.14	48	87.27	45.77	83.23
9	33	84.61	34.54	88.57	39	81.25	40.46	84.31
10	42	91.30	39.50	85.89	40	90.9	36.93	83.95
11	0	0	0	56.20	0	0	0	68.34
12	46	63.01	53.51	73.31	35	59.32	39.38	66.76
13	56	80	52.24	74.6	38	71.69	37.97	71.64
14	47	75.80	46.48	74.98	41	68.33	43.43	73.33
15	38	69.09	39.67	72.13	37	75.51	34.51	70.44
16	43	79.62	38.64	71.56	42	76.36	37.73	68.60
17	36	70.58	35.64	69.89	31	63.26	33.19	67.75
18	50	76.92	46.58	71.66	35	64.81	36.69	67.95
19	49	69.01	51.85	73.04	51	82.25	42.25	68.15

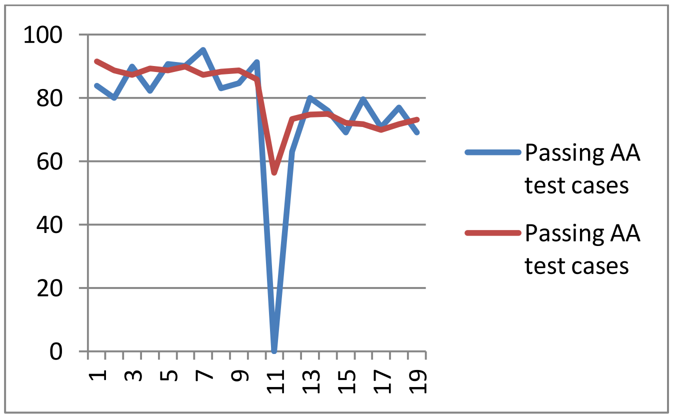


Fig. 4. Graphical Comparison of Actual and Simulated Automated Acceptance test (AAT) Pass Rate

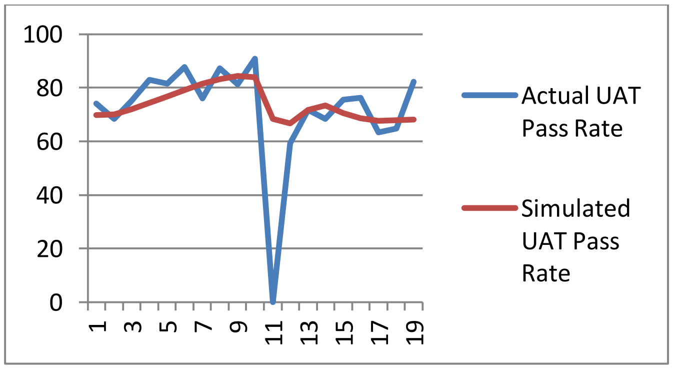


Fig. 5. Graphical Comparison of Actual and Simulated User Acceptance Test Pass Rate

The data provided in table 3 is used to examine the validity of the model by comparing the actual project outcome with the outcome produced by the developed simulation model. The actual automated acceptance test pass rate and simulated automated acceptance test pass rate represents the actual number of passing automated acceptance and user acceptance test cases expressed as a percentage of the total number of automated acceptance and user acceptance test cases and simulated number of passing automated acceptance and user acceptance test cases expressed as a percentage of the total number of automated acceptance and user acceptance test cases respectively.

Noticeably, the results from the model highly correlate with the actual project outcome. There were two main points of significant discrepancy in the values of the results for AAT results: The 1st, 2nd and 12th iteration. In the first and second iteration, the team recorded a low number of automated acceptance test cases due to the relatively few number of stories delivered which significantly reduced the total sample for that iteration. Hence, the high impact on the % variation between the simulated and actual results. It is plausible to believe that the actual pass ratios for these iterations with low test cases are exaggerated. In the 12th iteration, the team had significantly more actual failing tests due to the impact of major refactoring on the passing test suite which occurred in the 11th iteration. It has been reported that that software project teams generally experience problems of failing tests after major redesign due to the coupling among various components of the software [33]. The 11th iteration is not recognized as a non- productive iteration by the simulation model as the actual project progress was inhibited due to the management decision to carry out major refactoring. This behaviour is not built into the simulation model as this is a manual decision made solely by the discretion of the team.

The major point of disparity in the simulated and actual pass rate in the UAT scenario is apparent in the 19th iteration. A possible argument for this is that testers tend to overlook many possible scenarios when a project is seemingly coming to closure and build assumptions into the system to get the project over with; in extreme cases, testers actually pass failing tests and are not really ready to find faults to avoid project extension and look forward celebrating project completion. This phenomenon was further attested by an interviewee (I2). This phenomenon may explain the considerable disparity in the passing test rates in the final iteration than that projected by the simulation model.

C. Model Experimentation

Experiments are performed to carry out sensitivity analysis on the model to determine the impact of various policies on the quality of the developed software by altering the planned level of adoption of the major influencing agile practices. The major practices of interest are: PP, PT, customer involvement and TDD. The impact of the ability of the QA (cognitive ability and domain savvy) is also investigated. Schedule pressure plays a prominent role in the actual level of adoption of the practices. The project data used for the model validation is used and the level of adoption of each practice is altered. Table 4 below shows the various scenarios typifying various managerial policies regarding agile practices adoption.

TABLE IV. SCENARIOS FOR AAT MODEL SUB-SECTION

	PP	TDD	Customer involvement
Scenario 1	0%	0%	0%
Scenario 2	100%	0%	0%
Scenario 3	0%	100%	0%
Scenario 4	0%	0%	100%
Scenario 5	100%	100%	0%
Scenario 6	100%	0%	100%
Scenario 7	0%	100%	100%
Scenario 8	100%	100%	100%

TABLE V. SIMULATED AUTOMATED ACCEPTANCE TEST PASS RATE

Iteration #	**Scenario #**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	22.7	25.9	34.1	56.9	38.9	64.9	85.3	97.4
2	2m1.9	25,3	32.9	54.9	37.9	63.2	82.3	94.8
3	21.6	24.9	32.3	54.0	37.3	62.4	80.8	93.4
4	22.1	25.5	33.2	55.4	38.1	63.7	83.0	95.4
5	22.0	25.3	32.9	55.0	37.9	63.4	82.4	94.9
6	22.3	25.6	33.4	55.8	38.3	64.1	83.6	95.9
7	21.6	24.9	32.4	54.1	37.3	62.3	81.0	93.2
8	21.9	25.2	32.8	54.9	37.6	63.0	82.0	94.1
9	22.1	25.3	33.0	55.3	37.8	63.3	82.5	94.5
10	21.8	24.7	32.1	54.6	36.5	61.9	80.4	91.2
11	21.4	23.5	21.4	53.6	23.5	58.7	53.6	58.7
12	22.3	22.5	29.2	55.8	29.4	25.2	73.0	73.5
13	21.7	22.4	29.3	54.2	30.3	56.0	73.3	75.8
14	21.7	22.5	29.4	54.3	30.5	56.2	73.6	76.3
15	21.4	21.8	28.5	53.6	29.1	54.7	71.3	72.8
16	21.8	21.9	28.5	54.6	28.7	54.9	71.3	71.7
17	21.5	21.5	27.9	53.7	27.9	53.7	69.8	69.8
18	22.0	22.0	28.6	55.1	28.6	55.1	71.6	71.6
19	22.4	22.4	29.2	56.1	29.2	56.1	73.0	73
Average	21.9	23.84	30.5	54.8	30.4	59.67	76.5	83.6

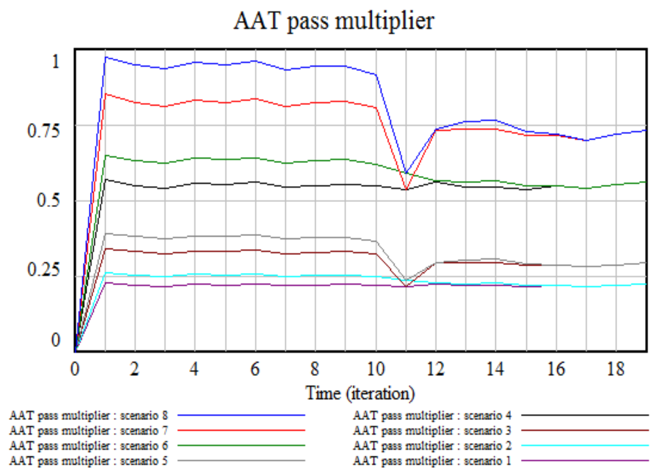


Fig. 6. Graph for Automated Acceptance Test Rate for Various Scenarios

Table 5 and Fig. 6 show the relative impact of applying various managerial policies on the quality of the software with the values rounded off to the nearest 1decimal point. Table 1 above shows the impact of various management policies on the AAT pass rates. Clearly, scenario 8 (adoption of al practices) provides the most outstanding results until iteration 5 when it levels up with scenario 7 (TDD and customer involvement).

While scenario 4's performance (customer involvement) is not the best, it provides the most stable pass ratios all through the project irrespective of the schedule pressure. Unsurprisingly, scenario 1 had the poorest results having not adopting any of the practices.

TABLE VI. SCENARIOS FOR UAT MODEL SUB-SECTION

	PT	**QA Cognitive Ability**	**QA Domain Knowledge**
Scenario 1	0%	low	low
Scenario 2	100%	low	low
Scenario 3	0%	high	low
Scenario 4	0%	low	high
Scenario 5	100%	high	low
Scenario 6	100%	low	high
Scenario 7	0%	high	high
Scenario 8	100%	high	high

TABLE VII. SIMULATED UAT PASS RATE

Iteration #	**Scenario #**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	50.6	57.0	58.3	53.9	68.3	61.9	61.5	69.5
2	51.3	57.8	59.1	54.7	69.3	62.7	62.4	70.5
3	52.7	59.2	60.6	56.1	71	64.3	64	72.5
4	54.5	61.3	62.7	58	73.5	66.5	66.2	75
5	56.2	63.2	64.7	59.8	75.8	68.6	68.3	77.3
6	58.1	65.2	66.8	61.8	78.2	70.8	70.5	79.9
7	59.7	67.1	68.6	63.5	80.4	72.8	72.5	82.1
8	61	68.5	70.1	64.9	82	74.3	74	83.9
9	61.7	69.2	70.9	65.6	82.8	75	74.8	85
10	61.6	68.4	70.3	65.3	81.2	73.9	74	85.3
11	59.4	59.4	66.7	62.5	66.7	62.5	69.9	69.9
12	56.6	58.9	61.7	58.8	65.3	61.6	63.9	68.1
13	58.8	61.9	64.9	61.4	69.9	65.3	67.5	73.3
14	59.3	62.5	65.5	61.9	70.6	66	68.1	74.1
15	58.8	61.5	64.4	61.2	68.8	64.6	66.9	72
16	58.2	60.5	63.5	60.5	67.1	63.4	65.7	70
17	58	60.1	63	60	67.1	63.4	65.7	70
18	58.2	60.3	63.2	60.3	66.6	63	65.4	69.2
19	58.4	60.5	63.4	60.5	66.8	63.2	65.6	69.5
Average	**57.5**	**62.23**	**64.65**	**60.5**	**72.1**	**66.5**	**67.7**	**74.5**

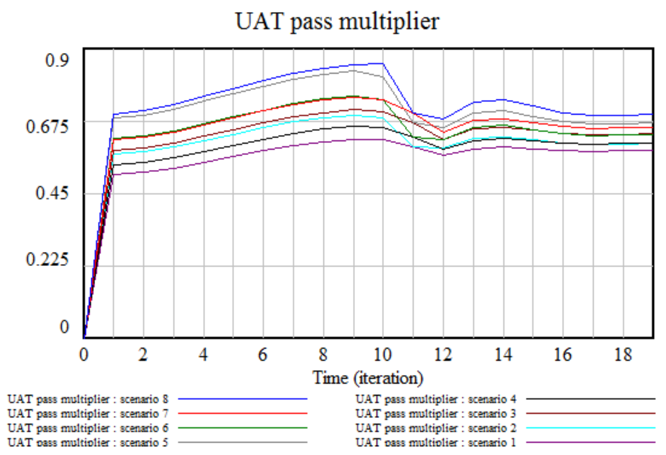


Fig. 7. Graph for User Acceptance Testing Rate for Various Scenarios

Table 6 shows the various scenarios of factors affecting the quality of user acceptance testing. The results for the *UAT pass multiplier* with various scenarios are presented in table 7 and Fig.7. Various PT adoption policies were simulated as well as determining the impact of management hiring options of the pertinent to the ability of the QA tester. These factors help improve the sad path test coverage and discover defects that are only usually discoverable by the system end user [34][36]. PT in the context of this paper is the practice of developers pairing with the QA alone or with the QA and onsite customer in writing and coding the test cases to run behavioral examples of system features authored by the customer[34].

As such, PT is not considered to have significant impact on the AAT pass ratio since the test examples written by the customers are unequivocally defined and does not necessarily need the exploratory testing skill input of a second tester/ developer. The impact of SP is clearly seen to reduce the pass ratio in some scenarios while it remains relatively inactive in some scenarios. The cognitive ability of the QA is noticeable to be most significant on the UAT pass rate in the project followed closely by PT. The adoption of PT and having a QA with high cognitive ability with commendable domain savviness yield 17% improvement in the UAT pass rate to a project with poor QA Ability without a pair tester. However, it remains to be known if the savings made by deploying a second tester and hiring a QA with immense domain savvy and cognitive ability are more than the cost of their introduction

D. Limitation of the Study

The calibration of the model was based on data from peer-reviewed literature, surveys and interviews. Bias of any of the sources could inhibit the validity of the model.

Furthermore, the sample size of the actual test cases per iteration produced by the team is relatively small. This being middle sized project, it may imply that this model is only applicable to middle-large sized projects with numerous test case developed due to high number of features; the model may yield different results for small projects.

Most importantly, these effectiveness of the various factors are valid under the conditions experienced by the project team, most notable the schedule pressure experienced. Intuitively, without the effects of schedule pressure process improvement practices, the adoption of these factors will yield better results.

VI. Conclusions and Future Work

This paper reports a developed SD model that acts as a decision making and process improvement pool to software development teams practicing CD. The goal of the model is to improve the effectiveness of the CD process and help managers optimize their development process. The impacts of practices such as PP, TDD, PT, customer involvement on the quality of the software were investigated. The authors also investigated the impact of the QA ability on the quality of software. The impact on SP experienced by teams is also substantilized in this study. Validating the model against data from a completed middle sized project, customer involvement proves to have the most significant impact on the quality of onsite AAT while the cognitive ability of the QA has the most impact on the quality of UAT.

The authors are addressing the limitations of this work and currently working on evaluating this model in an uninfluenced and "ideal" environment by simulating an exploratory project case study to fully evaluate the impact of various managerial policies on the CD process.

Furthermore, it is not enough to determine the qualitative impact of these various factors on the quality of the software project. This work points attention for possible concerns to address questions like: "what are the trade-offs of these practices and the optimal level of adoption of these practices on the CD performance metrics?"; what is the economic effectiveness of the adoption of the agile practices on the CD process?"; what is the extra resource requirement necessary to adopt these practices?"; "is the extra cost necessary to incorporate these practices better devoted to other value-adding tasks such as development or QA?"; "do the benefits(quality improvement) of the adoption of these practices overweigh their associated costs?"

References

[1] Beck, Kent; et al. (2001). *Manifesto for Agile Software Development*. Agile Alliance. Retrieved 14 Jan 2012.

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

[3] Lahtela.A and Jantti.M, *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] Fitz, T. *Continuous Deployment at IMVU: Doing the impossible fifty times a day*.2009. Published on 2nd October, 2009, Date accessed 2nd Jan, 2013. http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

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

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

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

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

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

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

[11] Borg, R. and Kropp, M. *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] Melis, M., Turnu, I., Cau, A., and Concas, G. *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] Humble, J., Read, C., and North, D. *The deployment production line*. Agile Conference, 2006, (2006), 6 pp. –118.

[18] Mantle, M.W. and Lichty, R. *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. R. and A. L. P. III, *Introduction to System Dynamics Modeling*. Pegasus Communications, 1981.

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

[22] Inst, C.M.U.S.E. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison Wesley, 1995

[23] K. Beck, *Test-driven development: by example*. Boston: Addison-Wesley, 2003.

[24] Y. Rafique and V. B. Misic, "The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 835–856, Jun. 2013.

[25] B. George, "An initial investigation of test driven development in industry," in *ACM Sympoium on Applied Computing*, 2003, pp. 1135–1139.

[26] R. C. Martin, *Agile software development: principles, patterns, and practices*. Upper Saddle River, N.J.: Prentice Hall, 2003.

[27] S. Yenduri and L. Perkins, "Impact of Using Test-Driven Development: A Case Study," Software Eng. Research and Practice,pp. 126-129, 2006.]

[28] Cornett, S., "Code Coverage Analysis," Bullseye Testing Technology http://www.bullseye.com/coverage.html , accessed 20th February 2014.

[29] Salesforce, *Force.com Apex Code Developer's Guide*, http://www.salesforce.com/us/developer/docs/apexcode/, date accessed 10th January 2013]

[30] L. W. H. Erdogmus, "The Economics of Software Development by Pair Programmers."

[31] Forrester, J.W, *Industrial Dynamics*. Cambridge, MA: The M.I.T.Press,1961

[32] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[33] [M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Boston: Addison-Wesley, 2012.

[34] J. Russell and R. Cohn, *Pair Testing*. Book on Demand, 2012.

[35] O. Akerele, M. Ramachandran, and M. Dixon, "System Dynamics Modeling of Agile Continuous Delivery Process," 2013, pp. 60–63.

[36] O. Akerele, M. Ramachandran, and M. Dixon, "Testing in the Cloud: Strategies, Risks and Benefits," in *Software Engineering Frameworks for the Cloud Computing Paradigm*, Z. Mahmood and S. Saeed, Eds. Springer London, 2013, pp. 165–185.