Investigating the Practical Impact of Agile Practices on the Quality of Software Projects in Continuous Delivery

Olumide Akerele(1), Muthu Ramachandran (2), and Mark Dixon(3)

(1) School of Computing, Leeds Metropolitan University (UK)
E-mail: o.akerele@leedsmet.ac.uk
(2) School of Computing, Leeds Metropolitan University (UK)
E-mail: m.ramachandran@leedsmet.ac.uk
(3) School of Computing, Leeds Metropolitan University (UK)
E-mail: m.dixon@leedsmet.ac.uk

ABSTRACT

Various factors affect the impact of agile factors on the continuous delivery of software projects. This is a major reason why projects perform differently- some failing and some succeeding- when they implement some agile practices in various environments. This is not helped by the fact that many projects work within limited budget while project plans also change-- making them to fall into some sort of pressure to meet deadline when they fall behind in their planned work. This study investigates the impact of pair programming, customer involvement, QA Ability, pair testing and test driven development in the pre-release and post -release quality of software projects using system dynamics within a schedule pressure blighted environment.

The model is validated using results from a completed medium-sized software. Statistical results suggest that the impact of PP is insignificant on the pre-release quality of the software while TDD and customer involvement both have significant effects on the pre-release quality of software. Results also showed that both PT and QA ability had a significant impact on the post-release quality of the software.

Keywords - Pair programming, Pair Testing, Test Driven Development, Continuous Delivery, System Dynamics

1- 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. It is therefore vital to ensure that a developed software ends up being deliverable- as that is the utmost goal. 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 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. The end result is imminent delivery failure.

Such 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 features behaves as required before being classified as *"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][3][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 due to pre release and post release feature failure [5][6].

Various interacting and interconnected factors are present in software projects and these are accountable for the inconsistencies in the quality of agile software projects [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 impact of these factors are further exacerbated by the fact that software project teams face the challenge of schedule pressure-- forcing them to make several shortcuts and relax on QA activities so as to meet up with the planned schedule [2][5][6][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 procedures [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 cost [10]. 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. This exemplifies the continuous active dynamics in a typical agile software project.

These behaviors as described in the above examples as well as their impact on the quality of the pre-release and post-release software quality within the CD process have not been studied in any context. This makes 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.

Furthermore, studying the impact of these factors in a "perfect" environment free of any budget or schedule pressure will yield less realistic results. Most software project at some point are faced with pressure to try and catch up when unexpected occurrences suffice in projects, therefore, impeding their productivity and making them fall behind schedule [2][5].It is therefore vital to study the behavior of these factors in such a realistic environment.

It is the goal of this research to develop a System Dynamics (SD) [8] model to study the dynamic effects of these investigated variables within the delivery lifecycle and their relative impact on the pre-release and post-release quality of software in a software project environment susceptible to schedule pressure. This work focuses on evaluating the effectiveness of critical agile factors, specifically*: Pair programming (PP), Test Driven Development (TDD), Pair Testing* (PT) and *QA Ability* within such a complex continuous delivery system.

SD stands out as the best approach for this study as it provides the leeway to alter the constituting variables within software projects and observing the behavior and evolvement of software projects over a period of time. Vensim [9], a SD software is used for this research work.

2- 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.

2-1 EMPIRICAL APPROACH

Poole and Huisman [12] conducted a case study at Iona technologies to evaluate the impact of XP adoption on the maintenances efficiency of a middleware product called "Orbix". The firm applied all the outlined XP techniques except the "40-hour week" to carry out maintenance on a legacy product and comparisons were made. Remarkably, they observed a 67% improvement in their iteration team bug- fixing productivity. However, the impact of each practice were not individualized.

Maurer and Martel [13] carried out a case study in a company specialized in web application product development - analyzing a completed development project involving nine programmers and lasted for 16months. Version one of the product was developed without any XP technique and lasted nine months while the second version was developed using all the XP technique lasted 5 months. Gains in productivity of 63%, 302% and 283% were realized in *NLOC/effort, number of methods/effort* and *number of classes/effort* respectively for the second version of the product. However, while the *bugs/effort* productivity ratio was reduced by 5%, the average code size of the second version was significantly more. Similar to Poole and Huisman's work [12], the individual impact of these practices were not identified.

Hodgetts and Philips [14] also carried a case study at an internet BRB company on two versions of a software product. Version one, utilizing 21 developers and lasting 20months and was developed without any agile methods while the second version utilized 4 developers and lasting 12 months was conducted with the 12 XP practices. The following results were recorded favoring the version 2 of the product: 80% reduction in development cost, albeit the result is clouded by the fact version two project consisted of more experienced developers; 70% reduction in defect density and the defects detected were also of less severity; 54% reduction in cyclomatic complexity of methods; 48% reduction in *average loc/method* ; 67% reduction in code size ;73% in *methods/size* which is an indication of improved decoupling and cohesion in the system.

Wood and Kleb [3] conducted a pilot project consisting of two release cycle of three iterations to evaluate the impact of agile practices at the NASA Langley Research Center by conducting a pilot project and adopting the 12 established XP practices. Analysis of the pilot project metrics showed that productivity was doubled when compared to similar completed projects developed without any of the practices. Better code design was also recorded due to improved readability and reduction in code size by approximately half the size.

Drobka et al [23] also conducted four pilot projects written in C++ by four different teams using 10 XP practices over an 18month period. A survey was sent to 29 of the participants in the project and their responses showed the participants felt XP boosted their work morale, reduced their project learning curve, improved their productivity and test coverage as well as quality and maintainability. Analyzing the quantitative data showed improvement in productivity ranging from 265% to 385% of waterfall projects.

These works lack the main focus on the impact of these factors on the behavioral quality of the software. In addition, none of these works considers the impact of the factors on the post-release quality of the software product.

Most importantly, none of the works mentions the telling impact of schedule pressure on the QA and agile practices and the consequential impact on the quality of the developed software.

2-2 SIMULATION APPROACH

Abdel-Ahmed [5] pioneered the application of SD for software process simulation modeling (SPSM) [6]. 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] investigates the efficacy of Test Driven Development (TDD) and PP using both SD and discrete event simulation- giving him the flexibility to introduce some randomness in the data input. The research was focused on the impact of these factors on the project effort and post-release defect density. These practices were investigated as standalone practices and the interactivity and dependence of these on other agile practices on the important risk variable outputs in agile projects are ignored.

Cao [21] studied the dynamics of agile software projects and the inter-relationship within the practices and variables. The author used relevant literature and interviews to develop and calibrate the full model. Validated both structurally and behaviorally, the model results suggest the cost of change in agile projects is actually cyclical due to the level of refactoring of the project and not flat as reported in literature [25][26]. The author also concluded that inefficient refactoring reduces team productivity and increases the cost of the project on the long run.

Kuppuswami et al [24] developed a SD model to investigate the effects of XP practices on project effort**.** An exploratory experiment was carried out on the model and results showed “On-site Customer" had the highest impact on the project effort while "refactoring" had the least impact. The author limited the scope of this paper to the impact of agile practices on solely project effort and ignored other crucial factors like quality. The author also makes some unrealistic assumptions in the model which questions the believability of the results. Furthermore, details of the parameterization of the model variables are vague. The authors went further to explore this model [25] in validating Beck’s hypothesis related to the constant cost of change in XP. Beck claimed “*when all XP practices are adopted are recommended, the cost of change in an XP project is nearly same irrespective of the time in which a request is introduced*”. [26]

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. Also, the quality of the software is not evaluated in the context of the system behavior.

Hitherto, to the best of the author’s knowledge, there hasn’t been any published work investigating the dynamics and impact of various factors, particularly agile practices, on the pre-release and post-release quality of an agile CD process.

3- 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 activated – 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 work is to develop a SD model to facilitate a repetitive and predictable delivery process of software while enabling users to have complete control over the delivery risk factors such as quality, 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 quality outcome of the CD process. In particular, full investigation of the direct and indirect effects of critical 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), User Acceptance Testing and Release to Production. The focus lies on the two testing stages within the delivery pipeline. The corresponding deliverables of each stage are represented in figure 1 below.

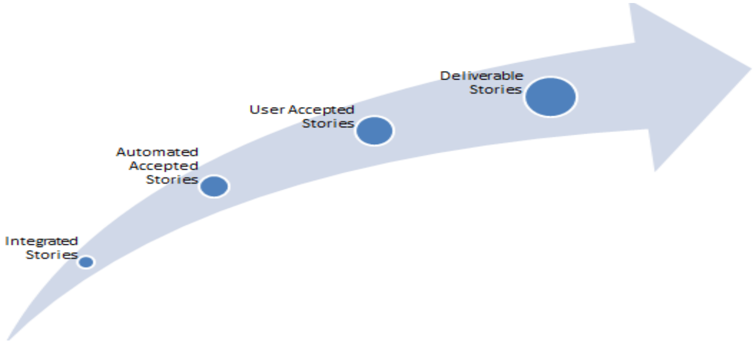

Figure 1 The delivery pipeline artifacts

3-1 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][27][60].

Numerous factors 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 controllability of quality of an agile CD project.

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 developed model will promotes achieving a fully controllable delivery environment and helps 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 planning an actualizing frequent release of their software to customers.

3-2 RESEARCH QUESTIONS

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

RQ1: What are the key factors impacting the success of CD? What are the agile practices having 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 schedule pressure on key factors in CD of agile software projects?

RQ4: What is the impact of agile practices, specifically, *on-site customer, TDD, PP and Pair Testing (PT)* on the pre-release and post - release quality of software. What is the impact of the ability of the Quality Assurance Analyst (QA) on the quality of the software?

3-3 RESEARCH OBJECTIVES

The following objectives were systematically 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.
- Carry out suitable hypothesis tests to determine the significance of the test results.

4- RESEARCH PLAN

This section describes how the objectives of the research were achieved.

4- 1 METHODOLOGY

Data Sources

- Interviews: Interviews with project managers, product owners, developers and testers were conducted in six companies.
- Literature review: Pertinent empirical findings from literature within the research topic domain were 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.
- Survey: An extensive survey was developed, sent and handed out to a host of relevant software practitioners. Surveys were particularly used when it was difficult to get hold of relevant data in the literature. This included a host of questions that are related to the present practices of the delivery process. In total, 392 full responses were analyzed for use in this study.
- Archived completed project data

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.

Simulation helps to overcome the shortcomings of empirical analysis: cost, flexibility and time consumption [5]. It provides the computerized prototype of an actual system run iteratively over time 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]. Figure 2 below shows some basic components of a SD model.

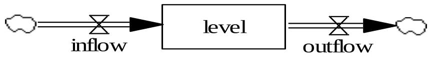

Figure 2 A simplistic SD model

Legend: "Level" is an Entity that increases or reduces over a specified period of time while the "Inflow"/"Outflow" represent rate of change in level.

Mathematically, the level/stock value above is derived by:

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

A SD model has a non-linear mathematical structure of first order differential equations expressed as:

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

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

5- SYSTEM DYNAMIC MODEL OF THE AGILE CONTINUOUS DELIVERY MODEL

The full CD model is designed into two sub-models for ease of analysis: *The schedule pressure sub-model* and the *delivery pipeline sub-model.* Due to space constraints of this paper, only the *schedule pressure sub-model* is discussed in this paper. The *schedule pressure sub-model* is selected as it is smaller in scope and the model variables can be easily zoomed upon and explained in totality--while meeting the space constraints of this paper. Discussing the full *delivery pipeline sub-model* will consume a lot more space and a section of it cannot be isolated and discussed separately due to the complex inter-relationship within all components of the model. Details of the full *delivery pipeline sub-model* are presented in [40]. The model variables are italicized for easier recognition.

THE SCHEDULE PRESSURE SUB-MODEL

Schedule pressure is defined as the ratio of the difference between the actual work to be done and the estimated work to be done to the estimated work to be done [5]. It has been recognized by many researchers as a major cause of project outcome inconsistencies -- making it's management crucial in software projects[2][5][6]. In our model, *schedule pressure* is calculated by the ratio of the *work gap* to the *estimated work left,* the *work gap* being the difference between the *actual work left* and the *estimated work left*. The formula for calculating *schedule pressure* is presented below.

$$schedule\ pressure = \frac{actual\ work\ left - estimated\ work\ left}{estimted\ work\ left} \quad (3)$$

The *estimated work left* is the amount of work that should be left in the iteration according to the release plan while the *actual work left* is the remaining work that is yet to be done in the project.

A negative value of *schedule pressure* indicates the team is very relaxed and well ahead of their work. This may be due to hyper-productivity, tasks reduction or overestimation of tasks [5][6][41].

Software features are broken down into chunks called *user stories* which are subsequently broken down to both technical and non-technical activities called *tasks*. Due to the level of granularity needed in this model, the project work is estimated in *tasks*. *User stories* can have tasks ranging from 1 to 8 , depending on the complexity and size of the user story[41]. However, the average and most common number of tasks in a user story is 4 [I3][I4][43]. Hence, in this model, a user story is estimated to contain 4 tasks.

Effort in software development has evolved through various units of measurements: man-hours [7], functional points [28], lines of code [29]. However, these metrics promote unscrupulous activities among team members [41][42]. Agile software projects adopt *story points* as the ideal metric of measurement of work output. *Story points* are a subjective approach to measuring effort in agile software projects where the scaling of one story is weighted relative to the other [41]. Ubiquitously, this is agreed to be the best representative of effort as it measures true value of work delivered [36][41]. Figure 3 below shows the *schedule pressure sub-model.*

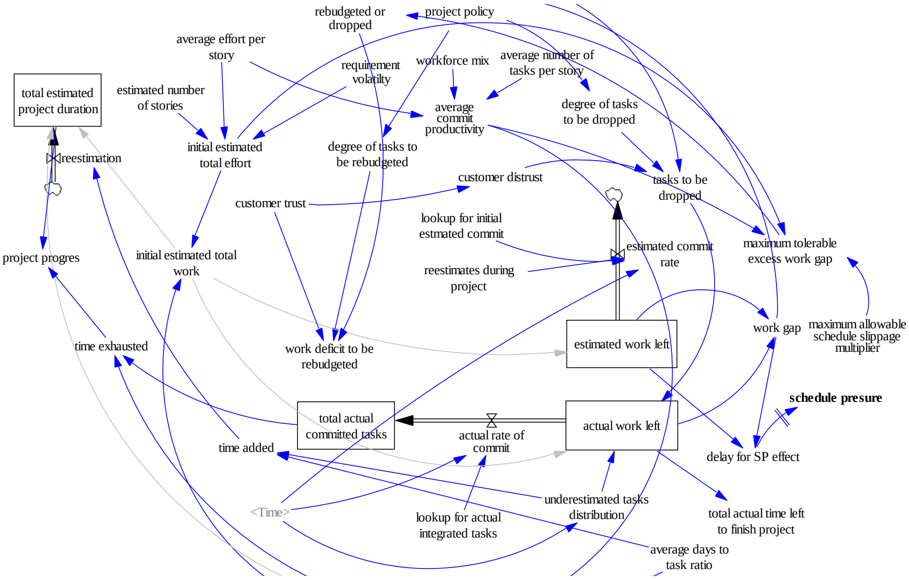
rebudgeted or dropped
project policy
average effort per story
requirement volatility
workforce mix
average number of tasks per story
total estimated project duration
estimated number of stories
average commit productivity
degree of tasks to be dropped
reestimation
initial estimated total effort
degree of tasks to be rebudgeted
tasks to be dropped
customer distrust
customer trust
lookup for initial estmated commit
project progres
initial estimated total work
estimated commit rate
maximum tolerable excess work gap
reestimates during project
work gap
maximum allowable schedule slippage multiplier
time exhausted
work deficit to be rebudgeted
estimated work left
schedule presure
total actual committed tasks
actual rate of commit
actual work left
time added
delay for SP effect
underestimated tasks distribution
<Time>
lookup for actual integrated tasks
total actual time left to finish project
average days to task ratio

Figure 3 The schedule pressure sub-model

The model contains numerous variables having direct and indirect impact on the two major schedule pressure level-determining variables: *Estimated work left* and *Actual work left.*

Initial estimated total effort is the high-level project estimation done during release planning. This estimated work is then implemented in accordance to the iteration plan based on the velocity of the team. The *initial estimated total effort* is the product of the *estimated number of stories*, *average effort per story* and the *requirement volatility*.

$$initial\ estimated\ total\ effort = estimated\ number\ of\ stories * average\ effort\ per\ story\ (1 + requirement\ volatilty) \qquad (4)$$

Requirement volatility is defined as the ratio of the modified and added requirements to the total requirements [30]. Changing requests in software projects are due to changes in customer's needs, stakeholder disagreements, changes in organizational goals and poor communication between customer and the development team [31]. Agile software projects begin with "fuzzy" and incompletely defined requirements and as the project evolves, the requirements become clearer and become more clearly defined, gradually fading out [31].

The modest case of the level of requirement volatility reported was in a waterfall project with an average of 6%, peaking at 16.5% [32]. Humphrey [33] suggested projects experienced 25% requirements volatility. Stark et al [30] examined 7 agile software projects and estimated the requirement volatility to be about 48%. Melnik and Maurer [34] also reported a requirement volatility of between 30-50% on a typical agile project. The "fuzzy" nature of the incompletely defined requirements as well as the continuous admittance of changing requirements by the teams is responsible for this relatively high phenomenon in agile software projects.

Productivity is defined as the ratio of the amount of software produced to the cost of producing it [32]. Mills redefines productivity as the ratio of the functional value created by a developed software to the cost of producing it [35]. In agile software development, user stories are the preferred method of quantifying the team's output since each story contains minimum of a software functionality which is useable by the customer [36]. Productivity is calculated in agile projects as the ratio of the user stories to the effort of producing it.

Since tasks are a breakdown of *user stories* and we need to express productivity in a lower level of granularity and also due to our assumption that all tasks developed in an iteration are "committed", the average development productivity equals the *average commit productivity.* The *average commit productivity,* hence, equals the ratio of *average number of tasks per story* "committed" to the *average effort per story.*

However, the *average commit productivity* is moderated by the *workforce mix*. Experienced team members are suggested to be more productive than inexperienced or new team members. Literature reports varying ratios of productivity between experienced and inexperienced programmers: 0.5 [5], 0.5 [6], 0.64[43], 0.45[44]. In our model, a ratio of 0.5 is assumed i.e experienced developers are twice as productive as their inexperienced colleagues.

$$average\ commit\ productivity = \ (average\ number\ of\ tasks\ per\ story/ average\ effort\ per\ story) * workforce\ mix \qquad (5)$$

The *initial estimated total work* is the sum total of the initial estimates in tasks. This variable is a product of the *average commit productivity* and the *initial estimated total effort*. A *pulse function* is used for *initial estimated total work* in the simulation to provide the actual value only at the first iteration and a value "0" for the rest of the iterations.

$$initial\ estimated\ total\ work = (initial\ estimated\ total\ effort * \ average\ commit\ productivity) * PULSE(\ 0{,}0) \qquad (6)$$

Estimated work left is the sum total of the number of *initial estimated work left* minus the *estimated commit rate*. The *estimated work left* is modelled as a *level*, gradually decreasing depending on the *estimated commit rate*. The *estimated commit rate* is the velocity of the team as estimated during the release planning. A *lookup function* is used to simulate the *estimated commit rate* per iteration for model validation purpose. *Reestimates during project* are the major amendments to the team velocity after the project commencement. As the team constantly re-evaluates their velocity using “yesterday's weather” [41], they make alterations to their anticipated velocity due to present or forecast change in circumstances.

The *actual work left* is the actual amount of work left in tasks for the project. It is modelled as a *level* with an initial value equivalent to *initial estimated total work.* The *actual work left* decreases iteratively in the model depending on the *actual rate of commit.* The *actual rate of commit* of the development team is the actual number of tasks committed per iteration and this is assumed to be the number of tasks completed in the iteration.

The *work gap* is the discrepancy between the *estimated work left* and the *actual work left*. It expresses the extent at which the team is lagging or ahead of the project plan. The *maximum excess tolerable work gap* is a function of the agreed *maximum allowable schedule slippage multiplier* with the customer since the agreed schedule slippage with the customer determines the excess amount of work that can be accommodated if the team lags behind. The *maximum excess tolerable work gap* is a multiple of the *maximum allowable schedule slippage multiplier* and *initial estimated total work.*

The *tasks to be re-budgeted or dropped* is the difference between the *work gap* and *maximum excess tolerable work gap.* It is designed with an *if-then-else* function such that it has a value of "0" when the *work gap* equals or less that the *maximum excess tolerable work gap* and retains a value of the excess if the *work gap* exceeds the *maximum excess tolerable work gap.* This retained value is then negotiated by the project manager with the customer for a decision based on the features project management policy.

Two main policies exist for managing features in agile projects: feature driven policy or iteration driven project [41]. A feature driven project means that the customer prioritizes the development of all the features at all costs without any compromise on the requested features. The customer is willing to extend the deadline date and perhaps increase the funding for all of the requested features to be completed. In this case, if the team is running behind schedule, the retained excess value is re-budgeted and the expected project completion date is extended with the customer. On the other hand, an iteration driven project provides no flexibility beyond the agreed project completion which determines the *maximum allowable excess work gap* based on the team's productivity*.* Some organizations may also adopt both policies: while they are flexible to extent the project delivery date, they are only willing to do so by a limited extent*.*

The *project policy* variable is designed to sum up to 100%. In the case of a feature driven project, the *project policy* variable is designed to send a value of 100% to the *degree of tasks to be re-budgeted* and sends 0% to the *degree of tasks to be dropped.* On the other hand in an iteration based project, a *project policy* value of 100% is conferred to the *degree of tasks to be dropped* and 0% is conferred on to the *degree of tasks to be rebudgeted.* If for example the organization is flexible with both policies equally, then a value of 50% each is assigned to variables *degree of tasks to be rebudgeted* and *degree of tasks to be dropped.*

The *work deficit to be rebudgeted* is moderated by the level of trust the customer has for the team [2][30][42]. Trust is the reliance of one party on another party that an expectation of a certain level will be met [45]. *Customer trust* is earned over time after the customer and the team have built a healthy working relationship. They experience a sequence of 4 activities which earns the respect and trust of both parties and consequently yielding synergy in the team: forming, storming, norming and performing [46]. *Customer trust* is a measure of the previous performance satisfaction by both parties (customer an project team) [47] *Customer trust* is needed for the customer to authorize and agree any rebudget request put forward by the team. As such, despite the customer adopting the feature-driven approach with sufficient funds to finance any re-budget, not all tasks identified to be re-budgeted are ratified by the customer. This behaviour is corroborated by one of our interviewees (I2):

"... .*it can be a real challenge when the customer doesn't believe more time and resources are needed to complete the project. They may reject to support any resizing, not because they don't need the features or can't afford it, they just don't trust the team with their estimations. This is more common with customers dealing with us for the first time.... as time goes on, their trust for the team builds...*"

Measuring customer *trust* is a challenge as there is no established or published scale for measuring it. In our model, it is designed to retain a subjective value assigned by the project manager, ranging from 0-100%. We intuitively assume a linear relationship between the level of *customer trust* and the *work deficit to be rebudgeted.*

$$\begin{aligned} &work\ deficit\ to\ be\ rebudgeted \\ &= tasks\ to\ be\ rebudgeted\ or\ droppedx\ degree\ of\ tasks\ to\ be\ rebudgeted\ x \\ &\quad customer\ trust \qquad (7) \end{aligned}$$

Likewise, *tasks to be dropped* are influenced by the management policy on the *degree of tasks to be dropped* and *customer distrust (1-customer trust).* All *tasks to be dropped* are removed from the *actual work left* in the project. This reduces the *work gap* and reduces the *schedule pressure* in the project.

Time added due to re-estimation is determined by the *work deficit to be re-budgeted* and the *average days to task ratio.* The *time added due to re-estimation* is added to the *total estimated project duration* to update the expected project delivery date. The *total estimated project duration* is modelled as a *level* which increases the expected delivery date every time there are tasks to be rebudgeted.

The *project progress* helps the team know how long they realistically have left to finish the project. It is formulated as a ratio of the *time exhausted* on the project and the *total estimated project duration.* The *time exhausted* in days is derived by multiplying the *total actual committed tasks* by the *average days to task ratio.* This value estimates the real-time progress of the team in percentage.

Data got from the actual number of tasks developed is compared with the estimated number of tasks committed. The difference reflects the discrepancy of the estimation and the actual turnout of the project, which is then distributed across the iterations in the project. This function is represented by the *underestimated tasks distribution* variable in the simulation model.

However, the rate of new requirements discovery isn't uniform during the project lifecycle: the initial stage of agile project is with little visibility and little knowledge about the full outcome of the software, hence, it is customary that more requirements are discovered at this stage [41][48]. The *underestimated tasks distribution*, designed as a *step function* [6], is used to build this behaviour into the model. It should be noted that this variable, *underestimated tasks distribution*, was solely used for the model validation purpose.

There is a time delay experienced before the team re-evaluates their actual progress and compare it to their estimated progress. After this time delay, the effect of schedule pressure takes effect and the teams start feeling the need to increase their productivity. In our model, *delay for schedule pressure effect* is the time delay variable needed for the effect of schedule pressure to take effect. The *delay for schedule pressure effect* in this model is given a value of 1 (iteration) since iteration planning occurs at the end of the iteration.

Table 1 summarizes the value and parameters of the major auxiliary variables used in the schedule pressure sub-model simulation for the project.

Table 1 Key variable parameters in the schedule pressure sub-model

Variables	**Values**
Requirement Volatility	40%
Initial Estimated Total Effort	882 points
Average Effort per Story	5 points/story
Average Commit Productivity	0.8 tasks /point
Workforce Mix	1
Estimated Commit Rate	40 tasks/iteration
Maximum Allowable Schedule Slippage Multiplier	7%
Customer Trust	100%
Degree of Tasks to be Rebudgeted	100%
Delay for schedule pressure effect	1

The project progress details were also used in the simulation of schedule pressure. The project details of each iteration used in the simulation are summarized in table 2 below.

Table 2 Project work data used for schedule pressure simulation

Iteration #	**Estimated Tasks Committed**	**Actual 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**

The pressure influences the adoption of major practices in the model and consequently the outcome of the project [5][6]. Figure 4 below shows the simulated graphical representation of the SP experienced by the team .

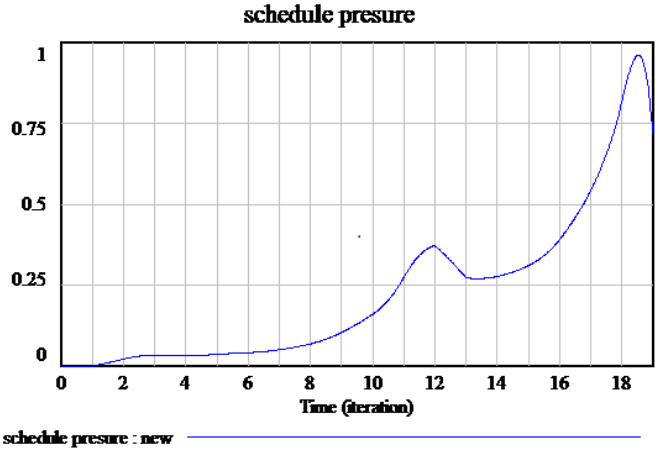

Figure 4 Project schedule pressure

The developed *schedule pressure sub-model* serves as reusable tool by project managers to help forecast the expected "cuts" in the commitment of the development team to various agile practices based on the underlying policies of the organization. This model consequently predicts the effect of such schedule pressure experienced by development teams has on the quality expectations of the developed software. To the best of our knowledge, hitherto, no similar model exists to specifically address the impact of schedule pressure on software projects, despite the severity of the problems caused by schedule pressure in software development.

6- 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. 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 XP 2013 (http://xp2013.org/) and Agile 2013 (http://agile2013.agilealliance.org/) conferences and valuable feedback was incorporated to rework the model.

Behavioral validation checks the model actually produces results that are similar to real life projects. The model has been validated against data of output variables from a completed software project with similar characteristics that successfully implemented CD. Details of the projects and simulation results are discussed in the next section of this paper.

6-1 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 table 3 below:

Table 3 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	BDD

Testing Framework	
Automated Acceptance Testing Tool	JBehave
Team Experience Mix	Average of 9years software projects experience
Working hours/day	7.5

6-2 SIMULATION RESULTS

Table 4 below summarizes the extracted results from the simulation model. "AA" denotes " automated acceptance" and "UA" denotes "user acceptance" in the table. Results for the automated acceptance test and user acceptance tests are presented graphically in figures 5 and 6.

Table 4 Results comparison of actual project outcome and simulated project outcome

Iteration	**Actual Passing AA test cases**	**Actual AA Pass Rate**	**Simulated 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

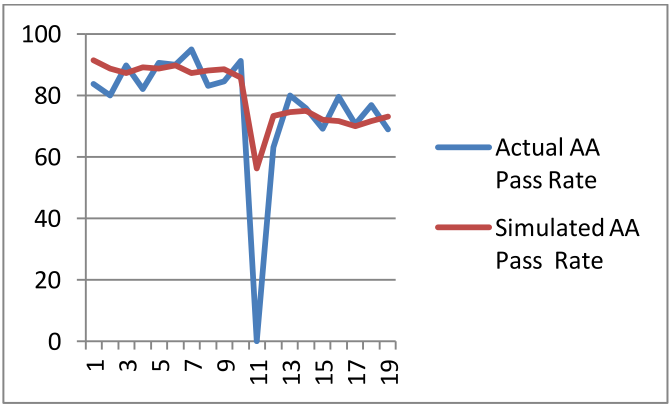

Figure 5 Graphical comparison of actual and simulated automated acceptance test (AAT) pass rate

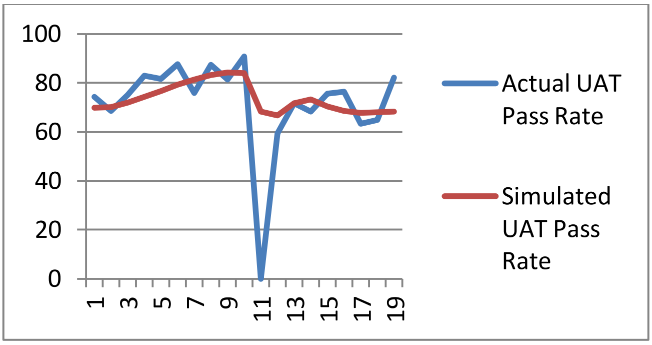

Figure 6 Graphical comparison of actual and simulated user acceptance test pass rate

The data provided in table 4 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 1st and second iterations, 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 data for that iteration. This results in the high impact of the % variation between the simulated and actual results for the first two iterations. It is plausible to believe that the actual pass ratios for these iterations with low test cases are exaggerated, hence the significant difference in the actual simulated results. In the 12th iteration, the team had significantly more actual failing tests due to the impact of major refactoring in the 11th iteration on the passing test suite. 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 at 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. This phenomenon may explain the considerable disparity in the passing test rates in the final iteration than that projected by the simulation model.

7- MODEL SENSITIVITY ANALYSIS FOR 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. The model experimentation is sub-divided into two sections, to analyze results of these pertinent factors on the pre-release and post-release software quality respectively.

7- 1 IMPACT OF TDD, PP AND CUSTOMER INVOLVEMNT ON PRE-RELEASE SOFTWARE QUALITY

Specifically, the impact of TDD, PP and Customer Involvement on the pre-release software quality is evaluated in this work. Table 5 presents the various management policies are explored to observe their impact on the onsite automated acceptance pass rate. The efficacy of each of these practices are investigated independently on the pre-release quality of the software. The final scenario, scenario 4, occurs when all the factors are incorporated in software projects. Table 6 presents the simulations results of the various scenarios.

Table 5 Scenarios simulated with model

	PP	**TDD**	**Customer involvement**
Scenario 1	100%	100%	0%
Scenario 2	100%	0%	100%
Scenario 3	0%	100%	100%
Scenario 4	100%	100%	100%

Table 6 below summarizes the simulations results of the various policies.

Table 6 Quality simulation results of various pre-release team policies

Iteration #	**Scenario #**			
	1	**2**	**3**	**4**
1	38.9	64.9	85.3	97.4
2	37.9	63.2	82.3	94.8

3	37.3	62.4	80.8	93.4
4	38.1	63.7	83.0	95.4
5	37.9	63.4	82.4	94.9
6	38.3	64.1	83.6	95.9
7	37.3	62.3	81.0	93.2
8	37.6	63.0	82.0	94.1
9	37.8	63.3	82.5	94.5
10	36.5	61.9	80.4	91.2
11	23.5	58.7	53.6	58.7
12	29.4	25.2	73.0	73.5
13	30.3	56.0	73.3	75.8
14	30.5	56.2	73.6	76.3
15	29.1	54.7	71.3	72.8
16	28.7	54.9	71.3	71.7
17	27.9	53.7	69.8	69.8
18	28.6	55.1	71.6	71.6
19	29.2	56.1	73.0	73
Average	33.4	58.0	76.5	83.6

Table 6 above shows the relative impact of applying various managerial policies on the pre-release quality of the software- with the pass rate values rounded off to the nearest 1 decimal place. Clearly, scenario 4 (adoption of all practices) provides the most beneficial results, with a mean of 83.6%. Scenario 1(without customer involvement) is observed to have the poorest results (-50.2%), suggesting it has the highest impact on the pre-release quality of the software. Scenario 3 (without PP) on the other hand was observed to have the lowest impact (-7.1%) on the AAT pass rate.

To determine if these improvements are significant, the authors conduct a series of significance tests on these results for the various scenarios. The following null and alternate hypothesis were developed for this purpose

Null Hypothesis, $\boldsymbol{H_o}$**:** When a software project adopts PP, TDD and customer involvement in an environment susceptible to schedule pressure at some point during the project, there is no significant difference in the pre-release software quality when compared with a project without PP or TDD or customer involvement.

Alternate Hypothesis, $\boldsymbol{H_\alpha}$: when a software project adopts PP, TDD and customer involvement in an environment susceptible to schedule pressure at some point during the project, there is a significant difference in the pre-release software quality when compared with a project without PP or TDD or customer involvement.

To test the hypothesis, it is vital to keep all other variables constant and only alter the variables of interest. Intentionally, the schedule pressure is not set to zero so as to reduce the possibility of any exaggeration in the results, leading to misleading favorable conclusions. As such, as experienced in the project case study, some time pressure is experienced by the team just before the middle of project completion till the project end. This helps to realize modest results leading to more realistic hypothesis test conclusions. More so, the schedule pressure is experienced by both the "control" and "experimental" groups. Scenario 4 -- being the base scenario in which all the practices are adopted -- is the "control group" while the other scenarios where the various practices are altered represent the "experimental group".

The hypothesis significance testing of data groups with normal distribution are best tested with standard parametric tests while data groups with dataset that don't follow normal distribution are best suited for non parametric tests [38]. In this study, the *independent t test* testing is adopted in the hypothesis testing of the normal distributed groups while the *Mann-Whitney U test* is used when carrying out hypothesis testing for two data groups not normally distributed.

Observing the data for all the groups, the effect of SP can be easily noticeable with some extreme values, suggesting the data groups are not normally distributed and will not form a single line in a *probability plot*. *Shapiro-Wilk* normality test is further carried out to determine if the dataset satisfies the conditions of normal distribution. The results are summarized in table 7 below.

Table 7 Shapiro-Wilk normality test results for pre-release pass rate simulation results

	μ	**α**	**Kurtosis**	**Calculated *W***	**Critical *W***	**p value**	**Ho (95% significance)**
Scenario 1	33.41	4.93	-1.513	0.8279	0.901	0.003	Rejected
Scenario 2	58.04	8.86	7.0	0.6069	0.901	0.000005	Rejected
Scenario 3	76.5	7.64	1.586	0.829	0.901	0.003070	Rejected
Scenario 4	83.58	12.36	-1.509	0.83	0.902	0.003186	Rejected

To reject the null hypothesis in Shapiro-Wilk normality test, *calculated W* must be less than *critical W.* In the four scenarios, *calculated W* is less than *critical W*, hence, the null hypothesis is rejected in the four scenarios. This implies that all the data groups are not normally distributed -- confirming their suitability for non-parametric tests. Table 8 summaries the results of the Mann-Whitney U test carried out on the significance testing of the various policies.

Table 8 Mann-Whitney U hypothesis test results for pre-release quality factors

	Mean of Ranks		**Calculated *U***	**Critical U (at 0.05)**	H_o
	Control Group	**Experimental Group**			
Case 1 (Scenario 4 vs. 1)	29	10	0	113	Rejected
Case 2 (Scenario 4 vs. 2)	28.5	10.6	10.5	113	Rejected
Case 3 (Scenario 4 vs. 3)	22.6	16.44	122	113	Cannot be Rejected

To reject the null hypothesis in *Mann-Whitney U* test, the value of *calculated U* must be less that the value of *critical U*. The first two hypothesis results of case 1 (without customer involvement) and case 2 (without TDD) produced results of *calculated U* < *critical U*, with values indicating a *highly significant* variation in the two medians. As such, we reject the null hypothesis for both cases, concluding that TDD and Customer Involvement are highly significant in the pre-release software quality of software projects susceptible to schedule pressure. On the other hand, the results of Case 3 (without PP) produced *calculated U* > *critical U.* Hence, there was insufficient evidence to reject the null hypothesis. We conclude that PP does not have a significant impact on the pre-release software quality of software projects prone to schedule pressure.

7-2 IMPACT OF PAIR TESTING AND QA ABILITY ON POST-RELEASE SOFTWARE QUALITY

The model was further explored to determine the efficacy of identified critical factors impacting the post-release software quality: *Pair Testing* (PT) and *QA Ability*. These factors are identified via extensive literature surveys and interviews. The *QA ability* is a function of cognitive ability as well as the domain savvy of the QA. Various authors have investigated and confirmed the criticality of QA cognitive ability [49][50][51] and QA domain knowledge [52][53][54][55][56] on the quality of developed test cases. Psychometric and IQ tests are ideal for measuring a QA's cognitive ability [57][58] while the experience QA is investigated to be highly correlated to the domain knowledge of the QA [59]. For simplicity, we have categorized the *QA Agility* as "high" and "low".

PT [39] in the context of this study is the practice of developers pairing with the QA alone or with both the QA and onsite customer in writing and coding the test cases to run behavioral examples of system features authored by the customer [34]. In essence, PT is not considered to have a direct 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. Table 9 shows the various managerial scenarios that can be applied in software projects using these factors.

Table 9 Scenarios for post-release model sub-section

	PT	**QA Ability**
Scenario 1	No	Low
Scenario 2	Yes	Low
Scenario 3	No	High
Scenario 4	Yes	High

The four scenarios in table 9 are simulated individually and their results on the *UAT pass rate multiplier* are presented in table 10 below.

Table 10 Simulation results for post release software quality

Iteration #	**Scenario #**			
	1	**2**	**3**	**4**
1	52.05	59.06	57.6	67.44
2	54.02	61.61	59.69	69.86
3	55.33	62.96	61.15	71.67
4	59.57	68.65	65.59	75.19

5	59.09	67.26	65.30	76.55
6	62.39	71.54	68.78	79.63
7	61.06	68.95	67.61	78.73
8	62.27	70.31	68.90	80.21
9	64.75	73.64	71.40	83.56
10	63.6	71.35	69.75	80.36
11	61.5	66.42	66.30	72.91
12	60.1	63.36	63.92	68.23
13	61.78	66.37	66.74	73.09
14	61.05	65.22	65.98	71.87
15	60.91	64.51	65.46	70.49
16	59.76	62.49	63.63	67.42
17	59.14	61.49	62.8	66.06
18	59.47	61.87	63.14	66.45
19	60.62	63.28	64.29	67.87
Average	**59.92**	**65.80**	**65.20**	**73.03**

Various PT and QA hiring options were simulated to observe their impact on post - release quality of the software. These factors help improve the sad path test coverage and discover defects that are usually discoverable by the system end user [39].

Scenario 2 results shows suggests n average of 7.8% improvement in the UAT pass rate when PT is adopted compared to when PT is not adopted in the project. Scenario 3 results show that "High" *QA Ability* is seen to have an average of 8.4% improvement on the UAT pass rate compared to “low" QA Ability. The impact of SP is clearly seen to reduce the UAT pass ratio in some scenarios while it remains relatively inactive in others.

Hypothesis significance tests are carried out on these results to test the significance of these findings using the null and alternate hypothesis described below:

Null Hypothesis, $\boldsymbol{H_o}$**:** There is no significant difference in the post release quality of software when PT and a QA with high ability is used compared to when PT and/or QA with high ability are not adopted in an environment prone to schedule pressure.

Alternate Hypothesis, $\boldsymbol{H_\alpha}$: There is significant difference in the post release quality of software when PT and a QA with high ability is used compared to when PT and/or QA with high ability are not adopted in an environment prone to schedule pressure.

It is essential to determine the nature of distribution of the data in each scenario to identify the apt significance testing technique to be used. Probability plot is drawn for each scenario to determine of the results of the scenario are *normally distributed*. Figures 7, 8, 9 and 10 shows the probability plots of scenarios 1,2,3 and 4 respectively.

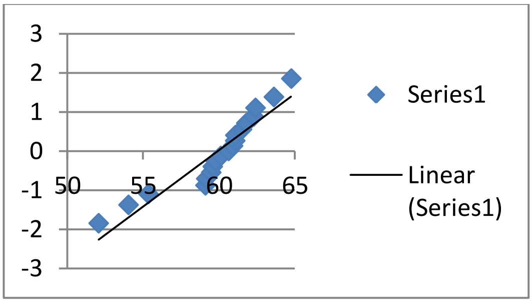

Figure 7 Probability plot for scenario 1

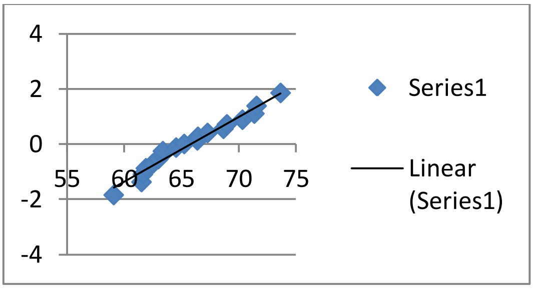

Figure 8 Probability plot for scenario 2

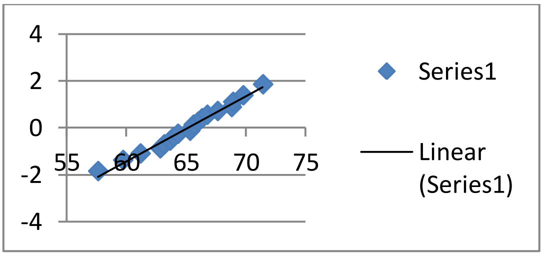

Figure 9 Probability plot for scenario 3

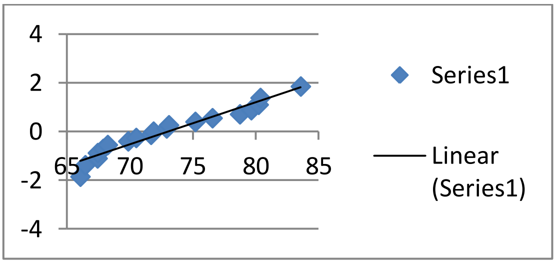

Figure 10 Probability plot for scenario 4

For a data group of normal distribution, the values of the data in the group cluster on or very closely to the trend line; otherwise, the data group cannot be attributed as normally distributed. Figures 8,9,10 are normally distributed as they fall on the same line. However, figure 7 shows substantial number of distant points from the trend line even though a number of the plots fall on the line - suggesting the data group isn't normally distributed. To verify that scenario 1 data is not normally distributed, we further perform a Shapiro-Wilks normality test. Results of the test are detailed in table 11 below.

Table 11 Shapiro- Wilks normality test result for scenario 1

	Kurtosis	**Calculated *W***	**Critical *W***	**p value**	**Ho (95% significance)**
Scenario 1	-1.288	0.8918	0.901	0.003	Rejected

As the *calculated W* (0.89173) is less than the *critical W* (0.901), we reject the null hypothesis at 95% confidence interval. Thus, we confirm data for scenario 1 is not normally distributed.

Unpaired t tests are adopted for hypothesis testing of case 2 (low QA Ability) and case 3 (no PT). In this case, Scenario 4 is recognised as the "control group" while scenarios 1, 2, 3 are used as the "experimental groups" for the hypothesis testing. Statistical *F test* is conducted on cases 2 and 3 to test for variance equality to identify the appropriate unpaired *t test* formula to be used. The null hypothesis in both case was could not be rejected at 95% confidence interval, concluding there is no significant difference in the variance of each data group. Table 12 below summarizes the results of the *unpaired t tests* carried out on the various scenarios.

Table 12 Hypothesis test results for cases 2 and case 3

	µ		**α**		**P value**	H_o
	Control Group	**Experimental Group**	**Control Group**	**Experimental Group**		
Case 2 (Scenario 4 vs. 2)	**73.03**	**65.80**	**5.16**	**4.06**	**P < 0.0001**	**Rejected**
Case 3 (Scenario 4 vs. 3)		**65.16**		**3.46**	**P<0.00001**	**Rejected**

With both p values for both cases less that 0.0001 at p> 0.05, the null hypothesis for both cases are rejected results, confirming that PT and QA Ability are highly significant in the post release quality of software in an environment prone to schedule pressure.

The hypothesis test for case 1(low QA Ability and no PT) was tested with the *Mann-Whitney U* test as both data groups are not normally distributed. The results are tabularized in table 13 below.

Table 13 Mann-Whitney U test results

Sum of Ranks		**Mean of Ranks µ**		**Calculated U**	**Critical U (at 0.05)**	H_o
Control Group	**Experimental Group**	**Control Group**	**Experimental Group**			
551	**190**	**29**	**10**	**0**	**113**	**Rejected**

The results above is highly significant as the *critical U* far exceeds the *calculated U*, hence, the rejection of the null hypothesis. We can therefore confirm that there is significant difference in the post release quality of project when PT and QA with high Ability are adopted in an environment prone to schedule pressure.

7-3 LIMITATIONS OF THE STUDY

External Validity

Similar to all experiments executed using SD, the model results validity are based on the calibration and parameterization of the model variables. When data for some variable are scarce and not available in literature, some of the parameters used in the variables are based on expert judgment and discretionary assumption, which may be prone to being biased. Moreover, variable parameters -particularly those related to the cost of implementing rework across the stages in the delivery pipeline- may be subject to debate across various environments and ethics where the QA resources are somewhat allocated differently. However, in such cases when the values of variables are subjective, effort is made to validate this value by conducting surveys to estimate the values the values generally agreeable by experts in the field. One of the mean, median and mode of these surveys were used to adopt the most realistic values in the industry so as to reduce the impact of using exaggerated or deprecated values.

Most importantly, the 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.

Internal Validity

Internal validity concerns were mitigated by testing the normality of the data groups, testing for equality in variance of data groups using *f test*, and ensuring the right hypothesis testing technique is used.

The number 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 not applicable to middle-large sized projects with numerous test cases developed due to high number of features. The only way to resolve this will be to validate the model using data from large projects practicing continuous delivery and using agile methods that evolve over a very long period of time which consist of high number of automate acceptance test cases and user acceptance test cases. In reality, such projects are hard to come by to effectively validate our work.

8- CONCLUSION AND FURTHER WORK

This study investigates the active factors in CD, the dynamics of these factors and focally investigates the impact of identified agile factors: PP, PT, customer involvement, TDD and QA Ability on the pre-release and post-release quality of software projects in an environment susceptible to schedule pressure using system dynamics. A system dynamics model was developed and validated against the quality metrics of a completed software project. The validated model was then explored to investigate the formulated hypotheses in this work.

The study shows that factors such as level of experience of developers, domain experience and cognitive ability of the QA, knowledgebility of the customer, among other factors have an impact on the quality of software and could cause variations in the performance of CD.

The model simulation results show that schedule pressure has a major impact on the level of adoption of key agile practices and consequently on the quality of software. Statistical analysis of the results shows that PP doesn't have a *significant* impact on the pre-release quality of software while TDD and customer involvement both have significant statistical impact on the pre-release quality of software. Results also showed that both PT and QA ability have significant impacts on the post-release quality of the software.

The developed SD model provides project managers with a tool to observe and anticipate the cause and effects of their actions within CD, enabling them have better controllability of the CD process. This facilitates achieving a repetitive, predictable and risk-free CD activity for software projects. Furthermore, the model will help management make realistic decisions based on the quality of impact of the agile practice invested.

There is ongoing effort to source another project validation data from a completed CD project that meets the validation requirements for the project. This is to re-validate the model and test its robustness and suitability for application in varying environmental configuration settings. Accomplishing this will further boost the generalization of the findings of this paper.

In addition, further work is to be conducted to investigate the trade-offs between the cost and improved quality achieved by adoption of these agile practices, the optimal level of adoption of these practices as well as the economic effectiveness of the adoption of the agile practices in the CD process. Ongoing study is being carried out by the authors to investigate these concerns and enlighten management on the financial efficacy of the adoption of these practices.

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] W. A. Wood and W. L. Kleb, “Exploring XP for scientific research,” IEEE Software, vol. 20, no. 3, pp. 30–36, May 2003.

[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), pp. 15–21, 2011.

[12] C. Poole and J. W. Huisman, “Using extreme programming in a maintenance environment,” IEEE Software, vol. 18, no. 6, pp. 42–50, Nov. 2001.

[13] F. Maurer and S. Martel, “Extreme programming. Rapid development for Web-based applications,” IEEE Internet Computing, vol. 6, no. 1, pp. 86–90, Jan. 2002.

[14] D. P. P Hodgetts, “Extreme adoption experiences of a B2B start-up”

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

[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, pp. 345–360, 2006.

[17] J.Humble, C. Read, and D.North, "The deployment production line," Agile Conference, 2006.

[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, pp. 5:1–5:26, 2010.

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

[23] J. Drobka, D. Noftz, and R. Raghu, “Piloting XP on four mission-critical projects,” IEEE Software, vol. 21, no. 6, pp. 70–75, Nov. 2004.

[24] S. Kuppuswami, K. Vivekanandan, P. Ramaswamy, and P. Rodrigues, “The Effects of Individual XP Practices on Software Development Effort,” SIGSOFT Softw. Eng. Notes, vol. 28, no. 6, pp. 6–6, Nov. 2003.

[25] S. Kuppuswami, K. Vivekanandan, and P. Rodrigues, “A System Dynamics Simulation Model to Find the Effects of XP on Cost of Change Curve,” in Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering, Berlin, Heidelberg, 2003, pp. 54–62.

[26] K. Beck, Extreme programming eXplained: embrace change. Reading, MA: Addison-Wesley, 2000.

[27] 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.

[28] A. J. Albrecht,”Measuring application development productivity", Guide/Share Application Develop. Symp. Proc., pp.83 -92 1979

[29] L. M. Laird and M. C. Brennan, Software Measurement and Estimation: A Practical Approach. Hoboken, N.J: Wiley-Blackwell, 2006.

[30] G. E. Stark, P. Oman, A. Skillicorn, and A. Ameele, “An examination of the effects of requirements changes on software maintenance releases,” J. Softw. Maint: Res. Pract., vol. 11, no. 5, pp. 293–309, Sep. 1999.

[31] N. Nurmuliani, D. Zowghi, and S. Powell, “Analysis of requirements volatility during software development life cycle,” in Software Engineering Conference, 2004. Proceedings. 2004 Australian, pp. 28–37, 2004.

[32] C. Jones, Programming Productivity, 1ST edition. New York: Mcgraw-Hill College, 1986.

[33] W. S. Humphrey, A Discipline for Software Engineering, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[34] G. Melnik and F.Maurer (2007), agile 2007 K. Beck, Test-driven development: by example. Boston: Addison-Wesley, 2003.

[35] H. D. Mills, Software Productivity. New York, N.Y: Dorset House, 1988.

[36] M. Cohn, User Stories Applied: For Agile Software Development, 1 edition. Boston: Addison-Wesley Professional, 2004.

[37] E. L. Lehmann and J. P. Romano, Testing statistical hypotheses. New York: Springer, 2005.

[38] R. G. Lomax and D. L. Hahs-Vaughn, An introduction to statistical concepts. New York: Routledge, 2012.

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

[40] Akerele, O., Ramachandran, M., & Dixon, M., "Evaluating the Impact of Critical Factors in Agile Continuous Delivery Process: A System Dynamics Approach," International Journal, 2014.

[41] M. Cohn, Agile Estimating and Planning, 1 edition. Upper Saddle River, NJ: Prentice Hall, 2005.

[42] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, 1 edition. Upper Saddle River, NJ: Addison Wesley, 2012.

[43] I. Benbasat and I. Vessey, "Programmer and Analyst Time/Cost Estimation," MIS Q., vol. 4, no. 2, pp. 31–43, Jun. 1980.

[44] Weiss D.M, "Evaluating Software Development by Error Analysis", Journal f Systems and Software, Vol.1, pp. 57-70, 1979.

[45] P. Hart and C. Saunders, "Power and Trust: Critical Factors in the Adoption and Use of Electronic Data Interchange," Organization Science, vol. 8, no. 1, pp. 23–42, Feb. 1997.

[46] D. B. Egolf, Forming Storming Norming Performing: Successful Communications in Groups and Teams. San Jose: Writers Club Press, 2001.

[47] K. Siau and Z. Shen, "Building Customer Trust in Mobile Commerce," Commun. ACM, vol. 46, no. 4, pp. 91–94, Apr. 2003.

[48] N. Nurmuliani, D. Zowghi, and S. Powell, "Analysis of requirements volatility during software development life cycle," in Software Engineering Conference, 2004. Proceedings. 2004 Australian, pp. 28–37, 2004.

[49] B. A. Uzundag, "Confirmation Bias in Software Development and Testing: An Analysis of the Effects of Company Size, Experience and Reasoning Skills." [Online]. Available: http://www.academia.edu/301912/Confirmation_Bias_in_Software_Development_and_Testing_An_Analysis_of_the_Effects_of_Company_Size_Experience_and_Reasoning_Skills. [Accessed: 22-Jun-2014].

[50] A. D. Da Cunha and D. Greathead, "Does Personality Matter?: An Analysis of Code-review Ability," Comm. ACM, vol. 50, no. 5, pp. 109–112, May 2007.

[51] L. Shoaib, A. Nadeem, and A. Akbar, “An empirical evaluation of the influence of human personality on exploratory software testing,” in Multitopic Conference, 2009. INMIC 2009. IEEE 13th International, pp. 1–6, 2009.

[52] J. Zander-Nowicka and F. FOKUS, Model-based Testing of Real-Time Embedded Systems in the Automotive Domain. Stuttgart: Fraunhofer IRB Verlag, 2009.

[53] L. J. White and E. I. Cohen, “A Domain Strategy for Computer Program Testing,” IEEE Transactions on Software Engineering, vol. SE-6, no. 3, pp. 247–257, May 1980.

[54] G. J. Myers, C. Sandler, and T. Badgett, The Art of Software Testing, 3 edition. Hoboken, N.J: Wiley, 2011.

[55] J. Itkonen, M. V. Mantyla, and C. Lassenius, “The Role of the Tester’s Knowledge in Exploratory Software Testing,” IEEE Transactions on Software Engineering, vol. 39, no. 5, pp. 707–724, May 2013.

[56] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander, “A Study on Agility and Testing Processes in Software Organizations,” in Proceedings of the 19th International Symposium on Software Testing and Analysis, New York, NY, USA, 2010, pp. 231–240.

[57] M. J. Ree and J. A. Earles, “Intelligence Is the Best Predictor of Job Performance,” Current Directions in Psychological Science, vol. 1, no. 3, pp. 86–89, Jun. 1992.

[58] M. W. Matlin, Cognition, 8th Edition. Wiley Global Education, 2012.

[59] R. R. Armin Beer, “The Role of Experience in Software Testing Practice,” pp. 258–265, 2008.

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