Simulation-Based Decision Support for Bringing a Project Back on Track: The Case of RUP-Based Software Construction

Elham Paikari
Department of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
epaikari@ucalgary.ca

Guenther Ruhe
Department of Electrical and Computer Engineering and Department of Computer Science
University of Calgary
Calgary, Canada
ruhe@ucalgary.ca

Prashanth Harish Southekel
SAP Practice
Accenture Canada
Calgary, Canada
prashanth.harish@accenture.comx

***Abstract*—RUP-based development has proven successful in various contexts. The iterative and phased development approach provides a framework for how to develop software efficiently and effectively. Yet, there are plenty of occasions that the projects go off-track in terms of the key parameters of the project such as quality, functionality, cost, and schedule. The challenge for the software project manager is to bring the project back on track. Simulation, in general, and system dynamics based simulation in particular, is established as a method to pro-actively evaluate possible scenarios and decisions. The main contribution of this paper is a method called SIM-DASH; it combines three established techniques for providing decision support to the software project manager in the context of RUP-based development. SIM-DASH consists of (i) a system dynamics modeling and simulation component for RUP-based construction, (ii) dashboard functionality providing aggregated and visualized information for comparing actual versus targeted performance, and (iii) knowledge and experience base describing possible actions that have proven successful in the past for how to bring a project back on track. As part of (iii), decision trees and experience-based guidelines are used. The interplay between these three components provides pre-evaluated actions for bringing the current project iteration back on track. As proof-of-concept, a case study is provided to illustrate the key steps of the method and to highlight its principal advantages. For this purpose, SIM-DASH was substantiated retrospectively for a real-world SAP web system development project within the banking field. While the method is applicable for different issues and scenarios, we study its impact for the specific issue of adding personnel to testing and/or development in order to ensure improved project performance to achieve established quality levels of feature development.**

***Keywords*-decision support; project management; system dynamics modeling and simulation; effort reallocation; RUP; construction phase; case study**

I. INTRODUCTION

Software project management is an experience-based undertaking and application of skills, tools, and techniques to complete a set of activities including, risk management, scheduling, estimating effort, monitoring development, and tracking project progress [1]. All these activities become more complex over time due to factors such as increasing number of dependent variables, their dependencies, requirements volatility, schedule changes, and cost overruns to name a few. So, decisions made without considering the holistic view could potentially cause project failure. Improper estimations and decisions made without considering future consequences can also result in failure (i.e. delayed project with poor quality) [2].

Given the repercussions of a failed project, it is imperative to have a "simulator" to dynamically factor-in the variables in the project system. System Dynamics (SD) is a method for studying the behavior of complex systems, the changes that can occur over time and are influenced by the cause-effect relationships that are implemented with the feedback loops in SD. System Dynamics feedback behavior, nonlinear relationship and delays are the main features of software development. While traditional methods and tools cannot deal with such a dynamical behavior, ignore the interactions with a static view, and partial presentation of different parts of development process, SD does help managers to learn about dynamic complexity of a process and design more efficient policies in the long term [3].

In the past two decades, especially since [4] addressed the "why, what and how" questions of process simulation, the number of applications of SD in project management has increased [5]. Studies have investigated different aspects of software project management, such as process improvement, quality assurance, risk management effects of pair programming, different testing policies, development effort allocation, etc.[6].

We have employed the SD approach to simulate a software development process inspired by a real world process from SAP Canada. This model looks at different iterations of the construction phase of a RUP-like incremental development process. In addition, we introduce a dashboard built on top of an established SD simulation model as a decision support tool for software project management. As a management tool, this dashboard is used to get an overview of enterprise health and provide decision makers with entering the necessary inputs. Then, as a graphical user interface, it displays the output results based

upon any alteration of key parameters, in the form of graphs, to highlight the important modifications. In this paper, we investigate a use case scenario based on a retrospective real world project.

This paper is organized as follows: Section II provides a brief review about the application of SD in project management and related causes leading to the off-track situation of a software development project. Section III states the problem explored in this paper, and section IV explains our methodology to tackle the problem. Furthermore, Section V contains an illustrative example of employing the methodology and presents the simulation results. Section VI offers the conclusion and outlook to future work.

II. Related Work

As Fred Brooks stated in 1987: "… major problems with software development are not technical problems, but management problems" [7]. Now, twenty-five years later, things have not changed dramatically. Charette has analyzed factors about why software projects fail so often [8]. Among the 12 factors mentioned, half of them are closely related to project management.

From a project management perspective emerging tools, such as effort estimation models provide the initial estimation essential for project planning as a part of the project management task. In particular, machine-learning methods have been used, including case-based reasoning, neural networks, fuzzy systems, and evolutionary modeling. Unfortunately, most of them are static (time independent), and lack the ability to capture the dynamics of management decision making. In addition, they are not useful when adjustments are needed after the project starts [9].

In [4], the authors performed a systematic literature review of software process simulation modeling research. They ranked SD as the most popular technology in software process simulation modeling. Their review indicates the most common initiatives for simulation of software process are time, effort, and quality.

There are studies that look at the SD modeling approach as a decision support tool. In [10], an illustrative example of a simulation model from the automotive industry presented the process simulation as a useful management tool for selection among alternative plans regarding the project performance. In addition, in the context of software process simulation some aspects of strategic software project management have been investigated to support process improvements, software project management, and training. It also presents recommendations for selecting a simulation model approach for practical application.

In terms of the training aspect of SD simulation model, in [11] the model helps to understand the key factors and behaviors in complex scenarios. Brooks' law is implemented. The effects of Pair Programming in eXtreme Programming (XP) investigated as case study with the intention to exhibit basic concepts.

In [12], a SD model is presented to integrate lifecycle and project management activities and policies. The study demonstrates that a complete understanding of the software-development process needs to synthesize the knowledge of all parts of an integrated process. They investigate the cost-schedule trade-off in software development and consider all the relevant parameters in an integrated SD model. Later, the same author in [13], focuses on the dynamics of software project staffing in software development for the purpose of effective software project management. They present a SD model to answer 'what-if' questions in effort allocation during the software development process.

Some studies look at the simulation results gained from dynamic behaviour of key parameters to support managers in evaluating different policies. In [14] a SD simulation model is used to depict the key factors that impact on increases in workforce and considers the overwork.

On the other hand, a number of studies have investigated the factors that cause software projects to fail. Majority of them discuss failure in general, and provide lists of risk and failure factors [15], [16], [17]. In [18] the authors investigated the data from approximately 70 failed projects and analyzed 57 influential development and management factors as potential root-causes. They reported all of the projects suffered from poor project management; in addition, many of them were also confronted with organizational factors.

In this research, we employ a schematic decision tree to describe the structure of possible root causes of an off-track project. In addition, we present a dashboard to visualize the consequences of different parameter settings.

III. Modeling and Problem Statement

In this study we consider the construction phase of the RUP-based development methodology [19] inspired from actual development processes at SAP Canada.

In an iterative RUP-based software development process, what needs to be built and how it should be built is defined during the inception and elaboration phases. The construction phase is strictly about building the product, then deploying to end users in the transition phase. In the construction phase, the primary objective is to build the software system by development of features. The construction phase incorporates the implementation and testing disciplines.

The construction phase is selected because of numerous construction iterations and excessive effort to divide the use cases into manageable segments that produce demonstrable prototypes. Moreover, the main focus of this phase on implementation tasks causes a number of feedback loops; especially since managing this phase involves changing effort allocation policies. The aforementioned characteristics make this phase complex and dynamic enough to be an effective choice for modeling with SD. The 5 sets of influential parameters in construction are considered in the

SD model, these parameters are explained in following 5 subsections in detail.

A. Quality Levels of Features Implementation

Three levels of quality are considered for the features to be implemented; they are called quality levels zero (Q0), one (Q1), and two (Q2).

Definition 1: A feature is in Q0 if it has passed programming and subsequent unit testing. At any point in time *t*, the set of all features at Q0 is called FQ0(t).

Definition 2: A feature is in Q1 if it has passed the integration test. At any point in time *t*, the set of all features at Q1 is called FQ1(t).

Definition 3: A feature is in Q2 if it has passed the system test. At any point in time *t*, the set of all features at Q2 is called FQ2(t).

With the total number of features to be implemented at the end of a specific iteration denoted by TNF, we have inequalities stating that the number of features in a given quality level cannot be greater than the number of features in a higher quality level, at time t, before the end of that iteration.

$$0 \leq FQ2(t) \leq FQ1(t) \leq FQ0(t) \leq TNF \qquad (1)$$

for all t taken from the interval of the iteration.

B. Quality Levels and Releated Skill Sets

In quality levels one and two testers execute the test cases and report the defects. Subsequently, programmers fix the defects. Once a defect is dealt with by the development team, it should be retested to confirm that it is fixed correctly. Testing, defect fixing, and retesting should be completed for both quality levels Q1 and Q2. Quality levels and their associated skills are listed in Table I.

TABLE I. QUALITY LEVELS, ASSOCIATED TESTING LEVELS, AND THE NECESSARY SKILL SETS TO COMPLETE EACH QUALITY LEVEL

	Q0	Q1	Q2
Testing level	Unit Test	System Test	Integration Test
Skill set	Programming (coding)	Testing Programming (defect fixing)	Testing Programming (defect fixing)

If all required features are implemented and signed-off from Q2, then any defects in the existing functionality EX (completed features from previous iteration of the project) also need to be removed.

C. Defects Transition from Programming through Feature Sign Off in Q1 and Q2

In our model functionality is described by the features implemented in the project. On the other hand, quality is achieved incrementally by passing different levels of quality. The quality levels have different consequences in the process, such as different amount of effort to remove the related defects.

For testing process three levels of quality are considered in the model. These levels conform to the V-Model application in testing [20]. Quality level zero Q0 is achieved when the programmers complete programming and unit testing. Then, two other levels are based upon the (i) integration testing and (ii) system testing, which are concerned with the system design and functional specification, respectively. In quality level one Q1 and two Q2, testers execute the test cases and report the defects. Programmers then fix the defect; once a defect is dealt with by the development team it should be retested.

A defect in each level of quality is subject to the transitions presented in Figure 1. After programming, features can be considered as signed-off from quality level zero, Q0. During programming some defects may be injected. Then the flow of defects in quality level one Q1 and two Q2 is the same. Outstanding defects are fixed and then retested to confirm if the defect is removed. If the retesting result is successful the defect is removed. Otherwise, it goes back to fixing.

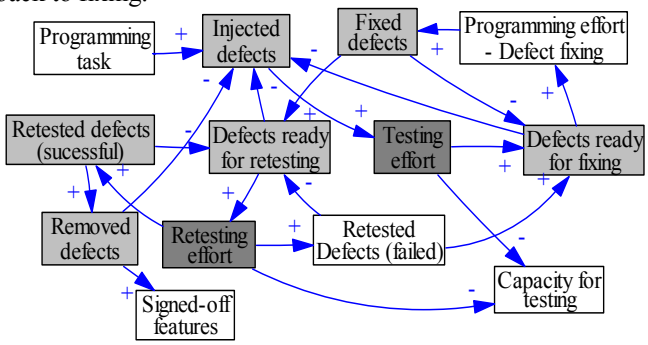

Figure 1. Causal diagram of the life-cycle of feature sign off in both Q1 and Q2.

D. Efforts for Programming and Testing

Definition 4: EPQ0(t) is the available programming effort at time t for quality level Q0. EPQ1(t), EPQ2(t), and EPEX(t) denote the available programming effort for defect removal tasks at quality level Q1, Q2, and for existing functionality, at time *t*.

Definition 5: IdleCP(t) is the part of the available programming capacity that is not used by any other programming effort at time *t*.

The capacity available for programming at time *t* is called CP(t). Changing the value of these programming efforts, representing the allocation for each one of these efforts, can affect the stabilization policy. For example, allocating less effort EPQ0(t) for programming will result in more capacity to available for defect removal at time *t*. The relation among these efforts and the programming capacity is formulated in (2).

$$CP(t) = EPQ0(t) + EPQ1(t) + EPQ2(t) + EPEX(t) + IdleCP(t). \qquad (2)$$

In addition to the efforts defined in the programming at each level of quality, there is an assigned effort allocation for

testing. In quality level zero Q0, features are coded and there is no testing effort allocated for quality level zero Q0; features in this level are tested by programmers.

Definition 6: ETQ1(t), ETQ2(t), and ETEX(t) denote the available effort for testing in quality levels Q1, Q2, and for the existing functionality EX, at time *t*.

Definition 7: ERQ1(t), ERQ2(t), and EREX(t) denote the testing effort for retesting tasks in quality levels Q1, Q2, and the existing functionality EX, at time *t*.

Definition 8: IdleCT(t) is the part of available testing capacity that is not used by any other testing effort at time *t*.

In our method, (re-) allocating of effort is based on the number of defects found during testing in Q1 and Q2. More effort for testing reveals more defects for fixing. Even after fixing, expending effort for retesting reveals new defects that are injected as a result of fixing or failed fixing. The policy of effort allocation in testing is based on the available testing capacity called CT(t). Their relations are shown in (3).

$$CT(t) = ETQ1(t) + ETQ2(t) + ETEX(t) + ERQ1(t) + ERQ2(t) + EREX(t) + IdleCT(t) \quad (3)$$

Complete information about this SD model is beyond the scope of this paper. To provide views of relations and equations there are explanations and screen shots from the model available in [21].

E. Constants in the Simulation Model

There are a number of constants used in this SD model. The actual values of these constants are determined based on expert knowledge gained from previous similar projects. The meaning of the constants is listed below:

TNF – Total number of features requested to be implemented at the end of a specific iteration.

ANFP – Average number of features programmed in parallel during feature sign-off.

NDFH – Number of defects fixed per hour.

NDRH – Number of defects retested per hour.

NDDH – Number of defects detected per hour.

PDF – Percentage of defects for retesting which were fixed successfully.

NDIDF – Number of defects injected per defect fixed.

NDIQ1 – Number of defects injected per task in Q1.

NDIQ2 – Number of defects injected per task in Q2.

NDIEX – Number of defects injected per task in EX.

IV. METHODOLOGY

The main contribution of this paper is to provide a method called SIM-DASH to combine three established techniques for the purpose of providing support to the software project manager asking the question: How to bring my project back on track? SIM-DASH consists of (i) a SD modeling and simulation component describing RUP-based construction and originated from a real-world project with SAP Canada, (ii) dashboard functionality providing aggregated and visualized information for comparing actual versus predicted iteration performance, and (iii) a knowledge and experience base describing possible actions, proven successful in the past, to bring a project back on track.

To illustrate the suggested methodology, we present a small example for an off-track situation at a particular cycle, C_i, with defined start and end time [a, b]. Using the dashboard, this situation is simulated by the SD model. The model is able to predict and investigate the results for the next cycle, C_{i*} = [b+1, c]. We are now able to compare the plan with the new situation, in this cycle, C_{i*}, as shown in Figure 2. The gap between the target and actual number of features in C_i and C_{i*}, signed-off from each level of quality, $FQ0/1/2_{target}(t)$, $FQ0/1/2_{actual}(t)$, is shown by the bi-directed arrows in Figure 2.

$$OT(T): FQ0_{actual}(t) < FQ0_{target}(t) \; OR \; FQ1_{actual}(t) < FQ1_{target}(t) \; OR \; FQ2_{actual}(t) < FQ2_{target}(t) \quad (4)$$

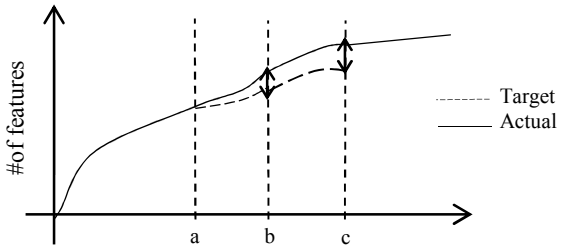

Figure 2. Illustration for an off-track situation.

The proposed method SIM_DASH consists of eight steps. The sequence and dependencies are shown in Figure 3.

Step 1: SD modeling of RUP-based construction phase.

Step 2: Detecting the deviation between target and actual performance values and measurement of key parameters of actual development processes.

Step 3: Request actions to bring the project back on track based on the deviations analyzed between target and actual performance values.

Step 4: Recommend actions on how to bring the project back on track.

Step 5: Run simulation scenarios to pro-actively evaluate the impact of the proposed changes.

Step 6: Analysis of results and provision of recommended actions.

Step 7: Implementation of the recommended action in the actual project.

Step 8: Learning: Add the experience gained from applying the recommended action to the existing knowledge and experience base.

In what follows, further details are given related to the content of the workflow introduced in Figure 3. This includes the SD simulation model, the implementation of the

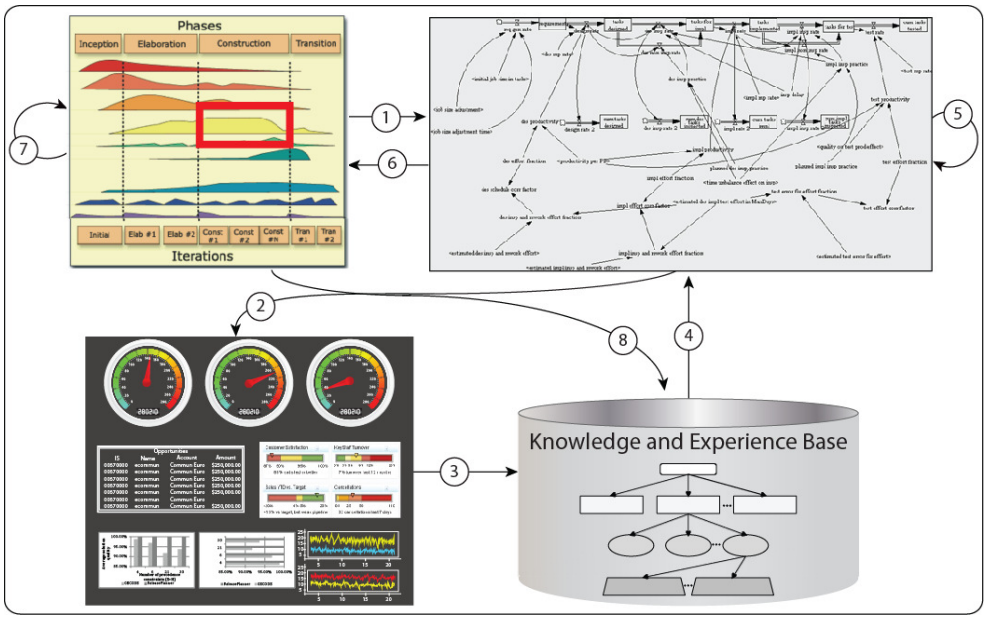

Figure 3. Workflow of SIM-DASH method.

knowledge and experience base (KEB) with a decision tree, the dashboard to provide the consequences of different simulation scenarios and recommendations of the proper action.

A. Modeling of RUP-Based Construction Phase Using SD

Considering that in larger projects several construction iterations take place, the main challenge is to manage the iterations in terms of producing operational capability [19]. In terms of manageable prototypes, the model aims to support the decisions during the construction phase of the large software development process.

An important input to the model is the stabilization policy, i.e. under what circumstances during the project efforts will be allocated for different tasks in programming or testing. For example, the effort assigned to programming may be allocated for programming the new feature versus the removal of injected defects, and there is similar policy for testing; the testing effort may be allocated for testing versus validation at each level of quality.

B. The Dashboard – A Management Tool to Monitor the Process

The dashboard is an interface to present the systematic view of project progress and proposes a number of options to deal with off-track situations. The simulation model is created by SD tool *Vensim* [22]. The parameters that can be updated and the desired outputs from the model are summarized in a visualized format in a project dashboard; this offers a user interface to track the actual versus targeted project states.

The sample dashboard created for the simulation model is shown in is shown in Figure 4. The dashboard provides two tabs; the first tab includes control variables (top). By changing these variables the simulation model is able to pro-actively evaluate the impact of changes of parameters. Three separate output graphs show the number of signed-off features in the three levels of quality Q0/1/2, respectively.

In the second tab, two slides, corresponding to programming and testing capacities, can define the value of these capacities. By modifying these variables, the number of outstanding features will be updated in the graphs below them. Finally, drilling down to more detailed information, the two graphs at the bottom of this tab provide information about the idle effort of different capacities.

The example dashboard provided Figure 4 looks at a specific sample project. The model is based on a real-world software development process, which was provided by the third author of the paper from the retrospective analysis and the perspective of a project manager.

Start and end dates of an iteration can be specified in the dashboard. After each cycle, the simulation model can be updated based on the reality, with parameters provided in the first tab of the dashboard. In each cycle, the actual performance is compared with the plan.

C. Implementation of the Knowledge and Experience Base with a Schematic Decision Tree

The content of the knowledge and experience base (KEB) is integrated and updated based upon the extracted information recording the different off-track situations. The corresponding action taken, their consequences and the cost needed, all this information is taken to update the KEB.

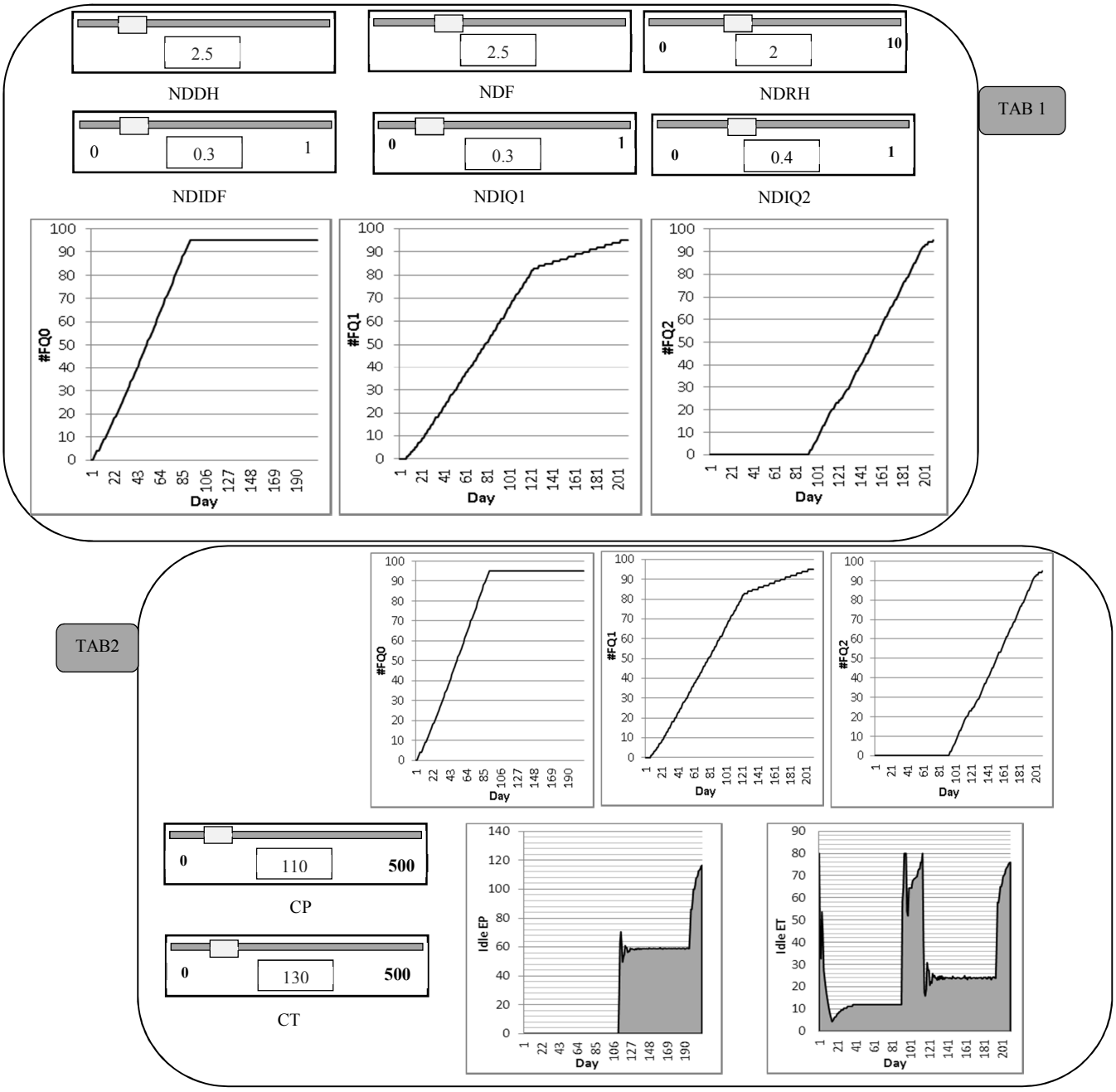

Figure 4. Dashboard provided based on a sample simulation run.

There are established methods, tools, and techniques proven to help the project manager to bring a project back on track. In general, there are two key factors related to the process of bringing a software project back on track. The first one is the cause and the second factor is the action to take in order to bring it back. Studies [23], [24] provide a list of factors occurring most frequently.

In terms of the cause, the variety of parameters of a development process can cause the off-track situation, which occurs when the progress criteria of the current situation do not meet the targeted values. Typically, it is a combination of multiple factors that cause a software project to fail [25].

Here, we look at factors related to an incorrect decision made by the project manager; such as improper estimation of the effort needed to complete a task, i.e. project underestimate, no incorporated risk factor in plan, parallelization of the sequential activities, poor scope change management, time conflict in task completion, or reduced productivity with complex tasks. On the other hand, there are a number of methods to prevent a failure, or to get an off-track project back in order. Such as re-allocating resources,

risk reassessments, checking for possible changes in dependencies, preventing the scope changes, etc.

Each technique assists in dealing with a specific cause. There can be number of methods to deal with a specific cause, but not all of them can be prescribed for an off-track project. The project constraints regarding budget, due date, required quality, and complexity determine the appropriate approach. Whenever a cause is detected the proper method to tackle the specified cause can be adopted; these (problem-solution) pairs constitute the content of the KEB.

The causes and the recovery actions can form a schematic decision tree, a correct path through the decision tree can be defined based on predefined constraints. A simplified view of a decision tree is presented in Figure 5. The first level presents root causes and the second level presents the options to tackle the cause. Each level of the tree is completed by a process inspired by the fishbone diagram [26] that identifies potential factors causing an overall result. This method breaks down (in successive layers of detail) root causes that potentially contribute to a particular effect, as we need here. Moreover, based on the constraints of the project, each leaf in the tree can be tagged with the cost of taking the corresponding path. Each leaf derives a number of methods to implement the action. For example, process improvement can be accomplished by choosing a different technique of testing or development.

To complete the methodology to assist in bringing the software project back on track, we use the decision tree and the SD model in conjunction, to demonstrate the efficiency of the selected action. To facilitate the comprehensive communication between two components of this methodology, a dashboard, representing the options provided by the SD model, is also introduced. An illustrative example from a real world project is presented in the next section to demonstrate how to employ the methodology and the results from the simulation model.

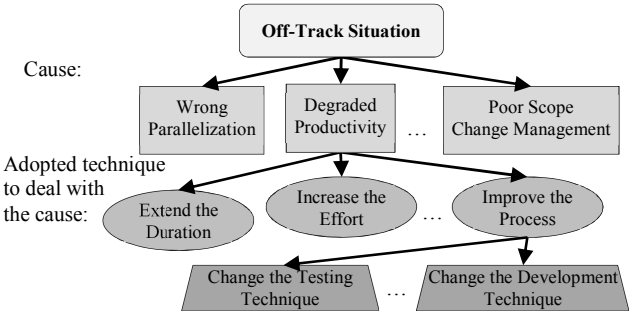

Figure 5. Decision tree for selecting the proper action for an off-track project.

V. APPLICATION

An example scenario presents an off-track situation, discovered at time t, when the gap between the target and the actual number of features signed off is detected. In this section we present this example according to the steps explained in the methodology.

A. Step 1: SD Modeling of RUP-Based Construction Phase.

In this real world project there are 17 modules, containing 95 features that must be signed off from the highest level of quality i.e. Q2 at the end of the project. The development is with Java, using web services for communicating with the backend SAP banking application. The project duration is seven months, with 11 programmers and 15 testers worked to complete it. The project budget is approximately 2.5 million dollars. In addition, the total function points are about 282 and the final software contained about 60K lines of code. The SD model, previously introduced, simulates the adopted process in this project. The construction phase of the project is considered as the input for the SD model. The example project with all the input variables is modeled as a SD model, and the results, in terms of the number of features signed off from quality levels Q0/1/2 at the end of the project, t = 210, based on the plan, are presented in Figure 6.

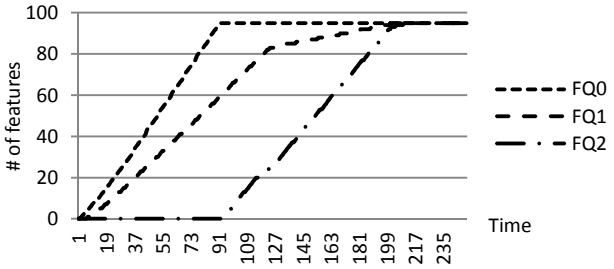

Figure 6. Target number of signed-off features from quality levels Q0/1/2 at the end of seven months.

Regarding the proposed methodology to bring a software project back on track, the simulation model is employed to show the results for the considered cycle.

B. Step 2: Detecting the Deviation between Target and Actual Performance Values and Measurement of Key Parameters of Actual Development Processes.

The completion of the project is modeled regarding the input variables, such as the considered capacity for programming and testing, constant variables such as the productivity of programmers in defect fixing, and the policies to assign the effort for each task in programming and testing. The SD model reflects the baseline situation at t = a is called *SD[P,a]* where P is the vector of all baseline parameter settings at t = a.

At the end of each cycle, considered as 30 days, the number of features in each quality level is compared with their values in the plan. The gap, Gi0/1/2, between the actual and target plan imply a deviation between target and actual performance values.

$$Gi0/1/2 = FQ0/1/2_{actual}(t) - FQ0/1/2_{target}(t) \quad (5)$$

The detected gap can be the result of altering the value of input parameters, which are predefined in the plan and implemented in the SD Model as *SD[P,a]*. Some of these

parameters, included in the dashboard, represent the productivity of programmers and testers, and are considered variables to be adjusted based on their actual values versus their predefined values in each cycle. The parameters are the components of the dashboard. The number of features signed off from each level of quality, are also considered as output parameters in the first tab.

The model is updated by adjusting the input parameters, which caused the gap in the previous cycle, C_{i-1}, in the dashboard. The number of features signed off can be predicted by the simulation model in the next cycle, C_i, denoted by *SD[P*,a]* in the SD model,. In other words, the SD model reflecting the changed situation at t = a, is denoted by *SD[P*,a]*, where P* is the vector of all changed parameter settings at t = a. In the example the off-track situation is detected at the end of the fourth cycle, $C_{i-1} = C_4 =$ [91, 120]. The deviation has been caused by degraded values of three parameters related to productivity of programmers and testers, and listed in Table II.

TABLE II. DEGRADATION PERCENTAGES OF THE DETECTED ROOT CAUSES

Parameter	% Degradation
Number of defects tested per hour	30%
Number of defects fixed per hour	20%
Number of defects retested per hour	25%

After defining the actual value of the parameters, they can be updated in the SD model using the components in the first tab of the dashboard. Adjusting these parameters, the behaviour of the model, in terms of feature sign off in the previous cycle is simulated. The continuation of the simulation for the next cycle, $C_{i*} = C_5 =$ [121, 150], predicts the number of features signed off from three quality levels, Q0/1/2. In Figure 7 the number of features signed off from quality levels Q0/1/2 is presented based on the planned and actual values.

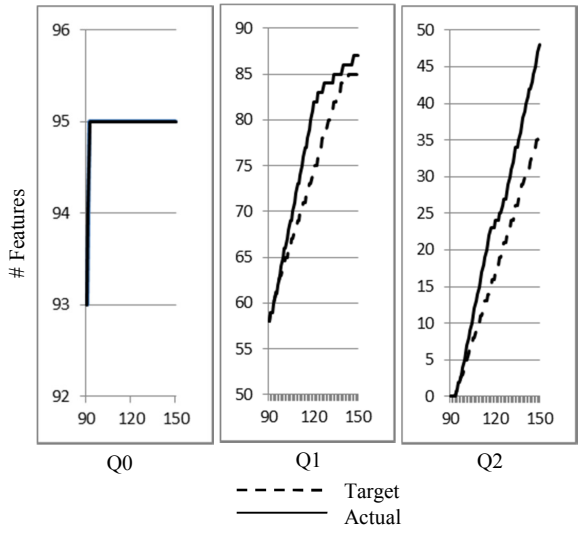

Figure 7. Deviation between target and actual values FQ0/1/2.

C. *Step 3: Request Actions to Bring the Project Back on Track Based on the Deviations Analyzed between Target and Actual Performance Values.*

In this scenario, the cause of the off track situation is stated as the degraded value of three productivity factors. This cause is included in the tree and the follow up methods to bring the project back on track, as a part of the KEB, are presented in Figure 8.

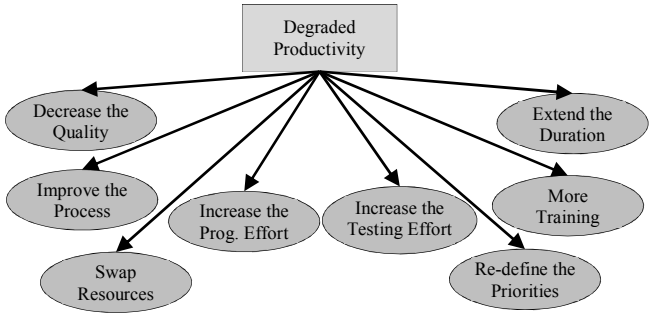

Figure 8. The cause and recommended methods to deal with it [17], [18].

D. *Step 4: Recommend Actions on How to Bring the Project Back on Track.*

The recommended actions from the KEB should be analysed based on project constraints. When the proper action as recommended by the KEB is determined, the choices regarding the implementation of options can be investigated with the SD model.

In this example, the recommended action from the KEB is to increase programing or testing efforts.

E. *Step 5: Run Simulation Scenarios to Pro-actively Evaluate the Impact of the Proposed Changes.*

To interact with the simulation model regarding the recommended action, the dashboard provides the related components in the second tab and also presents the output corresponding to options in the form of graphs. Pro-actively running the simulation scenarios assists evaluation of the options.

The results of the simulation model, in case of decreased productivity, show the availability of number of idle testers while there is no idle programmer during the fifth cycle, C_5. Although there are idle testers, their skill sets are different from the programmers and it is not possible for them to do programming. Therefore, the question here is narrowed to: "How many additional programmers are needed to reduce the gap between the plan and actual results?"

Assuming the application of a revised parameter setting P*, in C_5, the proposed action by the KEB is P*: Reallocate within *CP[a,b]* in $C_5 =$ [a, b] when a= 121 and b= 150, in order to minimize the gap between $FQ2_{target}(b)$ and $FQ2_{actual}(b)$. The application of a revised parameter setting P* is projected by the SD model as $FQ0/1/2_{projected}(b)$. We quantify the Projected Distance, *PD(P*,b)*, of different options regarding the cost of increased capacity for

programming. So, more details on minimization of the gap, regarding the FQ0/1/2 and the imposed cost of the taken action can be provided.

$$PD(P^*, b) = cost_2 * \Big(FQ2_{target}(b) - FQ2_{Projected}(b)\Big) + cost_1 * \big(FQ1_{target}(b) - FQ1_{Projected}(b)\big) + cost_0 * \big(FQ0_{target}(b) - FQ0_{Projected}(b)\big) \quad (6)$$

For the case study, the value of the cost0/1/2 is assumed to be 1, 4, and 2, respectively.

A significant drawback of increasing the capacity during execution of the software project is the impact of communication and training overheads on software development productivity as based on Brook's law [27]. It takes time for added developers to gain experience. Also, training requires communication between them; so, as Brooks' law states, "Adding people to a late project only makes it later". While new developers are added to the project, the SD model considers the training, communication overhead, and the assimilation rate parameters to be able to simulate the real world situation. Communication and training overheads result in decreased productivity. Another impact is that more people are available for development and the productivity will increase. It is difficult to draw a conclusion directly from only the qualitative analysis; the detailed influences need further analysis by simulation through the quantitative model.

In the SD model, Brook's law is implemented by considering impacting factors of training and communication overheads, increased capacity with less productivity, in addition to the assimilation rate of adding a new programmer to the group. The recommendation of increasing the programming effort introduces a number of simulation scenarios; these scenarios are derived from determining the proper amount of increased programming effort.

Running the pro-active simulation scenarios and evaluation of the results is accomplished by employing the dashboard. The component of increasing capacity for programming results in a different number of features, signed off from each quality level. The graphs in the second tab of the dashboard demonstrate the changes in these numbers associated with any increase.

In Table III the final number of signed-off features associated to each programmer added and the associated projected distance is presented. The original (baseline) number of developers is abbreviated by BL#P. The number of features signed off from quality level Q0 is excluded here, because it already reached 95 in the previous cycle.

The target and actual values of FQ1/2 are presented at the bottom of Table III. The results show that increasing the number of programmers by up to seven, positively decreases the distance of the target value. Adding more than seven programmers, the number of features FQ1/2 decreases and the reduction in quality level two is dramatic. Since no other changes are made in the simulation model, there are still idle testers waiting to complete their tasks, due to increased programmer's products (fixed defects for retesting and testing). The reduction of number of features is the result of the negative impact of increasing the capacity. The overhead of communication and training negate the power of added programmers to complete their tasks.

TABLE III. THE RESULTED FQ1/2 AT THE END OF THE FIFTH CYCLE FOR DIFFERENT SCENARIOS

# Progr.	FQ1 $_{Projected\ (150)}$	FQ2 $_{Projected\ (150)}$	PD(P*,150)
BL#P +10	84	28	52
BL#P +9	85	34	36
BL#P +8	85	38	28
BL#P +7	86	38	24
BL#P +6	86	38	24
BL#P +5	86	37	26
BL#P +4	85	37	30
BL#P +3	85	36	32
BL#P +2	85	36	32
BL#P +1	85	36	32
BL#P	85	36	N/A
Plan	87	48	N/A

F. Step 6: Analysis of Results and Provision of Recommended Actions.

After providing the evaluation results from the simulation scenarios recommendation can be made. In the example of an off-track situation, the criteria to provide recommendations is the minimal distance between the planned and actual performance, in terms of the number of features signed off from each level of quality, as presented in Table III, as the value of PD. Regarding the results provided in the previous step, the case of adding six or seven programmers minimizes the gap. Due to the extra cost of programmers, adding six programmers is the final recommendation as the proper action (case BL#P +7).

G. Step 7: Implementation of the Recommended Action in the Actual Project.

The recommendation about the implementation of the selected action is provided. In the next cycle this action will be taken to bring the project back on track.

H. Step 8: Learning: Add the Experience Gained from Applying the Recommended Action to the Existing Knowledge and Experience Base.

As one of the well-known goals of the SD model, learning is accomplished in this step. The recommended action and the results of applying it in the real world will be added to the KEB; this way the information integrated into the KEB is continuously updated. The KEB at this point is

conceptual. A successful experience management tool implementation is described in [28].

VI. Conclusion and Future Work

Software project management is a process that relies greatly on decision-making. In this paper, we have taken a system perspective by modeling the activities of RUP-based construction from a SD perspective. A hybrid approach is proposed, integrating dashboard functionality with simulation. The whole process of running pro-active 'what-if' scenarios is supervised by a knowledge and experience base, accommodating existing rules and lessons learned on how to react in case of unexpected situations. The approach is illustrated by a retrospective case study.

The benefit of the approach is to perform a pro-active analysis of the impact of the correcting actions; this is expected to increase the likelihood that the action taken helps to reduce the deviation from the target. While illustrated for re-allocation of effort, the approach is applicable to other measures, such as changing the process or technology, increasing productivity, and testing efficiency. As with any simulation-based work, the success is largely determined by the validity of the model; in our case, the model was validated in the context of a former SAP project.

The approach is intended for mature software organizations that have already established and explicit processes. Future research is needed to confirm the applicability for an ongoing project. In addition, further use case scenarios need to be studied to confirm the applicability and usefulness of the approach. In particular, we are planning to study the impact of changing the length of an iteration interval. Another direction of future research is to incorporate predictive models to dynamically update effort and defect estimates.

Acknowledgments

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grant 250343-07 and Collaboration Grant CRDPJ 358496-07. The comments of the reviewers have directed us to present the ideas of the papers more understandable.

References

[1] R. T. Futrell, D. F. Shafer, and L. Shafer, Quality software project management. Prentice Hall Professional, 2002.

[2] J. S. Aguilar-Ruiz, J. C. R. Santos, D. Rodríguez, and I. Ramos, "Generation of management rules through system dynamics and evolutionary computation," Proc. 4th Int. Conference on Product Focused Software Process Improvement, 2002, pp. 615-628.

[3] X. Meilong, L. Congdong, and J. Chen, "System dynamics simulation to support decision making in software development project," Proc. 4th International Conference of Networking and Mobile Computing on Wireless Communications, 2008, pp. 1-4.

[4] H. Zhang, B. Kitchenham, and D. Pfahl, "Software Process Simulation Modeling: An Extended Systematic Review," *New Modeling Concepts for Today's Software Processes*, vol. 6195, 2010, pp. 309-320.

[5] R. J. Madachy, *Software Process Dynamics*, 28th ed. Wiley-IEEE Press, 2008.

[6] M.I. Kellner, R.J. Madachy, and D.M. Raffo, "Software process simulation modeling:Why? what? how?," *Journal of Systems and Software*, vol. 46 (2/3), 1999, pp. 91–105.

[7] F. P. Jr. Brooks, The mythical of man-month, MA: Addison-Wesley Publishing, 1978.

[8] R. N. Charette, "Why Software Fails," *IEEE Spectrum*, vol. 42, 2005, pp. 36-43.

[9] C. Y. Lin, T. Abdel-Hamid, and J. S. Sherif, "Software-engineering process simulation model (SEPS)," *Journal of systems and software*, vol. 38 (3), 1997, pp. 263-277.

[10] D. Pfahl, G. Ruhe, K. Lebsanf, M. Stuppeerich, "Software process simulation with system dynamics - a tool for learning and decision support," *New Trtends in Software Process Modelling*, Series on Software Engineering and Knowledge Engineering, vol. 18, 2006, pp. 57-90.

[11] M. Wu and H. Yan, "Simulation in Software Engineering with System Dynamics: A Case Study," *Journal of Software*, vol. 4 (10), 2009.

[12] T. K. Abdel-Hamid, "Investigating the Cost/Schedule Trade-Off in Software Development," *IEEE Software.*, vol. 7 (1), 1990, pp. 97-105.

[13] T. K. Abdel-Hamid, "The dynamics of software project staffing: a system dynamics based simulation approach," *IEEE Transactions on Software Engineering*, vol. 15 (2), 1989, pp. 109-119.

[14] B. G. Ambrósio, J. L. Braga, and M. A. Resende-Filho, "Modeling and scenario simulation for decision support in management of requirements activities in software projects," *Journal of Software Maintenance and Evolution: Research and Practice*, 2010.

[15] B. W. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software*, vol. 8 (1), 1991, pp. 32-41.

[16] D. Rubenstein, Standish group report: There's less development chaos today. SD Times, http://www.sdtimes.com/article/story-20070301-01.html.

[17] R. L. Glass, Software Runaways. Prentice-Hall, New York, 1998.

[18] J. Verner, J. Sampson, and N. Cerpa, "What factors lead to software project failure?," Proc. 2nd International Conference on Research Challenges in Information Science, 2008, pp. 71-80.

[19] P. Kruchten, The rational unified process: An introduction. Addison-Wesley Professional, 2004.

[20] P. Rook, E. Rook, "Controlling software projects," *IEEE Software Engineering Journal*, vol. 1 (1), 1986, pp. 7-16.

[21] http://people.ucalgary.ca/~epaikari/SDModel

[22] Ventana Systems Corp., http://www.vensim.com/index.html.

[23] E.M. Bennatan, On Time Within Budget, John Wiley and Sons, 2000.

[24] L. A. Kappelman, R. Mckeeman, and L. Zhang, "Early warning signs of it project failure: the dominant dozen," *Information Systems Management*, vol. 23 (4), 2006, pp. 31–36.

[25] N. Cerpa, J. Verner , "Why Did Your Project Fail?," *Communications of the ACM* , vol. 52 (12), 2009.

[26] K. Ishikawa, (Translator: J. H. Loftus); Introduction to Quality Control; 448 p, 1990.

[27] W. Humphrey, "Managing the software process," Addison-Wesley: Reading, 1990.

[28] Seaman, C.B., Mendonça, M.G., Basili, V.R., Kim, Y.-M., "User interface evaluation and empirically-based evolution of a prototype experience management tool,,"*EEE Transactions on Software Engineering,* vol. 29 (9), 2003, pp. 838-850.