International Journal of Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-6, Issue-2)
Research Article
February 2017

Applying System Dynamics to Software Quality Management

Vimla Devi Ramdoo*
Dept. of IT, Charles Telfair Institute,
Moka, Mauritius

Oomesh Gukhool
Dept. of Computer Science & Engg.,
Réduit, Mauritius

Abstract—

Despite the significant evolution within the software industry, software quality remains a pertinent problem to many organisations today. This is because interactions between software project parameters in software development environment are complex and dynamic. Both management and developers have problems monitoring the fluctuations in quality that occur within the development process. This paper provides an insight on quality fluctuations that occur in a software project, using a quality index. The research focuses not only on changes on software project parameters that occur at the end but also throughout the software development process through dynamic modelling. The findings proved the ratio of 75-25 experienced to inexperienced staffs percentage to be the most practical combination. Likewise, the right balance of schedule pressure is determined and recommended so that it contributes to increasing the productivity during software development.

Keywords— Software Quality, System Dynamics, Modelling, Software Projects, Quality Index

I. INTRODUCTION

Software is among the most widely used product in human history; and it also has one of the highest failure rates due to the lack of quality in the end-products. Literature shows that software projects frequently suffered from quality-related problems in recent past years (Austin, 2001; Charette, 2005; Devedzic, 2001; Land *et al.*, 2005; Lindstrom and Jeffries, 2004; Meso and Jain, 2006; Molokken-Ostvold and Jorgensen, 2005; Rose, 2005; Wiegers, 2009). It is therefore of crucial importance to understand the element of quality when developing software. Software quality can no longer be considered insignificant or left unmeasured since there is too much financial and essential business value at stake.

The aim of this paper is to formulate a dynamic model that will improve software quality management. For this purpose, surveys will be carried out in software development companies, and causal loop diagrams for quality models will be developed. Subsequently, a dynamic model will be formulated from the cause and effect diagrams using System Dynamics. The results of the model shall be calibrated, verified and validated with original data from 3 web-based software projects. Furthermore, the margin of error will be calculated to show the precision of the simulated output.

II. DYNAMISM OF SOFTWARE PROJECT PARAMETERS

Software project parameters are interrelated and form a complex and dynamic relationship. They basically consist of the software project's cost (resources), scope and schedule that influence quality, also known as the Triple constraint. Managing the triple constraint requires making trade-offs between scope, schedule and cost. For example, the scope of the project might be reduced in order to meet schedule and cost goals. Alternatively the project's budget might increase to meet the scope and schedule goals. In 2005, Rose stated that "a project manager should never ever trade-off quality during project implementation" as quality is key element in satisfying the customer.

In 2010, Kathy explicitly mentioned that quality-related activities should be included into technical and resource planning since it is very critical and complex to control and monitor quality. It should however be noted that it is difficult to keep the triple constraints triangle balanced. Managers have many responsibilities and decisions regarding trade-offs during software development. These decisions impact the software project parameters than in turn influences software quality.

An imminent challenge in software project management is the lack of reference methodology to measure the fluctuations of quality, which is the Quality Index within the software development process. With technological advancement in software development, such a measure will become a fundamental requirement to promote further innovation by allowing people to objectively measure and compare the quality of similar software projects.

III. EXITING QUALITY FRAMEWORKS

The commonly mentioned quality standards in literature are ITIL, CobiT, CMMI, and ISO 9001 (Cater-Steel *et al.*, 2006; Gerke and Ridley, 2006; Vidal, 1998). Following Crosby's work on TQM, maturity models have become popular and include quality management. CMMI, CobiT and ITIL acquire the process maturity framework. Increasing amount of information about ITIL and CobiT have increased awareness and adoption of ITIL and CobiT (Casson, 2005; Deloitte, 2003; Hochstein *et al.*, 2005; Potgeiter *et al., 2005*). The SEI provides CMMI reports and advice on its web site. ISO 9001 appears in journals focusing on quality and magazines. Recent studies show that 60% of all corporate Six Sigma schemes fail to yield the expected outcomes (Chakravorty, 2010). Other frameworks gaining recent awareness are Balanced Scorecard, ISO 17799 (IT security techniques), PMBOK and Prince 2 (Cater-Steel *et al.*, 2006).

Nevertheless, the high cost factor remains a major consideration when integrating frameworks (Cater-Steel *et al.*, 2006; Paulk, 2004). Their concepts can however be adopted in this research, examples are the configuration management from ITIL, managing human resources and quality from CobiT, causal analysis, quality assurance from CMMI and quality audits from ISO 9001, and training from CMMI (Paulk, 2004)).

IV. THE SYSTEM DYNAMICS APPROACH

System dynamics (SD) modelling has been used since the 90's until recently and has proven to be beneficial in many fields. Formulated by Forrester in 1960, SD is a methodology used to analyse complex behaviours of systems and problems via computer simulation software. In 1984, Abdel-Hamid developed a simple model of the software development structure to connect the different dynamic relationships between processes. In 1996, Rodrigues and Bowers developed a SD model of the HRM cycle to analyse the effects of productivity, the number of staff working, and the work rate on project duration. Following the model of Abdel-Hamid and Madnick in 1989 and 1991, Lyneis and Ford (2007) provided a more widespread review of the literature on the applications of SD to software project management since the behaviour of parameters in the software development environment is too dynamic and complex in nature.

System dynamics make use of causal loop diagrams (CLDs) as well as stock and flow modelling technique. CLDs are built with interconnected causal links forming balancing and reinforcing loops, but they have their limitations for instance their inability to capture stock and flow. Stock variables represent the state variables that are the accumulations in a system. They generate information that decisions and actions are based upon. Flow variables modify the stocks by filling or draining the latter. Stock and flow models can be used to experiment alternative scenarios by simulating different variables values in the model. Simulated outputs are produced, in graphical or tabular forms, and are comparable with the real world (Sterman, 2003).

V. PROPOSED METHODOLOGY

To validate and assemble the critical software project parameters that cause a direct or indirect impact on software quality, a survey questionnaire was designed which covered various areas of software quality such as responsibility of quality, parameters influencing good software quality, software testing, and propositions of quality strategies. The questionnaires were circulated in software development organisations found in Mauritius, France and Denmark, that represented a sample size of 50 people in all. The targeted people work in software organisations within the private sector and are in touch with software quality during their day-to-day activities. SPSS statistical tool was used for analysis of data. This included non-parametric tests, regression analysis and correlation analysis. These tests and analysis were used to construct the cause and effect relationships (CLDs) of the software project parameters with respect to process quality, using VENSIM. To formulate the dynamic model, the CLDs have been converted into stock and flow diagrams and their respective formulae have been derived. The aim of the model is to determine the Quality Index and help to monitor process quality for a software development project.

Scenarios were used to test the model's validity. Managerial post mortem project reports were used to gather information on the individual scenarios. The reports provided highlights on the main elements of the respective software projects. Informal interviews were carried out to gather missing details of the post mortem project reports. A total of three scenarios are gathered from a single software organisation which are web-based software projects named A, B and C as described in Table I. Using the calibrated stock and flow diagram, various recommendations have been formulated with respect to software process quality.

Table I Software Project Scenario

A	E-Commerce web-based project
B	Content Management System (CMS) web-based projects
C	Search Engine Optimisation (SEO) and Software as a Service (SaaS) web-based project

This research consists of dynamic modelling based on dynamic hypotheses formulated on available knowledge from the literature review, working experience and information gathering techniques. The hypotheses will be validated by the model and then onwards recommendations shall be made.

A. Proposed Dynamic Hypothesis

Two main dynamic hypotheses were formulated based on literature, survey and working experience. They include the impact of staff experience dilution on quality and the impact of schedule pressure on quality.

1) Hypothesis H_1: Staff Experience

New staffs that lack skills, expertise and experience are less productive and more error prone thereby adversely impacting the quality level. On the other hand, experience and skilled staffs positively impact the quality level. When new staffs are involved in a team, they tend to produce more errors leading to low quality software. This is because ultimately errors will lead rework and delay the project completion. It is worth noting that software deliverables are highly interdependent. When the prior deliverables are of high quality, the current deliverables will also maintain its quality level. Alternatively, if the prior deliverables themselves contain errors, their negative impact on quality will be cascading as errors will remain embedded in the software throughout the development phases.

2) Hypothesis H_2: Schedule Pressure

When work remaining exceeds the scheduled time available, schedule pressure arises (Rai and Mahanty, 2001). Staffs work faster and longer hours in order to boost performance to meet the goals. When there is a rush and unrealistic goals, people tend to take shortcuts such as eliminating functionalities and sacrificing quality. These have an adverse effect that leads to rework, hence more work remaining thereby creating a vicious circle.

VI. ANALYSIS OF SURVEY RESULTS

The outcomes of the survey on the assessment of software project parameters impacting quality have been analysed followed by the identification of parameters, relationship and correlation among parameters using the Ishikawa concepts (1985) and causal loop diagrams.

Section A of the questionnaire was on the areas of software engineering, and the majority of respondents (60%) worked on web-based projects as developers. Section B of the questionnaire was about the responsibility of people towards software quality within the organisation. One of the analysed results is shown in Fig. 1, which disclosed that most of the respondents (70%) think that software developers and testers are responsible in ensuring good quality software.

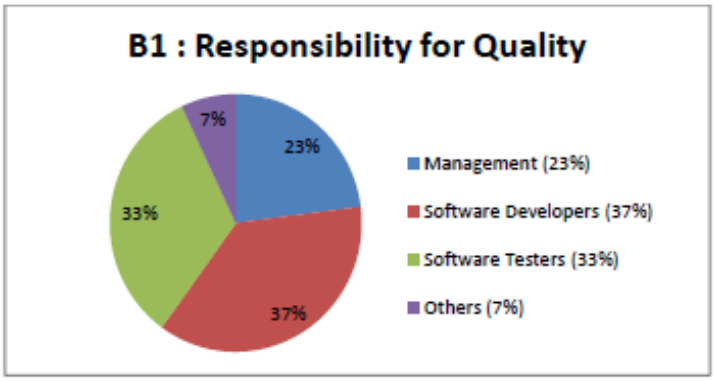

Fig. 1. Responsibility towards software quality and software testing

Section C of the questionnaire were rated on a Likert scale of 1 (strongly disagree) and 5 (strongly agree). The first 10 core software project parameters are shown in order of importance with respect to quality in Table 2 by the use of statistical mean of the results obtained.

Table 2 Software Project Parameters

	Software Quality Factors	Mean Result
1	Experience of team members	**4.16**
2	Well-defined requirement specifications	**3.94**
3	Software quality assurance activities	**4.38**
4	Maturity of the development process	**4.22**
5	Customer satisfaction	**4.16**
6	Good quality of prior work deliverables	**3.94**
7	The project is within schedule and there is no scope creep	**3.90**
8	The project is within budget	**3.88**
9	Error density of the software	**4.88**
10	Complexity of the project	**3.22**

To build the causal loop diagram, causal relationships were formulated through the analysis of the software project parameters impacting software quality. Descriptive statistics and T-test were used to analyse the responses obtained with mean and standard deviations. Partial results are summarised below. μ represents the mean value while σ represents the standard deviation obtained.

A. Scope and Quality

85% of the respondents agreed on the fact that scope creep ($\mu = 3.94$ and $\sigma = 0.82$) impacts the overall quality of software developed as they introduce more errors. Many suggested that investment into software verification ($\mu = 4.28$ and $\sigma = 0.83$) and validation ($\mu = 4.38$ and $\sigma = 0.75$) activities will improve quality.

B. Resource and Quality

90% of the respondents support the fact that new staffs are more error-prone ($\mu = 4.22$ and $\sigma = 0.76$). They also mostly agreed that the more the amount of errors in the software, the lower the quality of software produced ($\mu = 3.88$ and $\sigma = 0.96$). It was observed that the impact of fatigue was not significant on quality ($\mu = 2.88$ and $\sigma = 1.14$).

C. Schedule and Quality

The survey disclosed that most respondents believed that schedule pressure have an impact on both productivity ($\mu = 3.90$ and $\sigma = 0.84$) and quality ($\mu = 4.16$ and $\sigma = 0.82$). When there are errors in the prior developed tasks, they are propagated and create more errors in the software, and therefore lead to the decline in quality. 96% of respondents agreed that the quality of prior deliverables ($\mu = 3.74$ and $\sigma = 0.78$) has an impact on the quality of current work being done. Schedule slippage has $\mu = 4.16$ and $\sigma = 0.82$ implying that most of the respondents agreed it may cause an increase in the time remaining for development and testing activities.

It should however be noted that the list of software project parameters is not exhaustive. There are more factors that impact quality that will not be considered in this research since they are considered trivial, such as more investment into testing and having a dedicated quality assurance team. The defined model boundaries used in this research are shown in Table 3. Internal parameters of the system are known as endogenous whereas those outside the system boundary as exogenous and excluded ones.

Table 3 Model Boundaries

Endogenous	Quality, Schedule, Human Resource, Cost, Scope, Rework, Productivity (labour)
Exogenous	Requirements elicitation and documentation, Software Design, Software Maintenance
Excluded	Risk, Environmental factors (for example internet connection)

D. Analysis of root causes to software quality issues

Though difficult to quantify, software quality is regarded as crucial to achieve in any software project for the various reasons discussed already. Since software quality issues are the main problem to be tackled, the traditional Ishikawa (1985) diagram is used as the diagnostics tool shown in Fig 2.

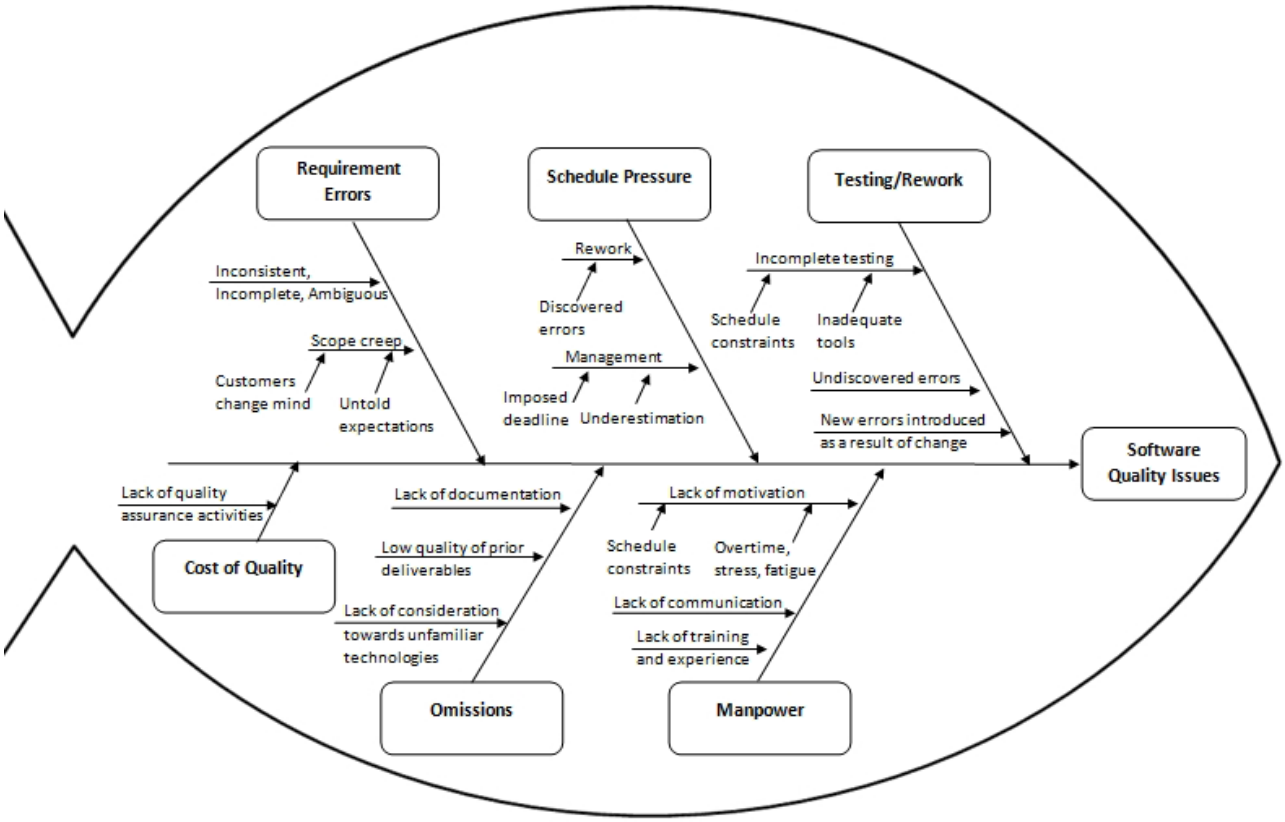

Fig. 2. Root causes of software quality issues

Requirements are the basis of software; software quality assurance activities such as software testing and error detection are dependent on requirements such that the Software Requirement Specification (SRS) document is the main artefact used to properly verify and validate software. If the requirements contain errors, this will lead to major software quality issues. Statistics have shown that errors follow the distribution: 60% design, 40% implementation. And 66% of the design errors are not discovered until the software has become operational (Galin, 2004). When errors are left undiscovered in software, it reduces the overall software quality.

Software testing is a part of the software quality assurance (SQA) process and is important to conduct for any software project. Studies have shown that extensively tested software contains 1-3 errors per 1000 lines of code (Galin, 2004), but not all software are extensively tested. Software testing has shortcomings, since in many projects it is done at the end of the software development life cycle (SDLC), where the schedule pressure is high. This often leads to inadequate testing and undiscovered errors that eventually bring a negative impact of the software quality. Moreover, other organisational factors amplify the schedule pressure such as underestimations on management behalf, absenteeism or resignations in the software project team, lack of motivation due to overtime or excessive rework. Under tight schedules, people work faster and longer to boost the output. Meanwhile, difficulties, stress and negligence incline to crop as a result of the negative impacts of schedule pressure. This in turn gives rise to errors that are injected, thereby impacting the software quality.

Manpower plays a pivotal role in the software development process as software remains a human-produced product as mentioned by Trammell *et al* in 2016. Even with the development of several tools and techniques to automate the production of code, the requirements and design must be produced by people. So a lot of human factors are involved in software production that can give rise to software quality issues such as lack of motivation, communication issues, training shortcomings, and lack of expertise or experience.

E. Cause-and-Effect Relationship

The Ishikawa diagram analyses the root issues of quality issues, nevertheless it is limited since the effect of the causes cannot be shown. Cause-and-effect (or causal loop) diagrams were used to show how the software project parameters impact software quality, in conjunction with the gathered literature review, analysed surveys and interviews.

F. Scope and Quality

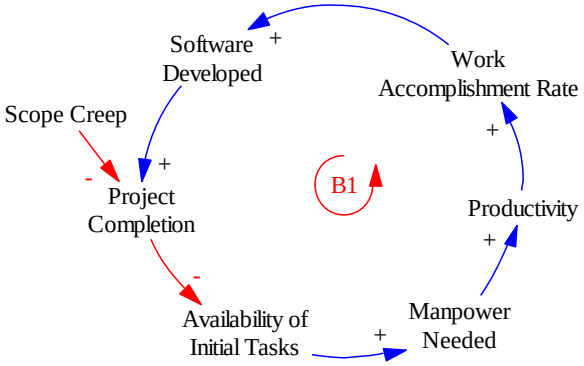

Fig. 3. Causal loop of scope

Balancing Loop B1 (Fig. 3): At the start of any software project, there are an initial number of tasks to be developed. These tasks require manpower effort to reach completion, hence related to the team productivity. Work is then accomplished producing the software incrementally which is a linear relationship. Moreover the more work accomplished, more software will be produced and vice versa, hence a positive causal direction. When software is developed, the initial amount of tasks diminishes giving rise to a balancing loop. However, changes in scope can counteract the balancing effect by increasing the number of initial tasks.

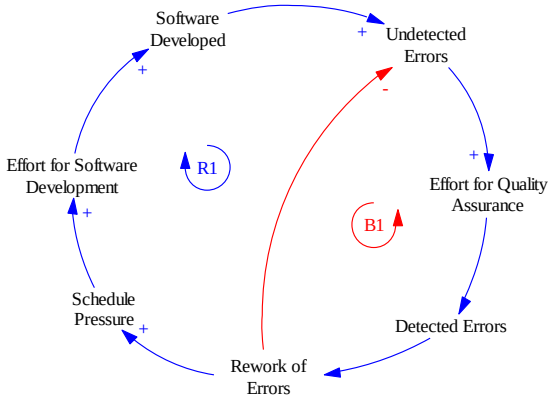

Fig. 4. Causal loop of software and rework process

Reinforcing Loop R1 (Fig. 4): It concerns the development of software and the rework process with respect to the error density. Software is being developed and contains undetected errors that require an effort from quality assurance. Undetected errors are then detected and go in the rework cycle. With an increase in the rework rate, there is an increase in schedule pressure where management re-calculates and increases the effort for software development as a result. The loop closes as the adjustments increases the software development rate.

Balancing Loop B1 (Fig. 4): It deals with the rework cycle of software development. When undetected errors in software pass through the quality assurance effort, errors are detected. This leads to rework of the error-containing tasks that eventually decreases the amount of errors in the software. Hence the amount of undetected errors is reduced producing the balancing loop behaviour.

G. Human Resource and Quality

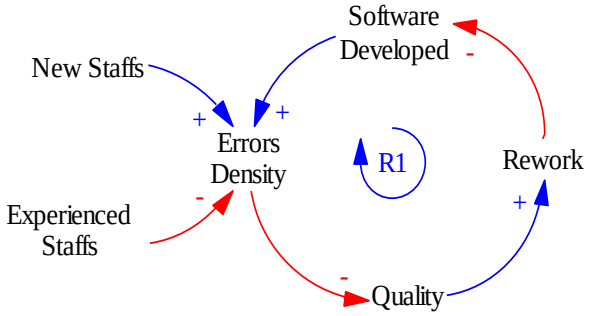

Fig. 5. Causal loop of human resource

Reinforcing Loop R1 (Fig. 5): The presence of more new staffs in a team compared to the number of experienced staffs increases the error density in the software. This is due to the fact that new staffs are more error prone than experienced staffs. Introduction of more errors in the software degrades its quality that implies a greater amount of rework that is needed. This in turn decrease the amount of software developed since effort is allocated in rework more than in development of the software giving rise to a reinforcing loop as more software developed will contain more errors as well.

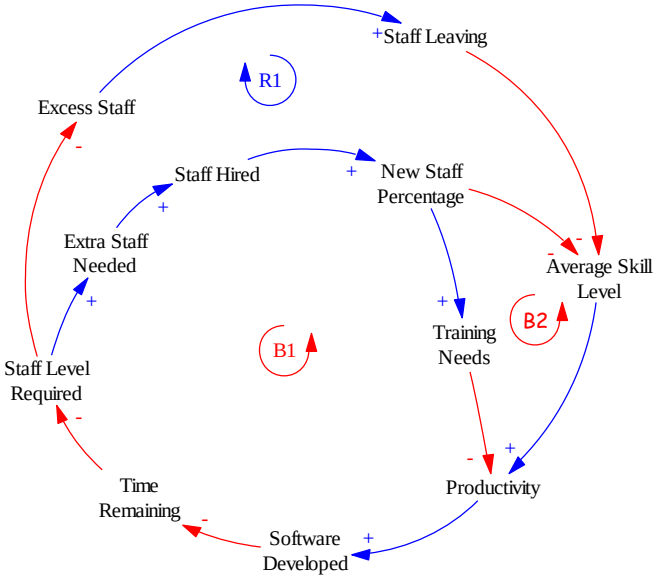

Fig. 6. Causal loop of productivity

Reinforcing Loop R1 (Fig. 6): When the staff level exceeds the required staff level, there is an excess amount of staff. The staff level is then adjusted by removing the excess staff by transferring the staffs to other projects. When experienced staffs leave the project, the average skill level is reduced. Normally, skills level increases team's productivity that causes more software to be developed. More software implies the time remaining decreases. By the end of the project, more staffs are required since there is lesser time giving rise to schedule pressure. There are also a need for more experienced staffs to correct errors detected.

Balancing Loop B1 (Fig. 6): When the staff level needed exceeds the staff level, extra staff is then needed. This is achieved through hiring of new staffs or transfer of new staff from other projects. New hires increase the new staff percentage in the team. This causes the average skill level to dilute since new staffs are not trained on the project. Skills in a team increase the productivity whereas a lack of resources decreases the productivity. Productivity causes more software to be developed with respect to time that reduces throughout the software life cycle. A reduction in time normally requires more staffs since there is schedule pressure when a project reached to its end. Injecting more staffs may later have consequences on quality. There are also a need for more experienced staffs to correct errors detected.

Balancing Loop B2 (Fig. 6): Boarding new staffs on the project will require experienced staffs to give them training. As experienced staffs are shifted on the training of new staffs, productivity on the project is reduced as there will be a shortage of staffs. When the productivity is low, the software development rate is reduced as well. Less software implies there is more time available and hence less staff level required.

H. Schedule/Time and Quality

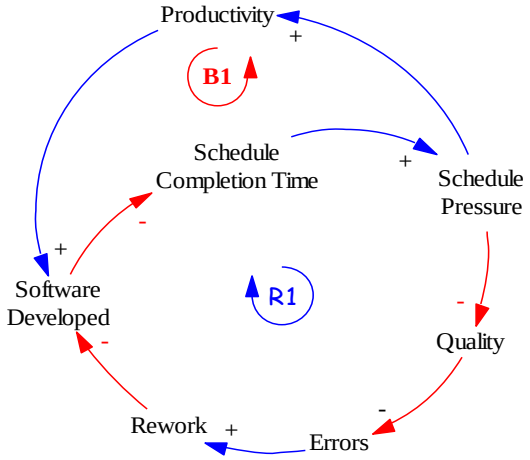

Fig. 7. Causal loop of schedule

Reinforcing Loop R1 (Fig. 7): It focuses on the schedule pressure and error rate. With increased schedule pressures the workforce will work harder to catch up. However, in a situation with pressure and stress the tendency to make mistakes increase as well, thus quality is affected. This relationship of schedule pressure with quality shows negative relationship. When programmers are tired, stressed or close to deadlines prompt decisions are required, which may lead to more error. With the added production rate they also produce more tasks per hour and may generate more errors per hour as a consequence. With an increase in the errors made the need for rework increases. The rework process is a process where faulty code has to be rewritten. People work harder and faster but not smarter. The result is a loop where more errors and rework leads to less software development that in turn leads to less progress and forecasts behind schedule. This will in turn lead to more schedule pressure and subsequently more errors committed.

Balancing Loop B1 (Fig. 7): It discloses the effect of added schedule pressure on productivity. When the progress reports and the forecasts indicate a project running behind schedule, the initial schedule pressure provides an increase in productivity. When production falls behind schedule, research shows that workers tend to cut their slack time, and devote more concentration to the work at hand. Staffs increase their man-hours in order to bring the project back on schedule. This effect increases the actual productivity as staffs work harder during working hours in order to close the time-gap. This in turn leads to an increase in productivity that provides new progress estimates. The increase in productivity provides a forecast within schedule and schedule pressure subsequently decreases.

VII. DYNAMIC MODELLING

The final model (Appendix I) consists of core stock and flow sub-systems that are the quality subsystem, basic software development and task implementation, project completion toggle, rework cycle, human resource allocation, schedule estimation, schedule pressure, productivity and cost estimation.

A. Quality Sub-system

Figure 8 depicts that quality primarily affects the rate of task implementation and error generation. When quality is low, more errors are generated per tasks developed giving rise to rework. Rework requires more time and effort that might possibly lead to schedule slippage and cost overruns. Quality has a significant impact on the project's overall progress and it is crucial to know the factors that determine the quality index.

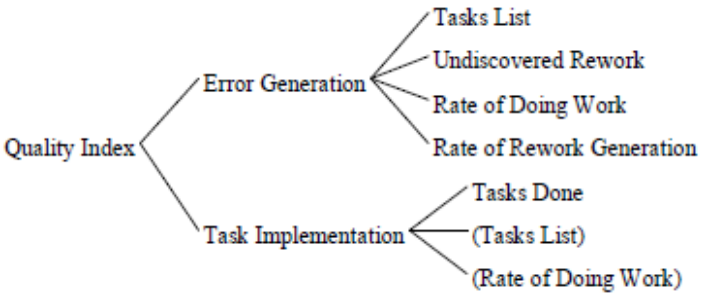

Fig. 8. Parameters that the quality index impacts

Even if the initial phase of software development produces high quality deliverables, high quality in overall software cannot be explicitly expected as quality is impacted by other software project parameters such as human resource planning and allocation for instance. Therefore a proper monitoring and control of the variations that occur in the quality index (QI) during the software development process is crucial. In the model, there are 8 parameters that directly affect the QI during software development shown in Fig. 9.

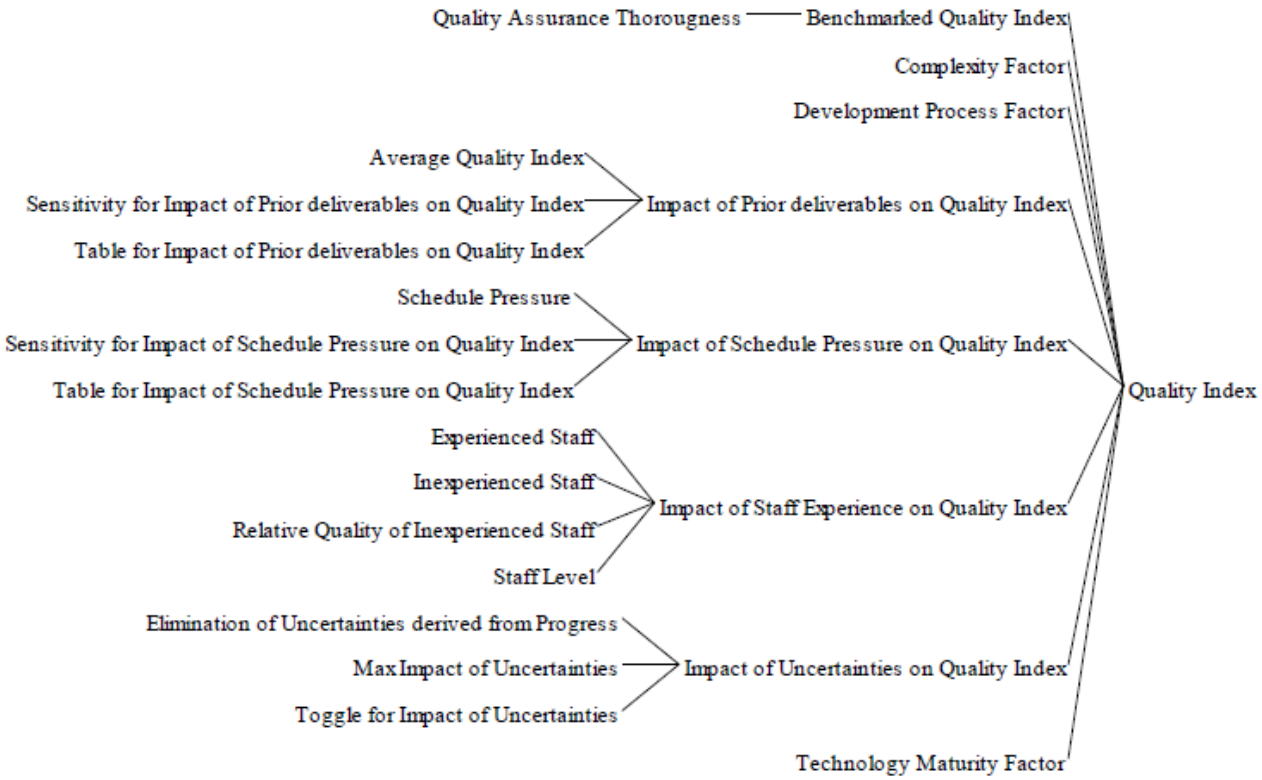

Fig. 9. Parameters that impacts the quality index

The relationship of the parameters with the quality index is described in Table 4.

Table 4 Parameters Affecting The Quality Index

Benchmarked quality index	The software organisation sets an initial benchmarked quality level with no other parameters influencing quality. It is derived from the quality assurance thoroughness and is modifiable during simulation for the proper calibration of the model.
Impact of uncertainties on quality index	Uncertainties impact the quality level since it allows for the generation of more errors in the software developed for instance uncertainty in requirements. The equation is driven by the elimination of uncertainties based on the project's fraction believed to be complete. As more work is accomplished, the more it gets validated by management and in certain cases by the customer, hence reducing the uncertainty factor.
Impact of schedule pressure on quality index	Schedule pressure affects quality adversely. When staffs work faster, they also make more mistakes in a "Haste makes Waste" effort.
Impact of prior deliverables quality on quality index	It represents the fact that undiscovered errors in previous tasks have a cascading impact on the current work deliverables. It implies that higher quality of prior deliverables increase the current quality level of the software.
Impact of staff experience on quality index	Since inexperienced staffs are more error prone than experienced staff, the experience dilution cause an impact on the quality of the software under development.
Complexity factor	It is used to capture the complexity level of the project. For instance, a project with many modules and user interfaces include built-in complexity resulting in rework for numerous reasons such as misunderstanding requirements, interface changes and module amendments.
Development process factor	It is used to capture the maturity level of the development process. A high value reflects a mature and rigorous development process and vice versa.
Technology maturity factor	It refers to the experience the development organisation has on the technology applied to the project where a high value reflects matured technologies.

B. Project tasks implementation and Rework cycle

Referring to the causal loop in Fig. 4, the development of software is governed by the rate at which the project's tasks are implemented by the developers. Similarly when tasks are developed, it is not necessarily correct as it may contain errors that are undetected. The tasks go through testing where the errors are discovered, and eventually lead to rework, hence more work to do. Fig. 10 shows the stock and flow relationship that is inspired from the work of Abdel-Hamid and Madnick (1989, 1991).

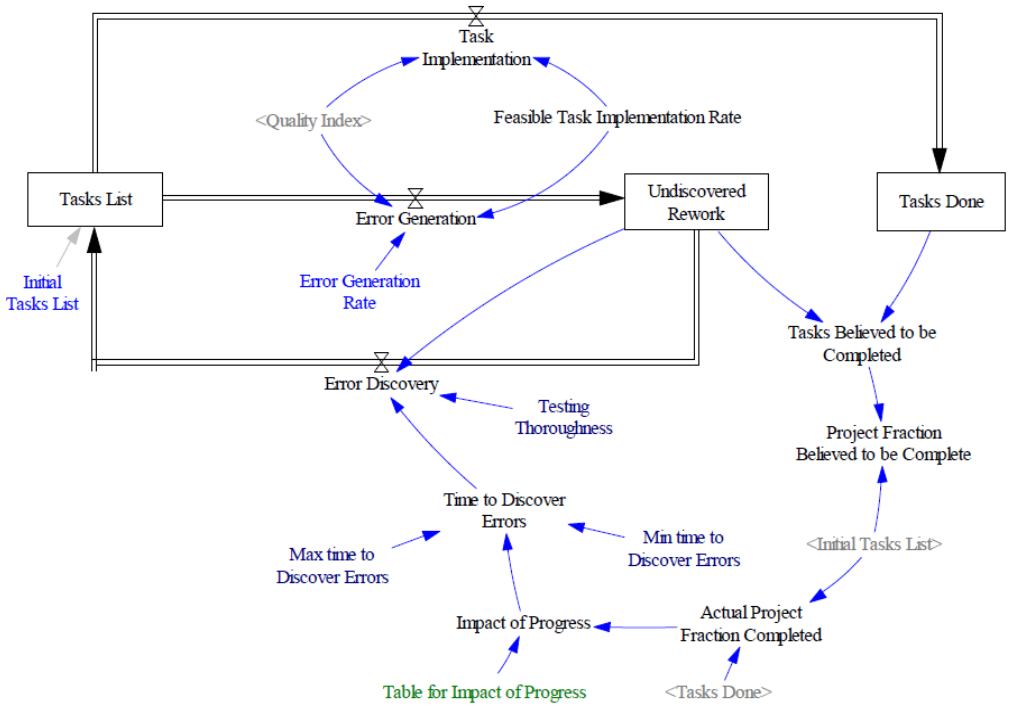

Fig. 10. Stock and flow of software development and rework

C. Staffing Sub-system

Project tasks are developed by human resource, the staffing sub-system is crucial in the model. People have a tendency to think that increasing the number of staffs on a delayed project will help gain time on the project. Brook's (1982) Law states otherwise: "*adding more staff to a late project will make it finish later*". Newly hired staff requires training and take time to gain experience thereby influencing the workforce productivity. This relationship is shown in Fig. 11.

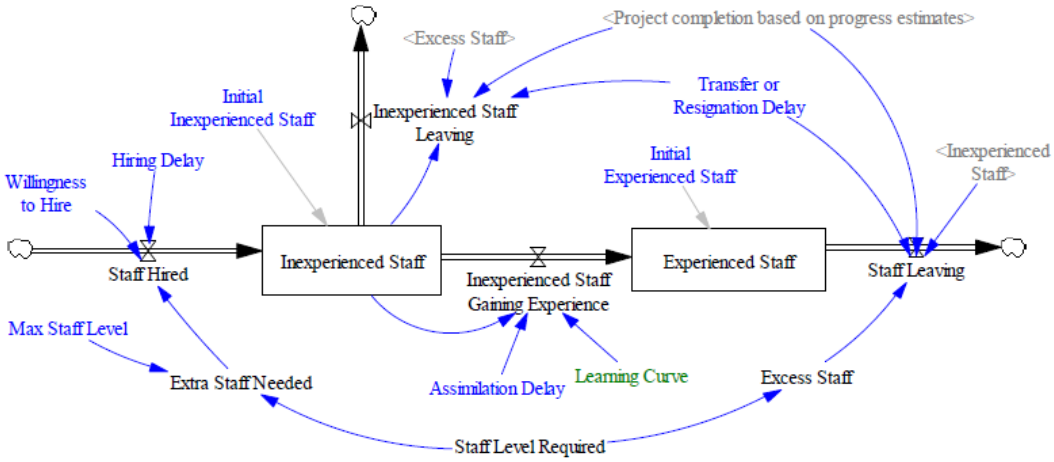

Fig. 11. Stock and flow of human resource (staffing)

The final model (refer to Appendix I) includes these sub-systems along with other sub-systems taking into consideration schedule estimation and schedule pressure, productivity and cost estimation.

VIII. MODEL CALIBRATION

Ensuring the level of quality during the development and testing process is crucial since it determines the final quality of the software. The final system dynamic model depicts the software project parameters that are dynamically interrelated by the use of stocks and flows. Lists of equations were derived from literature, survey and the software project scenarios used to come up with the final model. The model was calibrated using data from scenarios A, B and C as shown in Table 5.

Table 5 Calibration of model from scenarios A, B and C

		Simulated Value	Expected Value	MoE	Average MoE per scenario
A	Staff allocation	3	3	0.00	1.09 %
	Project completion	14.8125	14.5	2.16	
	Number of rework tasks	37.4899	37	1.32	
	Effort expended	42.3719	42	0.89	
B	Staff allocation	7	7	0.00	2.46 %
	Project completion	5.6875	5.5	3.41	
	Number of rework tasks	4.89712	5	2.06	
	Effort expended	26.094	25	4.38	
C	Staff allocation	3	3	0.00	1.07 %
	Project completion	14.1875	14	1.34	
	Number of rework tasks	65.1394	65	0.21	
	Effort expended	41.0873	40	2.72	
Average MoE for all scenarios A, B and C					**1.54 %**

Scenario B has the highest margin of error (MoE) of 2.46 % compared to scenario A and C that have a MoE of 1.09% and 1.07% respectively. Considering all the scenarios, the model has an overall MoE of 1.54% that produces a relatively low discrepancy as compared to the original data. Furthermore, the lower the margin of error, the closer is the simulated results to the original values.

IX. MODEL VERIFICATION AND VALIDATION

The structure in the dynamics model should be able to withstand certain extreme conditions that can occur in the real world. For example, if staffing level is set to zero, the productivity rate should fall to zero leading to stagnation in the task implementation rate. Robustness test is very effective to discover flaws in the model structure thus validating its usefulness. For example Fig.12 shows the effect of a zero staff level on a project in terms of task to do, tasks done and undiscovered rework. This depicts there were no stocks neither variable that had a negative depletion. Therefore the dynamic model had passed through extreme condition testing.

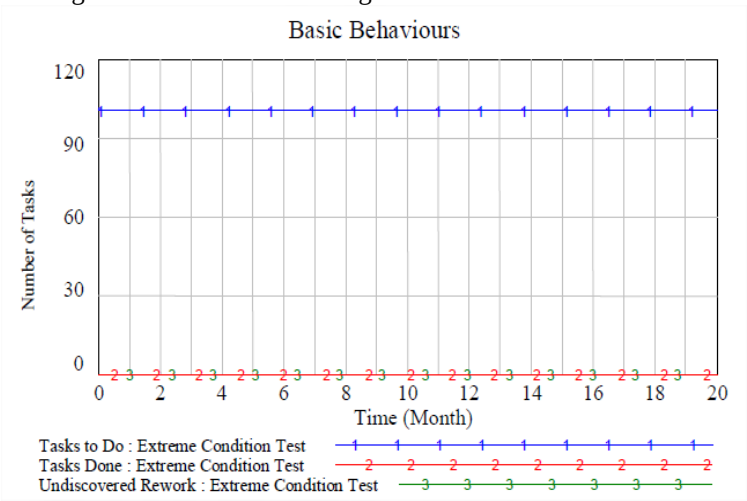

Fig. 12. Extreme conditions testing of the dynamic model

A. Validation of the derived Quality Index (QI)

Each scenario has a Customer Satisfaction Index (CSI) quality metric judged on a Likert scale of 1 (very dissatisfied) to 5 (very satisfied). From the simulated results and actual CSI, the recommended QI deviation can be assessed as shown by Quality Level in Table 6.

Table 6 CSI and Deviation from Benchmarked Q1 for Scenarios A, B and C

	Customer Satisfaction Index	Likert Scale Judgment	Deviation from benchmarked QI	Quality Level
Scenario A	3.8	Somewhat satisfied	8 %	**Average**
Scenario B	4.9	Very satisfied	1 %	**High**
Scenario C	2.1	Somewhat dissatisfied	20 %	**Low**

An average QI will produce an above average of customer satisfaction. Similarly a high QI will lead to very satisfied customer and vice versa. This implies that the CSI is directly proportional to the QI deviation thereby validating the QI derived. The simulated QI and quality level per scenario are shown in Table 7.

Table 7 Parameters Affecting The Quality Index

	Quality Index	Equivalent QI Percentage	Quality Level
Scenario A	0.87	87 %	**Average**
Scenario B	0.94	94 %	**High**
Scenario C	0.74	75 %	**Low**

A high QI indicate high quality in the software developed and vice versa. Since QI directly affect the rate of error generation, a high QI will generate the least amount of errors. Moreover as QI also impact the implementation rate; it will have a significant positive impact on the project's overall progress. Therefore a high QI will eventually be more cost effective as schedule will not be slipped; neither will the scope be exceeded due to rework. Human resources will not endure schedule pressure and lose motivation as is the case with excessive rework.

These analogies can be applied on other projects of similar types. Simulation will allow proper decision making during the software development process. The aim is to attain the highest possible level of quality in the software produced. Since only three software project simulations were done, the QI deviations are quite limited.

X. RESULTS AND DISCUSSION

Following the simulations made on the 3 scenarios, the proposed hypotheses were tested to eventually derive any logical recommendation. To test H_1, a hypothetical software project was considered with an initial team size of 6 experienced staffs and 100 tasks to be developed. The project was assumed to have a normal productivity of 4 tasks per person per month. The initial schedule estimation is 24 months. To test H_2, the simulations of scenarios A, B and C were considered since they had different levels of schedule pressure.

A. Hypothesis H1: Staff Experience

To test the staff experience, 4 different possibilities of staff mix were simulated on a pool of 6 staffs. The percentage mix was 75/25, 50/50 and 25/75 of experienced staff to inexperienced (or new) staff ratios as explained in Table 8. Using 100% of inexperienced staffs in a software project are unrealistic, therefore were not considered in the simulation.

Table 8 Experienced To Inexperienced Staff Mix

Ratio	Number of experienced staffs (used for ratio)	Percent of experienced staffs	Number of inexperienced staffs (used for ratio)	Percent of Inexperienced staffs
72/25	1.5	75 %	4.5	25 %
50/50	3	50 %	3	50 %
25/75	4.5	25 %	1.5	25 %
100	6	100 %	0	0 %

It can be interpreted from Fig. 13 that an increase in inexperienced staff in the team increases the amount of errors generated in the developed tasks thus in line with the fact that inexperienced staff are more error prone than experienced staffs. 100% of experienced staffs in a team are the best case scenario as shown by line 1 on the graph since it generates the least amount of rework. Dilution of the staff experience increases the number errors generated, shown by the lines labelled 2, 3 and 4 with respect to the percentage of staff ratio of experienced and inexperienced staffs. It is important to note that errors that are not discovered are termed as undiscovered rework. More the undiscovered rework, more will the cumulative tasks done with respect to the staff experience as shown in Fig. 14.

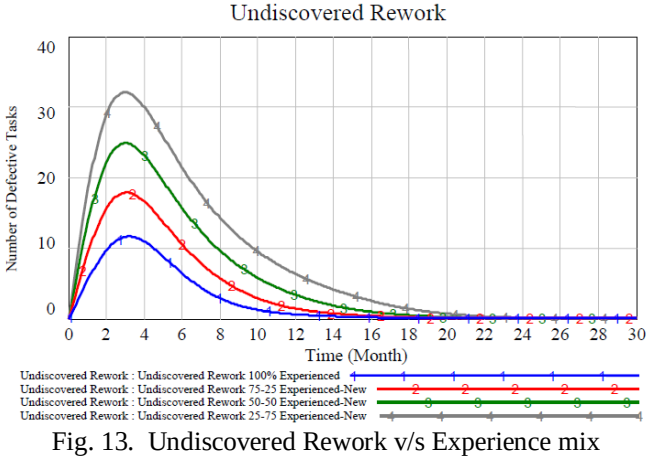

Fig. 13. Undiscovered Rework v/s Experience mix

A ratio of 25% experienced staff to 75% inexperienced is shown by line 4 in Fig. 13 and is likely to produce software with many errors. The detected errors (discovered rework) will be corrected and delivered for testing creating a vicious circle between development and testing phases. This is because more errors will be detected and sent back again for correction and so on. This vicious circle increases the cumulative tasks done drastically as shown by line 4 on the Fig. 14.

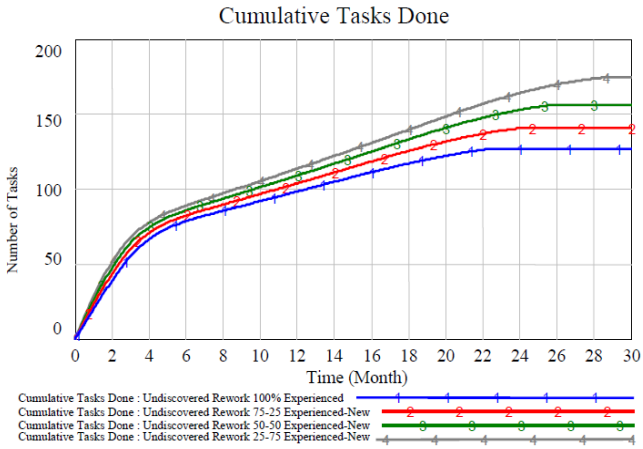

Fig. 14. Cumulative tasks done v/s Experience mix

When there are more errors generated in the developed tasks, this will cause fluctuations in the quality level of the software developed. It should also be noted that not all errors are discovered and corrected. Some errors may remain in the software and get delivered to the customer that might later turn into organisational losses as discussed in the introduction. Figure 15 shows that the staff experience impacts the level of quality significantly. The best case for this project is shown by line 1 where there were 100% experienced staffs within the team. The more the experienced staffs are diluted by increasing the number of inexperienced staffs on the team, the quality level falls, as shown by lines 2 and 3 where the experience ratio was 75-25 and 50-50 percent mix of experienced staff and new staff. With respect to the vicious circle discussed previously, line 4 shows that quality is likely to fall even more compared to other situations since there is a lack of experience in the team. Line 4 indicates that probability of error identified will still be high due to low quality, even if the software has already been delivered to the customer.

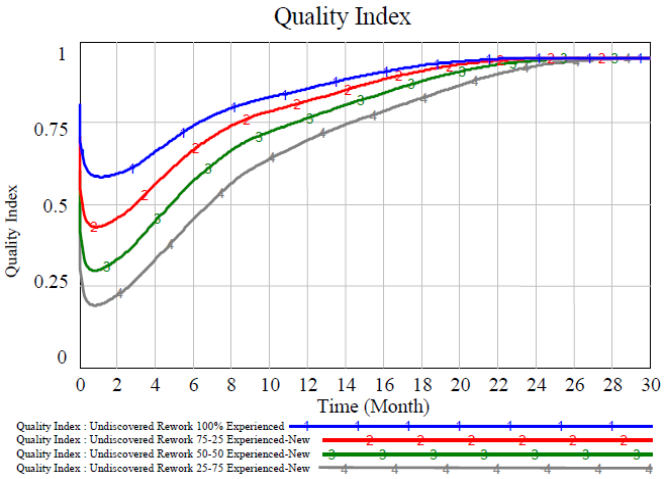

Fig. 15. Quality Index v/s Experience mix

It should also be noted that quality is driven by many other factors and not just staff experience mix. These factors are not easily isolated; hence assumptions were made to be able to understand the effects and magnitude that can result from the staff experience dilutions. It is clear that the right skills are critical to maintain a good quality level in the software developed as shown by the simulations.

Deadline is often slipped when there are more errors than expected in the work deliverables. More workload eventually requires more manpower. The experienced staff will work faster and hence finished earlier compared to inexperienced staffs. Hence the staff experience mix also impacts the estimated schedule of the project. The effect and magnitude on the project's schedule is shown in Fig. 16 where the "project completion" variable toggles from 0 (indicating project not completed) to 1 (indicating project completed). Line 1 shows the project ends before the expected duration of 24 months when the team comprised of 100% experienced staffs. Line 2 shows that the project ended within its estimated schedule. The remaining experience mix ratios delayed the project further beyond the estimated schedule due to several factors for instance the rework factor as discussed above. The simulations show that an increased dilution in staff experience makes the project more prone to slip the initially estimated schedule.

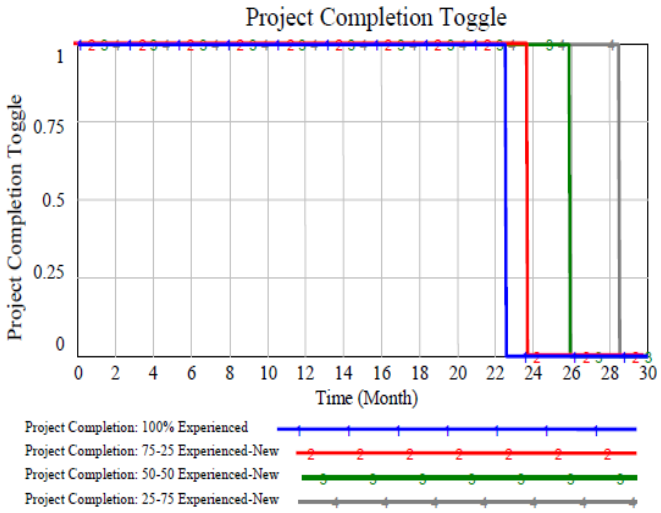

Fig. 16. Project completion v/s Experience mix

Although the best case scenario is explicitly the one with 100% of experienced staffs on a team, it is not always possible to allocate only experienced staffs in real life situations. The 75-25 experienced to inexperienced staffs mix however is more practical and a good practice too, since the project ends in the expected schedule of 24 months as shown by line 2. Moreover, the effort of new staff members in a team will also reduce the project's cost since experienced staffs are more costly compared to new staffs in term of salaries. With such simulations in hand, the software project manager will be in a better position to allocate the right proportion of experienced and inexperienced staffs on his/her software development team.

B. Hypothesis H2: Schedule Pressure

In an environment of schedule pressure, staffs work faster and longer hours in order to boost their performance in order to meet the initially set goals. However, when there are rush and unrealistic goals, people tend to take shortcuts such as eliminating functionalities and sacrificing quality. These have an adverse effect leading to more rework and increases the workload as a consequence.

Scenarios A, B and C had different levels of schedule pressure; hence their simulations are used for comparison as shown in Fig. 17. Scenario B did not experience any kind of schedule pressure since the project ended half a month before the estimated schedule shown by the flat line 2. Scenarios A and C however show respective overshooting and collapsing behaviours when the projects were near to their scheduled deadline. Nevertheless the schedule pressure exerted by Scenario C (line 3) was much higher than that of Scenario A (line 1). This was mainly because of a lack of staffs and a more than expected increase in the amount of rework in Scenario C compared to B.

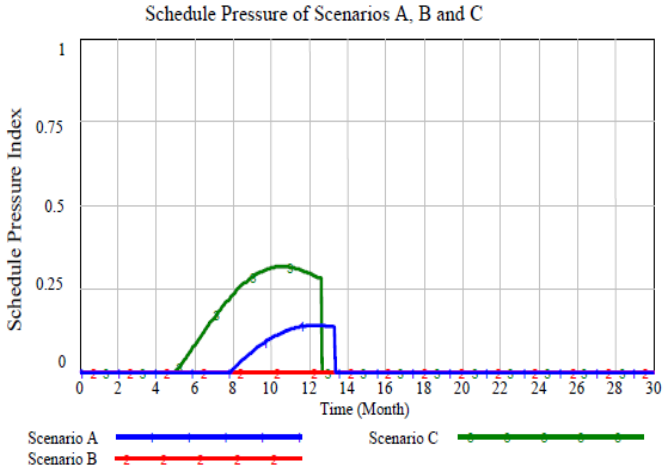

Fig. 17. Schedule pressure comparison of Scenarios A, B and C

Since schedule pressure has an impact on the quality of software throughout the development process, the scenarios with the highest magnitude of schedule pressure is bound to produce lower quality software. In this case, Scenario C showed the highest peak of schedule pressure (line 3). Scenario A however, had approximately half the degree of schedule pressure compared to C, hence the quality of A (line 1) was superior to that of C (line 3) as shown in Fig. 18, since schedule pressure has an adverse impact on quality. Scenario B on the other hand did not suffer from schedule pressure, hence was closest to the initially set benchmarked quality index producing a quality index deviation 1%.

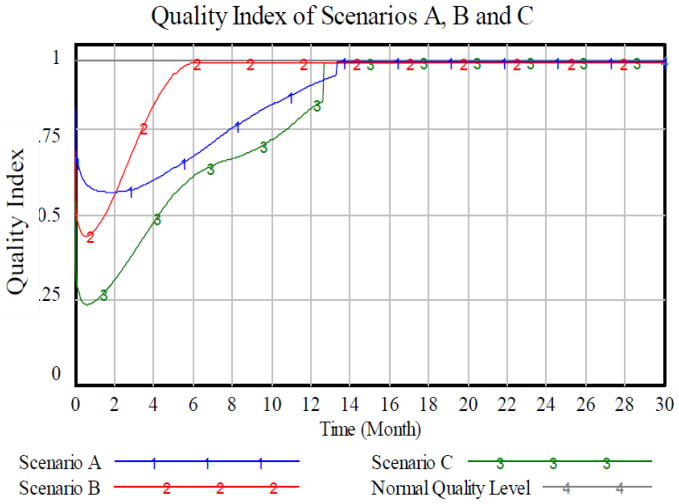

Fig. 18. Quality Index of Scenarios A, B and C

It is worth noting that although schedule pressure has a negative impact on quality; it also contributes positively towards an increase in productivity. If the right balance is found, it can also be regarded as a good thing in a project life-cycle. Since the quality index of Scenario C is far below the benchmarked quality index, it is recommended that Scenario C should have taken measures such as allocation of more experienced staffs on the team rather than allocating inexperienced staff.

XI. CONCLUSION

During software development process, both management and developers lose track of quality fluctuations within the software developed. Research has shown that there are many software project parameters that dynamically impact the quality of software during the development process. But there are no adequate frameworks that solve this particular issue though software quality is a pertinent problem to many organisations today. It is also perceived from literature that there are diverse frameworks and standards such as ISO, CMMI, and TQM used by organisations to improve their processes and therefore gain in terms of quality. However, they show a high development cost implication and is not at the reach of every organisation. There is also a lack in the transparency of quality fluctuations throughout the improved processes.

In software engineering, the notion of measurement has become fundamental since it is directly related to performance. An imminent challenge for this paper is the lack of reference methodology to measure the fluctuations of quality, which is the Quality Index (QI) within the software development process. System dynamics concepts and modelling approach provided both a challenge and an opportunity for the QI measurement in the software context as discussed in the paper. The core software development sub-systems was designed using causal and effect concepts that eventually led to the development of the dynamic stock and flow model. The final model consists of core stock and flow sub-systems that are the quality subsystem, basic software development and task implementation, project completion toggle, rework cycle, human resource allocation, schedule estimation, schedule pressure, productivity and cost estimation. The model allowed for various simulations with visualisation of the fluctuations of many parameters including the QI. The derived QI was validated with the Customer Satisfaction Index (CSI) and showed a direct proportionality, hence their high correlation.

More findings are in terms of staff experience dilution and the impact of schedule pressure that were tested as hypotheses. Although the best case scenario is explicitly the one with 100% ratio of experienced staffs on a team, it is not always possible to allocate only experienced staffs in real life situations. The 75-25 experienced to inexperienced staffs ratio however was more practical and a good practice too, since simulations showed that the project ended in the expected schedule. Schedule pressure is regarded as having a negative impact on quality, but this is not always the case. If the right balance of schedule pressure is determined, it can also be regarded as a good thing in a project life-cycle since it contributes to increasing the productivity as well. The idea is not to work out every detail in advance, not to test everything to absolute certainty; but to work on the fine detail of the software project parameters within its development phase with the aspect of quality in mind.

REFERENCES

[1] Abdel-Hamid T.K., Madnick S.E., 1989. Lessons learned from modeling the dynamics of software development. Communications of the ACM 32(12): 1426-1438.

[2] Abdel-Hamid T.K., Madnick S.E., 1991. Software project dynamics: an integrated approach. Englewood Cliffs: Prentice Hall.

[3] Abdel-Hamid T.K., 1984. The dynamics of software development project management: an integrative system dynamics perspective. Thesis (PhD). Sloan School of Management, MIT.

[4] Austin R.D., 2001. The effects of time pressure on quality in software development: An agency model. Information Systems Research 12(2): 195-207.

[5] Brooks J.R.F.P., 1982. The Mythical Man-Month. Addison-Wesley, Boston, USA.
[6] Casson D., 2005. An in-depth analysis of the current state and readiness of IT organisations to adopt ITIL-based processes. North American ITIL Assessment, Evergreen Systems.
[7] Cater-Steel A., Wui-Gee T., Toleman M., 2006. Challenge of adopting multiple process improvement frameworks. Proceedings of 14th European conference on information systems.
[8] Chakravorty S.S., 2010. Where process-improvement projects go wrong. Wall Street Journal - Eastern Edition 255.19.
[9] Charette R.N., 2005. Why software fails. IEEE spectrum. 42.
[10] Deloitte, 2003. IT Governance Practices in the Irish Public Sector. Dublin.
[11] Devedzic V., 2001. Software Project Management. Handbook of Software Engineering and Knowledge Engineering 2, 419-446.
[12] Forrester J.W., 1961. Industrial Dynamics. MIT Press. Cambridge. Massachusetts.
[13] Gerke L., Ridley G., 2006. Towards an abbreviated COBIT framework for use in an Australian State Public Sector. Proceedings of the 17th Australasian Conference on information Systems.
[14] Galin D., 2004. Software quality assurance: from theory to implementation. Addison-Wesley.
[15] Hochstein A., Tamm G., Brenner W., 2005. Service-Oriented IT Management: Benefit, Cost and Success Factors. Paper presented at the European Conference on Information Systems, Regensburg. Germany.
[16] Ishikawa K., 1985. What is total quality control?: the Japanese way. *Englewood Cliffs*, NJ: Prentice-Hall.
[17] Kathy S., 2010. Information Technology: Project Management. Course Technology Press. Boston, US.
[18] Land S.K., Mcgroddy J.C., Moore J.W., 2005. Learning from Software Failure. Neumann P.G., Spix G., Weyuker E.J.
[19] Lindstrom L., Jeffries R., 2004. Extreme Programming and Agile Software Development Methodologies. Information Systems Management 21(3): 41-60.
[20] Lyneis J.M., Ford D.N., 2007. System dynamics applied to project management: a survey, assessment, and directions for future research. System Dynamics Review 23(2-3); 157-189.
[21] Meso P., Jain R., 2006. Agile Software Development: Adaptive Systems Principles and Best Practices. Information Systems Management 23(3): 19-39.
[22] Molokken-Ostvold K., Jorgensen M., 2005. A Comparison of Software Project Overruns-Flexible versus Sequential Development Models. IEEE Transactions on Software Engineering 31(9): 754-766.
[23] Paulk M.C., 2004. Surviving the quagmire of process models, integrated models, and standards. In: Annual Quality Congress Proceedings, Toronto, 429-438.
[24] Potgeiter B.C., Botha J.H., Lew C., 2005. Evidence that use of the ITIL framework is effective. Paper presented at the 18th Annual Conference of the National Advisory Committee on Computing Qualifications, Tauranga, NZ.
[25] Rai V.K., Mahanty B., 2001. Dynamics of Schedule Pressure in Software Projects. The Proceedings of the 20th International Conference of the System Dynamics Society, Palermo, Italy.
[26] Rose K., 2005. Project quality management: why, what and how. J Ross Pub. USA.
[27] Sterman J.D., 2003. Business Dynamics: Systems Thinking and Modeling for a Complex World. Irwin/McGraw-Hill, Chicago, USA.
[28] Vidal H., Wan J., Han X., 1998. Capability Models: ISO and CMM. Kansas: Department of Computing and Information Sciences, Kansas State University.
[29] Trammell, M.., Allen, M., and Stuart E. Madnick, 2016. Effects of Funding Fluctuations on Software Development: A System Dynamics Analysis. Engineering Management Journal 28.2: 71-85.
[30] Wiegers K.E., 2009. Software requirements. Microsoft press.

APPENDIX 1
Final System Dynamics Model