International Journal of Software Engineering
and Knowledge Engineering
Vol. 13, No. 5 (2003) 513–530

World Scientific
www.worldscientific.com

SUPPORTING SOFTWARE PROCESS DECISIONS USING BI-DIRECTIONAL SIMULATION

DAVID M. RAFFO* and SIRI-ON SETAMANIT†

*School of Business Administration,
†College of Engineering and Computer Science,
Portland State University, Portland, Oregon, USA
*davidr@sba.pdx.edu
†sirion@pdx.edu

In this paper, we present a "forward-looking" decision support framework that integrates timely metrics data with a simulation based defect model of the software development process in order to support software project management decisions regarding product quality. These predictions are evaluated using Outcome Based Control Limits (OBCLs) and Bi-Directional (both *forward* and *reverse*) simulation models of the software development process. The Bi-Directional models provide useful guidance to the program manager when evaluating possible corrective actions on a project that has gone outside the OBCLs. The *forward* model predicts the potential performance impacts of the proposed corrective actions. The *reverse* simulation models are new and take the desired system outcome as the starting point and represent the system as it evolves backward in time to identify the necessary starting point to achieve the desired outcome. The approach is presented using an illustrative example.

Keywords: Software process modeling; project management; simulation; software measurement repositories; multi-criteria decision making.

1. Introduction

In a previous work, Raffo *et al.* developed a quantitative forward-looking approach to support software project management *planning.* Specifically, this approach evaluates the impact of potential process changes by developing quantitative models of each process alternative — AS-IS and TO-BE scenarios. These models explicitly capture process-level details including interdependencies among process components. The models predict the impact of process changes *before* they are implemented. The approach has been applied to real-world process change problems at leading software development firms [14–16].

The goal of our current research is to develop another "forward-looking" approach that integrates timely metrics data with models of the software development

*Corresponding author.

process in order to support the software project management *control* function. This new approach provides predictions of project performance and the impact of various management decisions. The initial concept for this approach was first reported in [17] and focused on the novel role played by the flexible metrics repository. In this paper, we focus on using Outcome Based Control Limits (OBCLs) in conjunction with *forward* and *reverse* simulation models of the software development process. We define *forward* simulation as models of the system of interest as it evolves forward through time. We define *reverse* simulation as models of the system as it moves in reverse or backward in time. Using both types of models together we call them Bi-directional simulation.

The concept for modeling a system as it evolves backward in time is used in Dynamic Programming optimization [1]. In essence the *reverse* simulation model starts with the desired final state of the system as set by the Outcome Based Control Limits and works backward through time to identify the key intermediate states that need to be achieved by the system along the way in order to achieve the overall goal. When the system is not able to perform to a level that will achieve the desired outcome, it is considered "Out of Control" and corrective action must be taken.

The OBCL method is novel and is distinctly different from the traditional Statistical Process Control (SPC) paradigm. This is discussed in Sec. 2. Using *reverse* simulation is new to the software process field and has important implications. We discuss the Bi-directional simulation model in Sec. 3. We also discuss how the model can be used in conjunction with Outcome Based Control Limits. In Sec. 4, we show an example of using OBCLs in conjunction with both *forward* and *reverse* simulation models. The example uses a scenario and masked data from a real-world software development project.

2. Statistical Process Control Versus Outcome Based Control Limits

2.1. *SPC background*

Statistical Process Control (SPC) is a tool that was developed in the 1920s and 1930s by Shewart [18]. SPC was widely applied in the manufacturing industry by Edward Demming in the 1950s in Japan [4]. SPC techniques became widely adopted in the manufacturing sector in the United States in the 1980s. Some of the key benefits and advantages of SPC include:

(1) easy to use and understand;
(2) provides clear and straightforward guidance using simple graphical tools;
(3) clearly focuses on the problem of process variability and provides tools to help monitor the process;
(4) broadly applicable to many different kinds of production processes as well as other business areas including forecasting, inventory control and others.

Because of SPC's broad applicability, ease of use, straightforward guidance and direct focus on process variation, some users in the Software Engineering field

advocated that SPC be applied to Software Development projects [3, 5 and 6]. The concept of applying SPC to software development and the goals software practitioners hoped to achieve were clear — to reduce process variability which would enable better estimating. It was widely recognized that:

(1) results on software development projects can be highly variable;
(2) this variability directly contributes to the software crisis of cost and schedule overruns and poor product quality;
(3) improving the software development process should reduce project variation thereby making project performance more predictable and reliable (i.e. should bring software project performance back into controllable limits).

The picture of software project performance variability as a function of CMM Level (see Fig. 1) is a familiar one [13].

To this end, a number of researchers have worked to apply SPC tools and techniques to software development projects. Moreover, the application of SPC has been recommended in the CMM Level 4 KPAs [12]. However, in traditional SPC

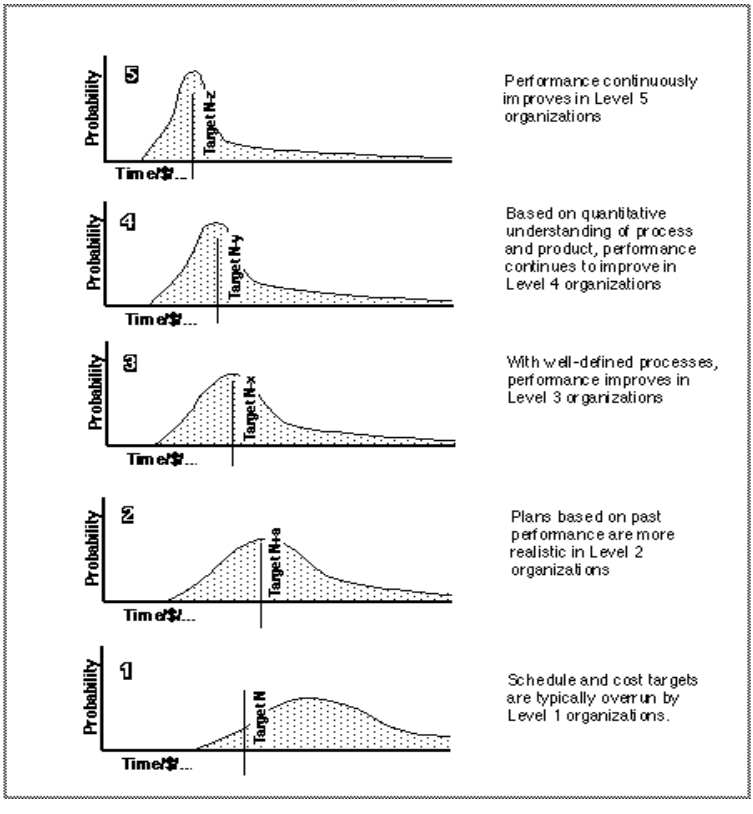

Fig. 1. Process variation by CMM level. (Taken from Paulk *et al.*, "Capability Maturity Model for Software, Version 1.1", SEI-93-TR-24, p. 23.)

analyses, there are some key assumptions about data that are used. If software project data do not comply with the key assumptions about SPC data, it is reasonable to question whether the traditional SPC approach can provide a real value in setting meaningful control limits that can be used to manage the project.

2.2. *What is SPC?*

Statistical Process Control (SPC) consists of a set of analytical tools and techniques that can be used to monitor a process [19]. The goal of SPC is to detect when a process is "out of control". This is accomplished by setting Control Limits that are based upon how the process has performed in the *past*. These *control limits* essentially create a confidence interval within which the measure being tracked is expected to fall under normal circumstances and are displayed in the context of a control chart (see Fig. 2). Control Charts are graphical tools that can be used to track many different kinds of data including manufacturing process data, sales forecasts, inventory levels, and so forth.

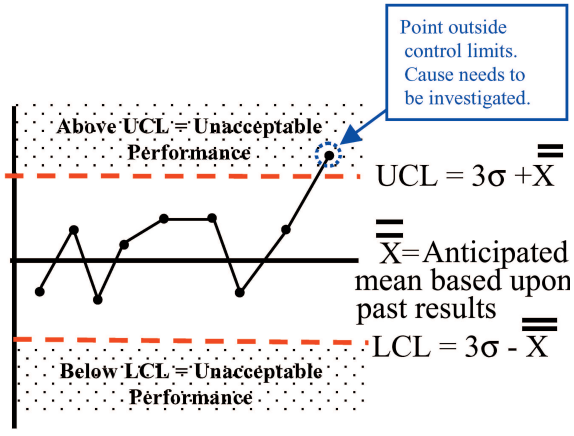

Fig. 2. Example of a control chart.

If the measure being tracked falls outside of the Upper or Lower Control Limits (UCL or LCL respectively), it must be investigated (as in Fig. 2). The reason is, in a formal sense, that the probability of the measure having a value that is outside the Control Limits under normal circumstances is extremely remote. Therefore, the probability is high that the cause of the measure being outside the control limits is due to a problem.

2.3. *The "trick" with SPC*

The "trick" with SPC is that the Control Limits must be set such that they provide a useful warning when the project is going off track. In the software development

domain, it is not uncommon to see empirical measurements where the standard deviation is between 5% and 100% of the mean

$$0.05 < \sigma/\mu < 1$$

This implies that traditional control limits on these data would range from being + or −15% of the mean to limits that range from 0 to as high as 400% of the mean. (Control limits are usually set at + or − three standard deviations. However, once 3σ is greater than 100% of the mean, the LCL side is truncated to be 0. In the case of $\sigma = \mu$ the limits will range from 0 to 400% of the mean.) The impact of that amount of variability on system level project outcomes is even greater.

Using traditional SPC, then, management would only be alerted when the values of the measure being tracked swung beyond the control limits. The problem is these limits may be too wide to be effective.

The essence of the problem is having to set meaningful limits. The cause of the problem is variability. The first step to attempt to reduce variability is to stratify the data. In typical SPC applications, data are stratified by operator, activity, and machine. In other words, data are tracked for each operator, doing a single task on one specific machine. Because in a typical manufacturing application the same operation is repeated many times, getting data is not a problem. However, in software development, the tasks, operators and tools may vary. As a result, these sources of variation are typically not controlled in software project data. Tasks are always different on some level. As a result, the ability to reduce process variation is limited. It is for this reason that meaningful control limits determined in a traditional manner can rarely be calculated for software development processes. Yet the need for having limits to alert management at the appropriate time remains.

2.4. *Outcome Based Control Limits (OBCLs)*

Because software projects can have high degrees of variability, managers need to get involved at the point when the project is going off track *regardless of whether the project is performing consistently.*

With Outcome Based Control Limits, management identifies the targets for project performance and acceptable ranges of performance for the overall project. By this we mean that management not only sets the values of the OBCLs but also the probability with which they must be satisfied.

If the project is able to perform to the desired level, it is defined to be "in control". If the project cannot perform to the expected level, it is then termed to be "out of control" and must be investigated with corrective action being a likely outcome.

With OBCLs, a model is used to map current performance (which is reflected in timely metrics data) to probable project outcomes. If the predicted performance deviates too much from the desired project outcomes, corrective action may be taken.

This is the key distinction between Outcome Based Control Limits (OBCLs) and SPC. SPC tracks the process based on where it has been. OBCLs track the process based on where it needs to go.

3. Combining OBCLs with Bi-Directional Simulation

3.1. *The scenario under consideration*

We illustrate the use of OBCLs with bi-directional simulation models using the following example where a software development firm is being contracted to do a 52 KLOC revision to an existing product. The situation described in this example was observed in the field at a company we are currently working with to apply these models. However, the data have been masked to protect company confidentiality. Hence, the results shown should be viewed primarily as an example to support illustration of the concepts presented in the paper.

The project employs a modified waterfall process with the life cycle phases of Requirements Specification (REQ), High Level Design (HLD), Low Level Design (LLD), Coding (CODE), Unit Test (UT), Functional Verification Test (FVT), and System Verification Test (SVT). Inspections of the Requirements Specification, High Level Design, Low Level Design, and Coding phases were conducted. Figure 3 also provides an overview of the Forward Simulation Model. In Fig. 3, we define INJ as the number of defects injected at each phase. ESC is the number of defects that escape to the next phase. CORR is the number of defects that are detected and corrected. INSP EFF is the inspection effectiveness or detection capability. It is the percentage of latent defects at each phase that are detected and corrected. TEST EFF is the test effectiveness or detection capability. It is the percentage of latent defects that are detected and corrected at each test phase.

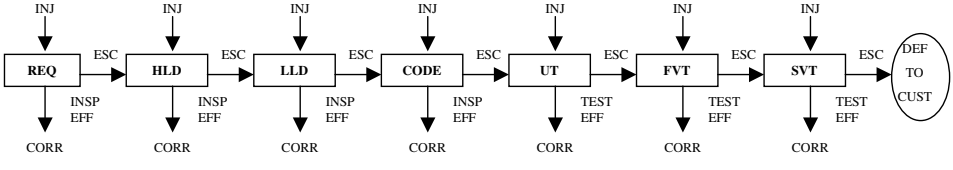

Fig. 3. Diagram of the company's process.

The model depicts the software development process used by the firm at the life cycle phase level. Actual project data were used for model parameters where possible. Specifically, actual defect injection and detection rates that were collected by the company were used in the model (see Table 1). The total number of defects injected into the product over the course of the development lifecycle was determined by the company to be 30 defects per KLOC with a standard deviation of 3 defects (10% of the mean).

The baseline simulation model of the lifecycle development was validated in a

Table 1. Defect injection and detection rates.

Life cycle phase	Injection rate (% total defects)	Detection capability
Requirements specification	12.3%	56.1%
High level design	15.2%	19.9%
Low level design	25.0%	23.6%
Coding	44.8%	17.2%
Unit test	1.5%	54.2%
Functional test	1.1%	72.5%
System test	0.2%	66.4%

number of ways. In particular, process diagrams, model inputs, and model parameters were reviewed by members of the software engineering process group as well as senior developers and managers for their fidelity to the actual (again, in this example, the metrics data have been masked). The model predicts product quality in terms of delivered defects to the customer. The results of the model were viewed as being reasonable. The model was developed using the Extend simulation language by Imagine That, Inc.

3.2. *Overview of the bi-directional model*

In this paper, we use two models. The first model is *forward* simulation model shown in Fig. 3. This model represents the system as it evolves forward through time. The *forward* model is used in conjunction with OBCLs to assess whether the system will achieve the desired level of performance. The second model is the *reverse* simulation model shown in Fig. 4. This model represents the system beginning with the desired end state of the system and simulates the process back to the starting point. The *reverse* simulation model essentially determines the "capacity" of the system. The purpose is to identify the required capacities of the intermediate process steps that enable the system to achieve its goals as defined by the OBCLs.

In Fig. 4, MAX ESC is the maximum allowable number of defects that can escape from the previous development phase as derived from the OBCLs and the expected inspection and test effectiveness levels. REQ CORR and REQ INSP EFF are the calculated levels of corrected errors and inspection effectiveness that are needed in order to achieve the OBCLs.

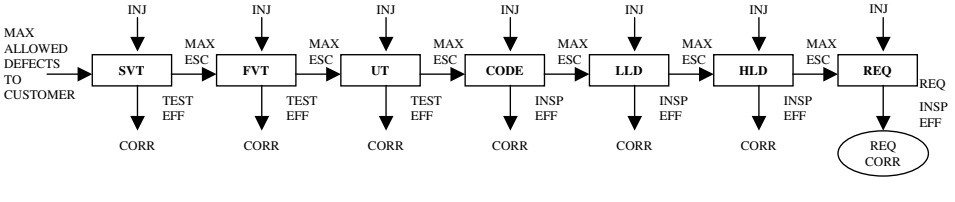

Fig. 4. Overview of the *reverse* simulation model.

The *reverse* simulation modeling concept is new to the area of software process simulation. *Forward* simulation models take a combination of complex inputs and use those inputs to predict *overall system performance*. We can use *forward* simulation to evaluate whether the predicted performance is within acceptable limits. We can also change *forward* model inputs and determine through a sequence of *trial and error* different values needed to bring system performance back on track.

In contrast, the *reverse* simulation model uses the *required system performance as input* to determine the capacity of the system at each phase. The *reverse* model simulates backward through time and determines the range of performance that must be obtained at intermediate phases of the process in order to achieve the overall system level standard as set by the OBCLs. In addition, the *reverse* model assesses the likelihood that those ranges will be achieved using the existing process. Although there are limits to this solution, *reverse* simulation takes an important step forward in offering specific guidance to the project manager. Specifically the *reverse* model can show a range of possible alternatives and required level of change. This enables the project manager to formulate a process improvement strategy by identifying productive change options early on and discarding quickly those options that will not achieve the desired effect. This is a powerful result.

The main limitation to the *reverse* simulation approach is that only one decision point can be evaluated each time. In other words, just as the *forward* simulation can predict overall system performance, the *reverse* simulation model can predict the performance required for one part of the system at a time. This means that when trying to identify the required performance for multiple parts of the system, all but one of the parts will need to be specified. Moreover, developing a *reverse* model for a system is non-trivial. It is like developing an inverse function. Because simulation formulations of a system are decomposed into stages, they are less complex than an analytic expression that would be used to represent the same system. As a result, the chances of creating a *reverse* simulation formulation of a system are greater than for analytic solutions.

3.3. *Description of the bi-directional model*

The main equations used for the *forward* simulation model are shown in (1) through (5):

(1) $\text{INJ\_DEF}_i = \text{INJ\_RT}_i\text{*DPK*KLOC}$
(2) $\text{DET\_DEF}_i = (\text{INJ\_DEF}_i + \text{ESC}_{i-1})\text{*INSP\_EFF}_i$
(3) $\text{ESC}_i = \text{INJ\_DEF}_i + \text{ESC}_{i-1} - \text{DET\_DEF}_i$
(4) $\text{ESC}_0 = 0$
(5) $\text{ESC}_{\text{SVT}} = \text{Defects Released to the Customer}$

The main equations used for the *reverse* simulation model are shown in (6) through (10):

(6) $\text{INJ\_DEF}_i = \text{INJ\_RT}_i\text{*DPK*KLOC}$

(7) $\text{DET\_DEF}_i = (\text{ESC}_{i-1})/(\text{1-INSP\_EFF}_i)^* \text{ INSP\_EFF}_i$
(8) $\text{ESC}_{i-1} = \text{DET\_DEF}_i + \text{ESC}_i - \text{INJ\_DEF}_i$
(9) $\text{INSP\_EFF}_* = \max(0, (\text{INJ\_DEF}_i - \text{ESC}_i)/ \text{INJ\_DEF}_i$
(10) $\text{ESC}_{\text{SVT}} = \text{OBCL}^*\text{DPK}^*\text{KLOC}$

where:

KLOC	The size of the project in thousands of lines of code.
DPK	The total number of defects injected into the software over the life of the project per KLOC for the example used in this paper, this value was 30 defects per KLOC with a standard deviation of 3 defects per KLOC.
INJ_RT_i	The percentage of total defects injected at phase i (see Table 1).
INJ_DEF_i	The number of defects injected at phase i.
ESC_i	The number of defects that escape detection at phase i.
INSP_EFF_i	The percentage of latent defects in the code at phase i that are detected and corrected (see Table 1). TEST_EFF can be used interchangeably for the appropriate phases.
DET_DEF_i	The number of defects that are detected and corrected at phase i.
INSP_EFF_*	The inspection or test effectiveness required to achieve the OBCLs at intermediate phase *.
ESC_{SVT}	In the *forward* model this is the number of defects released to the customer. In the *reverse* model this is the number of defects allowed to escape as set by the OBCL.

Some of the key input parameters used by the model are shown in Table 1. The above models would provide interesting results regarding the performance of the software development process. However, as specified, the above models assume that a defect injected during the Requirements Specification phase has the same value or cost as an average coding defect. A number of researchers have found that defects detected early in the development life cycle are more costly to repair and can cause other defects later in the development life cycle if not corrected [2, 10, 11]. To account for this, the models incorporate a multiplier for escaped defects entering the next phase. This multiplier was applied to defects from Requirements Specification to High Level Design, from Low Level Design to Coding. A lower multiplier was used between Low Level Design and High Level Design.

As a result, Eqs. (2) and (3) in the *forward* model for the phases mentioned were modified to be Eqs. (11) and (12) respectively.

(11) $\text{DET\_DEF}_i = (\text{INJ\_DEF}_i + \text{ESC}_{i-1}^* \text{MULT})^* \text{INSP\_EFF}_i$
(12) $\text{ESC}_i = \text{INJ\_DEF}_i + \text{ESC}_{i-1}^* \text{MULT} - \text{DET\_DEF}_i$

Equation (8) in the *reverse* model changed as well to Eq. (13)

(13) $\text{ESC}_{i-1} = (\text{DET\_DEF}_i + \text{ESC}_i - \text{INJ\_DEF}_i)/\text{MULT}$

Table 2. Defect multiplier values.

Phases where multiplier was used	MULT
Between REQ and HLD	4
Between HLD and LLD	2
Between LLD and CODE	4

The multipliers used are shown in Table 2. The multiplier values should be calibrated for each organization.

3.4. *Using OBCLs with bi-directional simulation*

The goal of combining OBCLs with Bi-directional simulation is to support project management ***planning*** and ***control*** decisions. As a first step, management sets OBCLs for each performance measure they would like to track. Next, models are developed to predict the process performance along the performance measure(s) of interest. In this paper, we present a simulation model that predicts product quality as defined by the number of defects delivered to the customer. Models for effort or schedule could also be developed. It is not required that the model be a stochastic simulation model. We have used a stochastic model because:

- Stochastic simulation models can capture the inherent uncertainty associated with software development processes
- Simulation models do not impose restrictive assumptions on the distributions used and model structure that analytical models do in order achieve a closed-form solution.

The *forward* simulation model is used to predict overall system performance. This is compared to the OBCLs. The *reverse* model takes the performance target and the OBCLs as its initial input. The *reverse* model then predicts the feasible range of performance for intermediate stages of the process. If the performance of the system falls within the limits set by the OBCLs using the *forward* model, no action is taken. However, if system performance is predicted to be outside the desirable limits, it is defined to be "Out of Control", the reasons are investigated and corrective action may be necessary.

When the system is "Out of Control", the *reverse* simulation model is used to identify at what *points* in the process the system performance fails to achieve the desired outcomes. We say *points* in the process because often there are multiple points of control that management can modify throughout the process to reach an overall desired outcome.

4. Analysis

4.1. *Setting OBCLs and getting baseline results*

In this example, management has set a goal of having no more than 3.25% of the injected defects released to the customer no more than 25% of the time (i.e., 3.25%

Table 3. Outcome based control limits.

	OBCL (less than 75% of the time)	Baseline expected
Percentage of defects released to customer	3.25%	2.89%
Number of defects to be released	50.70	45.10
Number of defects to be released (using Phase Multiplier Model)	188.8	167.9

Table 4. *Forward* simulation results for the baseline model.

	Mean	STD DEV	Coef. of Var.
Number of defects delivered to the customer	45.10	8.29	0.184
Number of defects delivered to the customer (using Phase Multipliers)	168.56	30.44	0.181
Probability of Achieving OBCL with current process (must be at least 75% in order to meet OBCLs)	74.69%		

or less of the total injected errors can be passed to the customer at least 75% of the time). Given an average overall defect injection rate of 30 defects/KLOC, this resulted in a maximum allowable number of defects to be released of 50.7. Using the phase multipliers from Table 2, the OBCL for the process was 188.8 (see Table 3).

With the OBCLs set, the next step is to use the *forward* simulation model to predict the performance of the baseline process. These results are shown in Table 4. As can be seen, the baseline process is performing to the OBCLs. The predicted average number of defects being delivered to the customer is 45.10 with a standard deviation of 8.29 defects. This results in 50.70 defects (the OBCL) being delivered to the customer no more than 25.31% of the time which, although slightly over the OBCL, was deemed acceptable (the Appendix shows an example of calculating the probability of achieving the OBCLs).

Using the *reverse* simulation model, other statistics were predicted. The *reverse* simulation model takes the desired level of performance as its initial input. Then, the model can be used to predict the probability that any phase will achieve the required minimum performance necessary in order to achieve the OBCL. At the current point in this example, the project has not yet started, we use the *reverse* simulation model to predict the required performance of the each of the defect detection phases assuming all other phases are performing as predicted. The results are shown in Tables 5 and 6. Table 5 shows the distribution of inspection and test effectiveness levels required at each phase of the process (assuming that the other phases are performing as expected). Thus, Requirements Specification Inspection would only need to perform at a 36.7% effectiveness level in order for the OBCLs to be achieved. However, all of the "slack" would be taken out of the system at that point. This would require all subsequent phases to perform at or above expectations

Table 5. Inspection and test effectiveness required to meet OBCLs.

Life cycle process phase	Inspection and test effectiveness by phase Mean	Std. Dev.
Requirements specification	36.7%	26.7%
High level design	−1.8%	31.2%
Low level design	8.1%	20.4%
Coding	3.0%	19.8%
Unit test	46.6%	9.7%
Function test	68.6%	4.2%
System test	61.5%	6.3%

Assumes initial expected values for other phases

Table 6. Probability system will meet OBCLs at specified level of inspection and test effectiveness.

	Inspection and test effectiveness 0%	25%	50%	75%	100%	Expected probability efficiency	Expected inspection and test efficiency by phase
Requirements specification	8.5%	33.1%	69.1%	92.4%	99.1%	76.6%	56.1%
High level design	52.3%	80.5%	95.2%	99.3%	99.9%	75.7%	19.9%
Low level design	34.6%	79.6%	98.0%	99.9%	100.0%	77.6%	23.6%
Coding	44.0%	86.7%	99.1%	100.0%	100.0%	76.4%	17.3%
Unit test	0.0%	1.3%	63.7%	99.8%	100.0%	78.0%	54.1%
Function test	0.0%	0.0%	0.0%	93.6%	100.0%	82.5%	72.5%
System test	0.0%	0.0%	3.4%	98.4%	100.0%	77.9%	66.4%

in order to achieve the OBCLs. Looking at the next row, it appears that High Level Design Inspection could be eliminated and still enable the OBCLs to be achieved. This is an interesting finding. It shows that the HLD Inspection may be a step that merits further examination given the ***structure*** of the process (as the subsequent analysis indicates). We define the structure of the process to include not only the collection of process steps and their sequence, but also the relative performance of each process step in the context of the entire project.

Table 6 shows the implications of the results in Table 5 and is quite interesting. For the Requirements Specification Phase, we can see that if management eliminates the REQ inspection (0% inspection effectiveness), the project would have an 8.5% chance of achieving the OBCLs. If the REQ inspection detects 25% of the latent defects, the project would have a 33.1% chance of achieving the OBCLs and so forth. The last two columns of Table 6 refer to the expected inspection efficiency of each step and can be read as follows: The expected inspection efficiency of Requirements Specification is 56.1% (from Table 1). When Requirements Specification achieves the expected inspection efficiency, the *reverse* model predicts that the project has approximately a 76.6% chance of achieving the OBCLs. These results make sense in that performing as expected enables the project to achieve the

OBCLs. Performing less than expected reduces the project's chances. The *reverse* model provides the results of Table 5 which enable us to approximately determine by how much. A comparison between the numbers reported in Tables 5 and 6 do not match exactly because the Inspection and Test Effectiveness values reported in Table 5 are distributions reported from the simulation model, whereas the numbers reported in Table 6 are based upon point values for effectiveness levels that were input to the model.

For High Level Design, the interpretation is similar and also of interest. Assuming REQ inspections perform according to expectations, management could eliminate HLD inspections and still have approximately a 52.3% chance of achieving the projects OBCLs. As can be seen, HLD inspections are not expected to be a very effective means of removing defects for this project team anyway (expected performance is only 19.9%). Similarly, if HLD inspections were to be 25% effective, it would boost the chances of achieving the OBCLs to 80.5%. This is above the minimum standard of 75% set by management.

Defect detection effectiveness appears to become more strict at the testing phases. At Functional Test and System test, it is essential that Test Effectiveness be above 50% in order to have any hope of meeting the OBCLs. Moreover, by performing at 75% effectiveness (which is a small improvement over the expected level for Functional Test), appears to significantly boost the chances of meeting the OBCL targets. This makes sense that the effectiveness of the last few quality assurance phases do have a great impact on the quality of the product delivered to the customer.

The results in Table 6 give added evidence that the project can meet the OBCLs set by management and confirm the results in Table 4. If the defect detection steps perform below expectations, it is likely that the chances for achieving the OBCLs will be below the minimum requirement. If the performance is above expectations, chances improve. The project is ready to begin execution. The models now switch from planning mode to being used to monitor and track the project.

4.2. *Using OBCLs and bi-directional simulation to support project management control*

The previous subsection addresses the issue of software project management ***planning***. Using this forward-looking approach, we are also interested in using OBCLs to support project management during the execution of the project (software project management ***control***).

At issue: What is the impact if the project does not achieve the expected level of inspection effectiveness? Would the project still be able to achieve its OBCLs? What impact would the below par performance have on the remaining inspections and test phases? Will these defect detection processes need to be improved?

If the project is successful in achieving its objectives at each project milestone, the model shows that the OBCLs set by management will be achieved. On the

Table 7. Process performance with 92 defects detected at requirements specification.

	Mean	STD DEV
Number of defects delivered to the customer	179.1	33.9
Probability that 188.8 defects or less will be delivered to the customer	61.3%	

other hand, if problems do occur, the Bi-directional simulation models can be used together to (1) alert management to the potential problem early on and (2) plan strategies for potential corrective action.

As the project executes, new information is made available. Actual values observed from the project replace estimated values in the model. This is accomplished through timely metrics that are provided by corporate information systems or through a universal metrics repository [7]. In this example, we now move forward in time to the point when the project has executed through Requirements Specification. For purposes of this example, we suppose that 92 defects were detected during Requirements Specification inspections. If we assume that the usual number of defects are injected, this means that Requirements Specification inspections would be 47.9% effective. Using traditional SPC models, this level of performance would be within the SPC limits and no action would be taken. Using OBCLs and the *forward* model, we can see the expected performance of the project is 179.1 defects with a standard deviation of 33.9. As a result, we now have only a 61.3% chance of achieving the desired level of performance which is too low (Table 7). The probability that the project will deliver too many defects is too high. Action needs to be taken. Using the *reverse* simulation models, we can identify potential points in the project where improvements can be made and we can see the potential impact. This is discussed below. Again, the main point is that using traditional SPC, the results of Requirements Specification on the project would have seemed fine. Using OBCLs combined with *forward* simulation we see that management actually needs to get involved and take corrective action before the project gets too far off track. Using the *reverse* simulation models, we are able to identify potentially fruitful areas for improvement and see the potential impact. Using the *forward* simulation model again, we reconfirm the impact and take action.

Table 8 shows the impact of this poorer than expected defect detection effectiveness on the number of defects that will be delivered to the customer and the ability of the project to now meet its OBCL goals.

Given the performance level reported in Tables 7 and 8, the question becomes: What can be done to bring the project back within the OBCLs. At this point, many different process alternatives may be evaluated. Management has the ability to modify the process at multiple points. Should improvements be made at only one phase or all along the process?

The Bi-directional simulation models identify not only when the project is "Out

Table 8. Probability system will meet OBCL at specified level of inspection and test effectiveness (REQ Defected = 92).

	Inspection and test effectiveness					Expected efficiency probability	Expected inspection and test efficiency by phase
	0%	25%	50%	75%	100%		
High level design	39.6%	71.6%	92.0%	98.8%	99.9%	65.6%	19.9%
Low level design	21.6%	68.8%	96.1%	99.9%	100.0%	66.2%	23.6%
Coding	30.7%	81.1%	98.8%	100.0%	100.0%	67.5%	17.3%
Unit test	0.0%	0.7%	51.2%	99.4%	100.0%	67.0%	54.1%
Function test	0.0%	0.0%	0.0%	87.5%	100.0%	70.2%	72.5%
System test	0.0%	0.0%	1.7%	96.7%	100.0%	68.0%	66.4%

Table 9. Probability project will achieve OBCL if average inspection effectiveness is 35%.

	Probability project will achieve OBCLs
High level design	85.87%
Low level design	87.38%
Coding	96.32%

of Control", but also provide an indication of what magnitude of improvement needs to be made in order to bring the project back on track.

As can be seen in Table 8, the expected inspection effectiveness for HLD, LLD and CODE on this project is low to achieve the OBCLs. Using the *reverse* simulation model, we can see that even bringing the performance of these inspections up to the 25% effectiveness level seems to offer an improvement. This level of improvement is very realistic as inspection effectiveness levels reported in the literature often exceed 35% [9]. Table 9 shows the potential impact of improving any of these inspections to the 35% effectiveness level.

Determining the best alternative process requires a full business case taking into account the costs and benefits of each process alternative which is beyond the scope of this paper. Models of development costs and project duration would also need to be included. The focus of this paper is to introduce the new concept of Bi-directional simulation and its integration with Outcome Based Control Limits to support both Software Project Management Planning and Control.

The *forward* simulation model can be used to experiment and examine a variety of process alternative combinations as well as to predict the impact of the proposed corrective action on project performance.

5. Conclusions

Software development is a challenging and complex endeavor. When developing a product, many decisions need to be made while managing the development of a

project. In this paper, we present a new approach that combines the use of Outcome Based Control Limits with Bi-directional simulation models of the software development process to support decisions about Software Project Management *Planning* and *Control*. The software development process is represented using both "*forward*" and "*reverse*" discrete event simulation models. Using these two models coupled with Outcome Based Control Limits enable management to:

(1) Recognize when a project is going off track;
(2) Provides guidance regarding what points in the process may need to be improved and by how much; and
(3) Enables managers to test out combinations of process alternatives to see if they will solve the problem.

This approach supports the ability to quantitatively monitor and assess software projects. As a result, this approach supports the Quantitative Process Management and Software Quality Management Level 4 KPAs of the CMM. This also addresses Level 5 KPAs of the CMM related to Continuous Process Improvement [9, 12].

References

1. D. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice Hall, Englewood Cliffs, NJ, 1987.
2. B. W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.
3. D. Card, "Statistical Process Control for Software?" *IEEE Software* **11**(3) (1994) 95–97.
4. W. E. Demmings, *Out of the Crisis*, MIT Press, 1986.
5. Florac and Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison-Wesley, Reading, MA, 1999.
6. Florac, Park, and Carleton, "Practical Software Measurement: Measuring for Process Management and Improvement", Technical Report CMU/SEI-97-HB-003, Software Engineering Institute, Carnegie Mellon University, 1997.
7. W. Harrison, "A universal metrics repository", in *Proc. Pacific Northwest Software Quality Conference*, Portland, Oregon, October 18–19, 2000.
8. W. Humphrey, *Managing the Software Process*, Addison Wesley, 1989.
9. C. Jones, *Applied Software Measurement*, 2nd Edition, McGraw-Hill, 1996.
10. S. H. Kan, *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995
11. S. T. Knox, "Modeling the cost of software quality", *Digital Technical Journal* **5**(4) (1993) 9–16.
12. M. C. Paulk, W. Curtis, M. B. Chrissis, and C. V. Weber, "Capability Maturity Model for Software, Version 1.1", Technical Report SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, February 1993.
13. M. C. Paulk, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995, pp. 28.
14. D. M. Raffo, "Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance", Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, UMI Dissertation Services #9622438, 1996.

15. D. M. Raffo and M. I. Kellner, "Field study results using the process tradeoff analysis method", in *Proc. 1996 Software Engineering Process Group Conference*, Atlantic City, New Jersey, May 20–23, 1996.
16. D. M. Raffo, J. V. Vandeville, and R. Martin, "Software process simulation to achieve higher CMM levels", *Journal of Systems and Software* **46**(2/3) (15 April 1999).
17. D. M. Raffo, W. Harrison, and J. V. Vandeville, "Coordinating models and metrics to manage software projects", *Int. J. Software Process Improvement and Practice* **5**(2/3) (2000) 147–157.
18. W. Shewhart, *Economic Control of Quality of Manufactured Product*, Van Nostrand Reinhold, New York, 1931.
19. D. Summers, *Quality*, 2nd edn., Prentice Hall, 2000.

Appendix — Example for Calculating the Probability of Achieving OBCLs

To compute the probability of achieving OCBLs, we use the distribution of the selected performance measure. In the application presented in this paper, the performance measure prediction is normally distributed. In general, output performance measures from discrete event simulations are usually normally distributed regardless of the input distributions because the simulation results are based upon the summation of many random variables and the result follows the Central Limit Theorem. A unique thing about the application in this paper is that the OBCL is only an upper bound (i.e. delivered defects must be less than or equal to 3.25% at least 75% of the time.) In most SPC applications, the measure of interest must be neither too high nor too low. As a result, there are both upper and lower control limits. However, in the application presented in this paper, we only have an upper control limit.

An Example

Let the distribution for the Number of Remaining Defects (NRDs) be:

$\mu = 168.56$ defects

$\sigma = 30.44$

as shown in Table 4. From Table 3, we observe that the OBCL performance states that the number of delivered defects can be no more than 188.8. Now we must calculate the probability that the distribution will result in a value of 188.8 or less.

We denote the mean and standard deviation of the distribution of NRDs as μ and σ. We denote the Outcome Based Control Limit as an upper bound and set the upper control limit (UCL) equal to it. We compute the following:

(14) $Z_U = (\text{UCL} - \mu)/\sigma$

(15) $\text{PROB} = N(Z_U)$

where:

Z_U is the number of standard deviations the UCL is away from the mean of the performance measure.

PROB is the probability that the observed value of the selected performance measure will be within the OBCL.

Once the Z score is determined we may use a table that maps Z scores to normal distribution probabilities. Also, MS Excel has a statistical function called NORMDIST and is used as follows:

$Z_U = (188.8 - 168.56)/30.44 = 0.6649$

PROB = NORMDIST(0.6649) = 74.69%

as shown in row 3 of Table 4.