Using Design of Experiments, Sensitivity Analysis, and Hybrid Simulation to Evaluate Changes to a Software Development Process: A Case Study

Wayne Wakeland
Systems Science Ph.D. Program, Portland State University
wakeland@pdx.edu

Robert H. Martin
Software Management Consulting
bobm4@teleport.com

David Raffo
School of Business Administration, Portland State University
davidr@sba.pdx.edu

Abstract

Hybrid simulation models combine the high-level project issues of System Dynamics models along with the detailed process representation of discrete event simulation models. Hybrid models not only capture the best of both of these simulation paradigms, but they also are able to address new issues that are important in managing complex real-world development projects that neither the System Dynamics nor Discrete Event simulation paradigms are able to address alone.

In order to reap the full benefits from a simulation model, a structured approach for analyzing model results is necessary. The recommended approach is a combination of the Design of Experiments (DOE) technique and sensitivity analysis performed in a specific manner. DOE is a statistical technique that provides a more objective measure of how the impact of a given change to the model (such as a process change) might be dependent upon the values of other model parameters (such as the project environment, worker motivation, schedule pressure and so forth). Consideration of the interaction effects coupled with sensitivity analysis is essential for insightful interpretation of model results and effective decision-making.

This paper applies DOE and broad range sensitivity analysis to a Hybrid System Dynamics and discrete event simulation model of a software development process. DOE is used to analyze the interaction effects, such as the degree to which the impact of the process change depends on worker motivation, schedule pressure and other project environmental variables. The sensitivity of the model to parameter changes over a broad range of plausible values is used to analyze the nonlinear aspects of the model. The end result is a deeper insight into the conditions under which the process change will succeed and improved recommendations for process change design and implementation.

Keywords: Software Process Modeling, Software Process Simulation, Hybrid Simulation, Design of Experiments, Sensitivity Analysis

1. INTRODUCTION

It is well understood that managing a software development project is a multi-faceted problem that seems to transcend the complexity of normal project management. Initially a manager must supply cost, schedule and quality estimates using incomplete information about requirements and resources. The manager must decide the scope of the work, the project methodology, the schedule and the staff. She must anticipate the impact of environmental factors such as staff experience, requirements stability, corporate culture and development environments.

To further complicate the problem, most of these factors are dynamic and will change throughout the project. Many of the factors interact with each other. Experienced managers may develop an intuition within the domain of their experience, but changes in technology, such as the move to web-based applications, can challenge the validity of that intuition.

These complications increase the potential value of using computer models. Models of the development process and of the development environment represent a concrete expression of a manager's understanding of the project. Models also provide a more complete and more precise description of the various assumptions that previously may have been only partially understood in an intuitive sense. Simulation of project behavior over time allows the manager to examine the effect of policy alternatives given different assumptions about the parameter values.

In this paper, we present a hybrid simulation model of a software development project. Hybrid models integrate the high-level project issues of System Dynamics models along with the detailed process representation of Discrete Event simulation models into one model. As a result, Hybrid models are able to capture a number of important aspects of the software development environment and address how these factors impact the overall performance of the process and development effort.

We examine the Hybrid model using the well-known Design of Experiments (DOE) methodology in conjunction with sensitivity analysis conducted over a broad range of values for the selected input parameters. DOE is a statistical method that can be used to examine complex simulation model behavior in order to see how changes in parameter values lead to different effects depending upon how system components interact with each other. Sensitivity analysis complements DOE by helping to reveal non-linear effects. As a result, one can see how a given process change may be beneficial to a project under certain project environmental conditions, while it would not be beneficial under others.

Although showing the impact of parameter changes is the goal of sensitivity analysis in general, broad range sensitivity analysis (BRSA) reveals possible non-linearities that may not be detected using traditional sensitivity analysis in which parameters are varied systematically by a small fixed percentage.

Hybrid simulation models tend to be particularly complicated and non-linear, which increases the potential value of using DOE and BRSA to help uncover the implications of the complex behavior generated by such models.

This paper opens by describing general simulation paradigms, and a particular hybrid simulation model for a software development process. Next, we describe DOE and BRSA, and discuss their application to the analysis of hybrid simulation models. Finally, we provide a case study that illustrates the application of DOE and BRSA to a hybrid model of a software development process.

2. BACKGROUND AND LITERATURE REVIEW

In this section, we review three simulation paradigms that have been applied to software development processes and identify some of the types of questions that can be difficult to answer using these paradigms. Next, we describe Hybrid models with a focus on the special features and characteristics that enable it to address questions beyond the other paradigms. Finally, we provide an overview of DOE and BRSA, and describe how this combination can be used to help address important issues in software development simulation.

2.1 Common Simulation Paradigms Used to Aid Software Process Management

System Dynamics

System dynamics models capture the dynamic behavior of project factors and their interactions. Since the interactions may dominate the project behavior, these models provide a way to examine the sensitivity of the project to critical values in the factors. Abdel-Hamid and Madnick provided a sophisticated example of a model of the software development process. [1]

A key aspect of system dynamics (SD) models is the set of state variables that change dynamically over time. While this framework can be used to represent process activities (for example, the variable "Design" might represent the amount of code that has been designed at a given time), the paradigm does not directly or easily represent entities such as code modules. Thus, there is no opportunity to attach attributes such as complexity to the entity. A great deal of work has been done using System Dynamics models in the software process simulation community. Lehman and Ramil [14], Powell et al. [20], Wernick and Lehman [24], Madachy [15], Tvedt [23], among others have developed complex models to predict project performance issues ranging from software evolution to concurrent project performance to inspection effectiveness. While SD models have been built to examine questions regarding process activities [16] [23], these models are inherently limited by the underlying mathematical engine (first-order ordinary difference equations in time).

State Based Models

State based models directly represent process activities and model the process dynamics through state transitions triggered by events. The design completion may trigger both the start of the coding activity and the start of unit test plan development. A state based model easily represents parallel activities, which makes it attractive for examining questions about process bottlenecks. Equations of these models are often similar to those found in discrete event simulation models in SW Process Applications. Raffo used a state-based simulation to predict model time, cost and quality in a detailed applied model [21][22]. As the name implies, these models are based on state charts.

Since the model time advances only when events occur, updates to model dynamic project factors must be tied to the occurrence of events. This limits the ability of the state-based models to effectively represent feedback loops.

Discrete Event Simulation
Discrete models represent the development process as a series of entities flowing through a sequence of activities. Discrete models easily represent queues and can delay processing at an activity if resources are not available. Each entity may be described by unique "attributes". Changes to the attributes by the activities can provide much of the value of a discrete model. The effort or duration of each activity may be sampled from random distributions allowing the model to represent the uncertainty that exists in the process. This allows the simulation to capture the effects of variation in the entities (such as size, complexity, number of defects, defect type and so forth) on each activity. Discrete models can capture the interdependence that occurs between activities in a project. Activities in a development process may be delayed when a programmer is diverted to another task. Testing may be delayed until a test bed is released. If a model can capture these dependencies at a sufficiently detailed level, it may show ways to alter the process to reduce risk or increase efficiency.

Since discrete models advance time only when an event occurs, continuously changing variables can only be updated at specific event times. While the time between discrete events may be days or weeks in a software project, SD models containing continuous variables may require a time step in hours. This difference can cause errors in the integration of the continuous variables or may create instability in the behavior of feedback loops if a DES model is used for this purpose. A number of DES models have been developed in the software process domain. These include Raffo et al [22], Host et al [9] and Donzelli and Iazeolla [5].

2.2 Hybrid Models

Hybrid models [17][18] employ a more complex simulation technique than either System Dynamics models or Discrete Event models in order to avoid the limitations described with the other modeling paradigms. By combining discrete event and continuous simulation, hybrid models can represent entities with attributes being acted upon by activities that are influenced by continuously changing factors. The state changes that are directly represented in state-based models are implied through the action of entities in activities. Parallel activities are represented by additional virtual entities. Consequently, hybrid models enable one to examine the effects of changes in processing logic, within the context of a dynamic project environment.

Since Hybrid models treat work packages and project resources as discrete entities, as is typically done in discrete event simulation (DES) models, Hybrid models are able to accurately reflect the actual status of an item: designed, awaiting rework, awaiting inspection, and so forth. This allows the modeler to easily examine alternative policies for assigning resources to different classes of work: coding, testing, rework, and others depending on the amount of backlog present, for example.

In Hybrid models, parameters such as error rates and resource productivity vary dynamically over time, depending on schedule pressure and other factors, just as they do in System Dynamics (SD) models. Thus, it is possible to examine questions about staffing and overtime policies, and to explore different assumptions about worker fatigue and burnout.

Obviously, not all real-world questions fall neatly into categories such that one would naturally use either a DES model or a SD model. Many real-world questions would be better answered using a Hybrid model that combines both paradigms. Policies for assigning resources to different classes of work for example are likely to depend upon staffing and overtime policies. It is often worker fatigue and burnout that cause backlogs of work packages to develop. Without a Hybrid model, the analyst must attempt to approximate the relevant effects solely within the DES or SD paradigm.

While a Hybrid model can provide the most direct and complete representation of a software development project, Hybrid models also require a more sophisticated simulation engine. We developed the needed capability by modifying a commercial simulation package, Extend ™[6]. Although the standard Extend software does permit the user to create hybrid models, it is not computationally efficient when used this way. Modifications to the EXTEND simulation engine were needed to accurately and rapidly simulate combined continuous and discrete models. The resultant run times were on the order of a minute to simulate a single replication of a two-year project with a time step of one day. This level of performance is necessary for practical use of DOE and BRSA.

2.3 Hybrid Model Utilized for the Case Study

In this paper, we present a Hybrid model that combines the well-known System Dynamics model developed by Abdel-Hamid and Madnick [1] with a discrete model of a software process fragment that contains the phases of design, code and unit test.

The Abdel-Hamid and Madnick (AHM) model [1] contains eight main sectors as follows: Human Resources, Manpower Allocation, Job Size Adjustment, Productivity, Control, Planning, Quality Assurance and System Test. The AHM model contains parameters pertaining to staffing levels, worker motivation, changes in the workforce due to hiring, training, transfers, and attrition, dynamic variables based on experience, exhaustion, motivation, and communication losses and so forth. These parameters combine with others to form extensive feedback loops that effect overall worker productivity and the efficiency in a complex, non-linear manner. These effects impact the overall execution of the software process (captured by the discrete event portion of the model).

The ISPW-6 process fragment which is contained in the Discrete portion of the Hybrid model was initially documented by Kellner [12] as part of the 6th International Software Process Workshop. This process fragment was developed to test software process modeling paradigms for their ability to capture real-world software process issues and has been used as a benchmark by numerous researchers in the field.

Some of the changes we made to the ISPW-6 Process to make the discrete portion of the Hybrid model more realistic:

- The original ISPW-6 example process specifies the activities required to implement *one* change to *one* module. Our model extended the process to allow code creation and testing on a *series* of modules. This enabled the model to simulate the effects of queuing, and allows us to examine system bottlenecks. In addition, each activity is capable of operating on *several* modules simultaneously. This simulates the effect of (1) concurrent tasks within an activity (e.g. the effect of several modules being coded simultaneously), and (2) concurrent activities across development phases (e.g. several modules may be in coding some while others are in design and still others may be in test).
- The duration of an activity depends on both the characteristics of the module being processed (size, complexity, etc), and overall system characteristics such as resource allocation, productivity and error rates. We modeled these characteristics by implementing the System Dynamics model of Abdel-Hamid and Madnick [1] in Extend, and used it to generate dynamic error rates, productivity rates and resource levels. The dynamic nature of these characteristics means that the time required to process a module of a given size can change during the project. For more detail on the Hybrid model design and implementation see [17] or [18].
- The original ISPW-6 example process does not contain a code inspection. We based our discrete event process model on Raffo's ISPW-6 state-based model [21] which did include code inspections. Furthermore we included a re-inspection step. The process change that we present in this paper deals with whether or not to eliminate these re-inspections from the process.

As mentioned previously, the increased complexity of hybrid models, opens up a whole new (and necessary) spectrum of questions that can potentially be addressed. We believe that DOE coupled with BRSA makes it possible to fully capitalize on this potential.

Although there are notable exceptions (c.f. Hood and Welch [8] and Porcaro [19]), simulation studies often rely on ad hoc exploration of the input parameter space, a process that cannot be relied upon to reveal the subtle interactions that often exist among various components of a complex non-linear model. We advocate DOE and BSRA to help reveal the interactions and nonlinear effects at work in the model, leading to a better understanding of the model logic and model behavior, and hence to a better understanding of the underlying system/process; ultimately resulting in better policy recommendations.

3. DESIGN OF EXPERIMENTS (DOE) AND BROAD RANGE SENSITIVITY ANALYSIS (BRSA)

3.1 Design of Experiments (DOE)

DOE or experimental design is a statistical technique for organizing and analyzing experiments. An excellent overview is provided in Law and Kelton [13]. When applied to computer models, each experiment can require that multiple replications of the simulation model be run. The *factors* are the parameters that are varied (independent variables that are under the experimenter's control), while the *responses* are the dependent variables or outcome measures of interest. Factors might be the model parameters that would be modified to represent different policies for testing and rework under consideration for a SW project, and the responses might be project cost, duration, and escaped errors.

DOE has been applied to simulation in a variety of different ways. For example, in a recent article, Houston et al [10] used DOE to measure the relative contribution of factors to the variation in the response variables in order to behaviorally characterize four System Dynamics software process models.

A conventional and very useful form of DOE is the 2^k factorial design, where each of k factors is allowed to take on two values, a low or minus value and a high or plus value. This design has been shown to be not only economical but also effective at revealing interaction effects [13, pg. 660]. Each combination of factors is called a design point. The entire design is often summarized as a design matrix, with the design points arranged sequentially in the first column, the value of factor k (- or +) indicated in column 1+k. The response is shown in the final column. In the case of simulation runs, there are often multiple replications run at each design point in order to compute the response, which might be the mean value for, say, project duration, over N replications at a particular level of each factor.

Since the number of replications per response is frequently 10 to 30, and the number of design points varies geometrically with k, the number of runs when using DOE with simulation models can be quite large. Thus, one tends to keep the number of factors in the design small. A 2x2 design is quite typical and can be quite revealing.

The primary results from DOE include the amount of variance explained by all of the factors, the overall contribution each factor, and the degree of interaction between the factors. The latter is of particular interest, because it implies that certain combinations of factor values may be particularly effective or ineffective in terms of the resulting response.

3.2 Combining Broad Range Sensitivity Analysis (BRSA) with DOE

Traditional sensitivity analysis typically targets small incremental changes around the expected value of the model parameter. In BRSA, parameter values are varied over their entire plausible range in order to uncover non-linearities and specific parameter values that significantly impact the results. Interestingly, this is often not done because of the number of runs involved and the challenge of organizing the runs and analyzing the results. For our work, we leveraged the linkage between Excel and Extend in order to facilitate this process. One important result of BRSA is the determination of whether the response varies linearly or at monotonically with a particular parameter or if there is in fact a highly non-linear effect where the response shifts from varying directly with the parameter to varying inversely with the parameter.

BRSA complements DOE. While DOE can reveal interaction effects, it does not show non-linear effects. Sensitivity analysis on the other hand, while it does not show interactions, can reveal non-linearities. The combination improves significantly upon ad hoc sensitivity analysis. Moreover, the added cost in simulation runs is not as great as might be experienced using ad hoc sensitivity analysis because of the systematic approach that is used to design the simulation experiments and the efficient manner in which the output information is used.

As models become more complex, it becomes increasingly difficult to understand the subtle interactions and nonlinearities embedded in the logic of the model. Software Process Simulation models are generally complex forms of SD, DES, and Hybrid models (in particular). The BRSA approach provides valuable insights for these types of complex models.

The BRSA approach is particularly helpful in data poor environments such as software development, since it can help to identify the most influential parameters and the most influential ranges of those parameters (often in subtle or non-obvious ways) that might merit additional data collection efforts.

In a fashion similar to that suggested by Cheng and Lamb [4] who used VBA to link Excel and SIMUL8, we have linked the Extend simulation engine to both Excel and to a statistical analysis package (Minitab). This allowed us to design a set of experiments, and then analyze the results to uncover statistically valid interactions. We were also able to use this linkage to efficiently conduct the BRSA to reveal non-linear effects in the model.

Since the hybrid model presented in this paper was created using Extend, which allows the user to make significant modifications to the simulation engine, we were able to create simulation blocks that performed multiple runs using input data stored in Excel workbooks. Minitab was used to design factorial experiments and transfer the input data for each factor to Excel. The simulations were run in Extend, with the output data being captured in Excel. The output data was then moved back to Minitab for analysis. Excel was similarly used to organize and conduct the runs needed for the sensitivity analysis.

The remainder of the paper presents a case study of the application of DOE and BRSA using the Hybrid software development process model described in section 2.2. The DOE illustration is the simplest possible, a 2x2 design, and yet still shows non-obvious results. The BSRA illustration uses the same underlying model, and considers five parameters. Again, the results are not obvious, and greatly helped to further the modelers' understanding of the subtleties of the model logic and the process being modeled.

4. CASE STUDY: APPLICATION OF DOE AND BRSA TO STUDY THE IMPACT OF REMOVING INSPECTION STEPS ON PROJECT DURATION AND ESCAPED ERRORS

One important decision that software project managers need to make is whether or not to have inspections. While inspections can be very costly, the results of the inspections have a significant impact on product quality, development costs, and the time required to complete the remaining development and testing steps. Research has shown that inspections can be highly effective at removing defects and can have significantly beneficial impacts to project cost and schedule [3, 7, 11]. The key question for a project manager is: What will be the impact of adding inspection steps to my specific development process, given my staffing situation, and my particular development environment? For many project environments, worker motivation, schedule pressure, and workforce experience all have significant impacts on the effectiveness of inspections at any given point in the project. Moreover, these environmental factors combine and interact in complex ways and the results are not always intuitive.

It is important to understand how inspections impact overall project results, but the effects when these important project environmental factors are considered can be subtle and counterintuitive. We will use a Hybrid simulation model that captures both project environment effects as well as details regarding the software development process to explore this situation. We then use DOE and BRSA to gain insight into the interaction effects and nonlinearities.

4.1 Overview of the Software Development Process Model

The process used in this model was a modified version of the ISPW-6 software process example developed by Kellner et al [12] . This process was designed to capture eighteen realistic process issues. This process example has been used as a benchmark by numerous researchers and practitioners to test their software process modeling tools. The original ISPW-6 process model was modified by Raffo and Kellner [21] as part of a state-based simulation modeling effort, to include a Fagan code inspection [7] and to utilize a data set that was developed by interviewing several software engineering professionals. The software development process included in the Hybrid simulation model described in this paper uses this process modification and further changes the moderator inspection process step into a full code re-inspection.

The contemplated process change is the removal of the re-inspection step. One might anticipate that if the re-inspections are effective, then removing them might actually increase overall project duration because more errors would escape to the testing phases and would therefore be detected and corrected later in the process when it takes more effort to correct them. On the other hand, since re-inspection will detect fewer errors than a first-time inspection, perhaps removing them will shorten the project duration. The actual result will depend on the specifics of the project environment, and will be manifested in the simulation and subsequent analysis as an interaction effect.

In the modified ISPW-6 process, a full code inspection is done, the errors are fixed and then, if needed, a re-inspection is done. Many of the errors that escape both inspections are detected in Unit Test and reworked in the Unit Test Rework step. Removing the re-inspection step is likely to cause more errors to reach unit test, extending the duration of that step. On the other hand, not performing the re-inspection will save time and resources. The duration of the re-work step that follows Unit Test, and the number of errors corrected in Unit Test rework should both increase when code re-inspections are eliminated. Moreover, we would expect that a portion of these escaped defects from the code re-inspection would find their way into the later phases of Integration, Function and System test. Since these processes are beyond the scope of the ISPW-6 model, we treat Unit Test as a general testing step (with correspondingly higher costs than might be typical).

The factors of interest are whether or not re-inspections are performed (perform or skip) and inspection effectiveness (measured by the percentage of project effort allocated to inspection activity: 5% is low and 15% is high). Table I summarizes the practical aspects of the experiment.

	Skip Re-inspections	*Perform Re-inspections*
Effectiveness of Inspections [and re-inspections] is High (=15%)	Conventional wisdom says this would probably makes the most sense—do it right and do it once.	Quality zealots would probably advocate this scenario—in order to minimize escaped errors.
Effectiveness of Inspections [and re-inspections] is Low (=5%)	Time to market zealots might advocate this scenario—do it quickly and forget it.	It is not likely that anyone would advocate doing ineffective inspections twice.

Table I: Practical Interpretation of the Experiment

We will examine each factor at two levels, a 2X2 factorial design, with 10 replications per response, for a total of 40 runs (test runs had indicated that 10 replication per response were sufficient). We will examine the effects on two output measures, project duration and the number of errors detected in unit test. The assumption is that highly-effective re-inspections remove enough additional errors that if they are skipped, the increased errors detected and corrected in unit test will cause the overall duration of the project to increase. We also want to know if the errors detected in unit test increase when the inspection step is skipped, and if the resulting increase in unit test time is sufficient to nullify the time saved by skipping the re-inspection step.

4.2 DOE Results

Figure 1 shows the results of an analysis of variance (ANOVA) on the results of the 40 simulation runs. In half of the runs, the re-inspection was performed, and the mean project duration for these runs was 610 days. For the other half of the runs, re-inspection was skipped, and the mean project duration for these runs was 565 days.

Half of the 40 runs (a different half) had low inspection effectiveness. The right hand graph in Figure 1 shows that the mean project duration for these cases was just over 590. The mean project duration for other half of the runs, those with high inspection effectiveness, was just over 580 days.

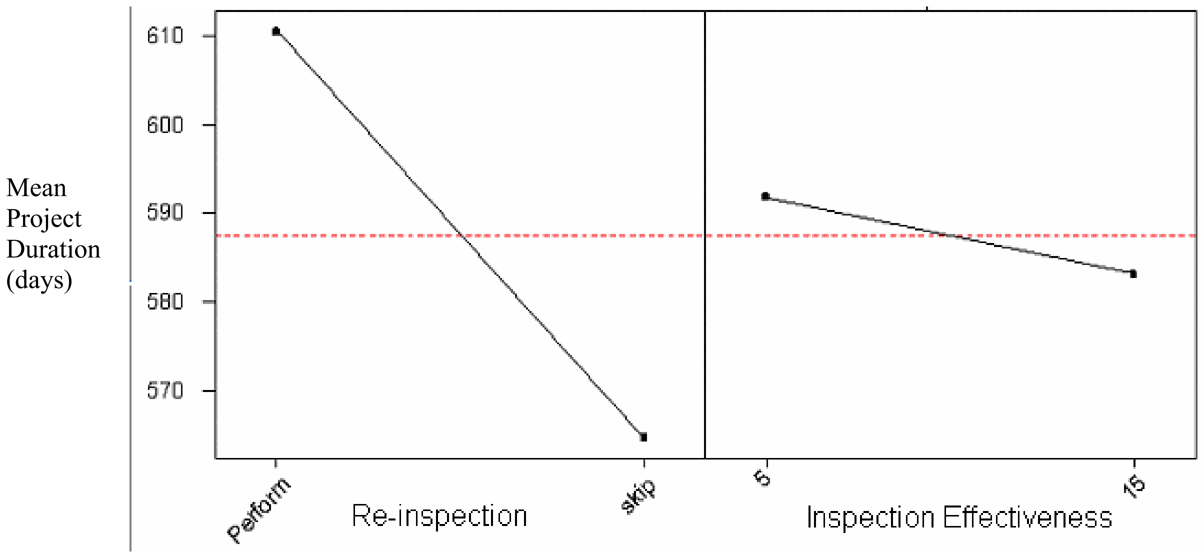

Figure 1 -- Main effects of Re-inspection and Inspection Effectiveness on Project Duration

The interpretation of Figure 1 is that skipping re-inspection has a significant beneficial impact on the project duration, whereas the impact of low vs. high inspection effectiveness is in the expected direction but not significant. The interaction effect between inspection effort and skipping the re-inspection step (not shown) was also not significant.

This result is contrary to what one might expect--that skipping the re-inspection would *increase* project duration when the inspection effectiveness is high, and would decrease project duration only when the inspection effectiveness inspections is low.

To further examine the simulation results, we consider the number of errors detected (which is a quality measure that can be directly observed) in unit test in each of the four cases. Figure 2 plots the main effects of Re-inspection and Inspection Effectiveness on Errors detected in Unit Test.

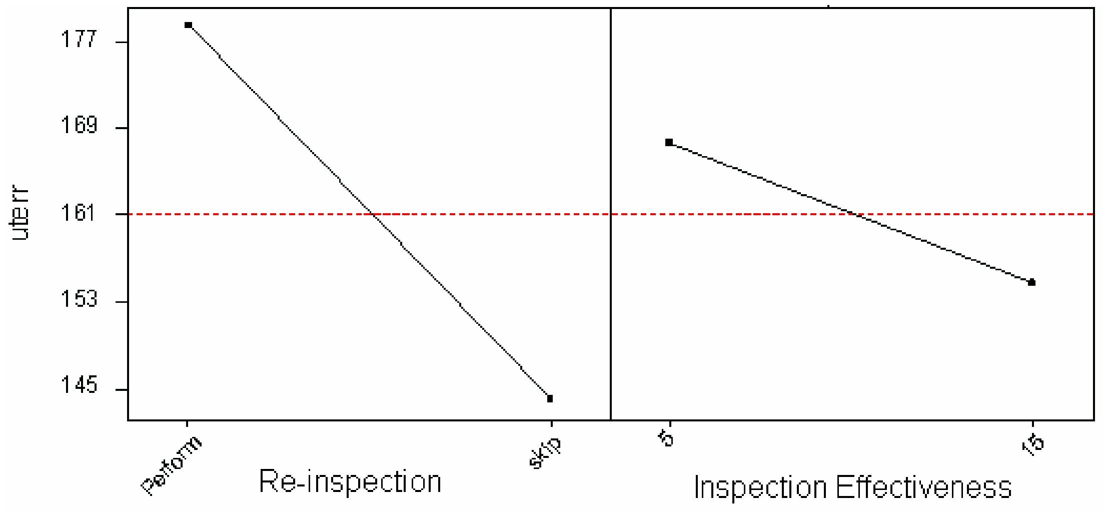

Figure 2 -- Main Effects of Re-inspection and Inspection Effectivness on Unit Test Errors

Here again we see that only the Re-inspection had a significant effect. Once again, the results differ from what one might expect--that removing the Re-inspection would cause more errors to reach Unit Test, and therefore cause more errors to be *detected* in Unit Test, as is frequently discussed in the literature. For example, Jones [11], Fagan [7] and Boehm [3] describe the increased cost of fixing errors late in the project.

However, the DOE factorial experiment shows the assumption not to be true *in this case*. Such a behavioral anomaly forces the modeler to either correct the model if the anomaly is due to flaws in the model, or leads to a deeper understanding of the referent system.

In order to reconcile this anomaly, we must examine the assumptions of the case. Specifically, we need to examine the assumptions about the error generation rates for the project. Using design error generation rates of 25 errors per KLOC and code error generation rates of 12.5 errors per KLOC over 800 errors escape into the finished code. If Unit Test is only capable of detecting between 100 to 200 errors[1], as indicated by Figure 2, the Unit Test step will be able to detect approximately the same number of errors regardless of inspection efficiency or the presence or absence of a moderator inspection.

Thus, it is possible that the anticipated result will only occur when the error generation rate is small enough that the number of potentially detectable errors that reach Unit Test is less than the detection capacity of Unit Test. We test this by repeating the previous experiment with a reduced error rate (6 errors/KDSI for design and 3 errors/KDSI for code).

[1] In Hybrid Models, defects are detected at a certain rate. As a result, the number of defects detected largely depends upon the amount of effort (staff hours) allocated to that activity, although the rate depends upon a variety of factors. In this model, given the time allocated to Unit Test, 100-200 defects is the limit of the Unit Test Phase's capacity to detect defects.

The results of the ANOVA for the reduced error rate case indicates that the interaction effect between Re-inspection and Inspection Effectiveness is now significant, as shown in Figure 3.

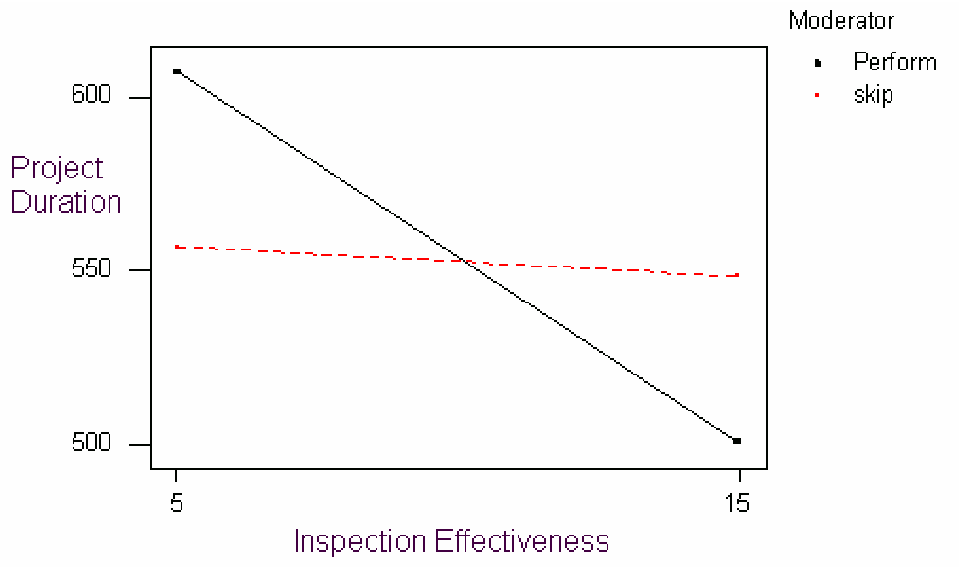

Figure 3 -- Interaction between Inspection Efficiency and Re-Inspection on Project Duration with Low Error Rates

The interpretation of Figure 3 is that with low error injection rates, when inspection effectiveness is 5, performing Re-inspections yields an average project duration of just over 600 days. The duration is significantly lower, 550 days, when Re-inspections are skipped. However, when inspection effectiveness is 15, project duration drops to 500 days when Re-inspection is performed; and now, when the re-inspection is skipped, the duration *increases* significantly to 550 days.

This result is significant and suggests that Re-inspections are advantageous when project error injection rates are relatively low.

However, as indicated in the first set of experiments, when the error injection rates are high, skipping Re-inspections will *not* increase project duration (see Figure 1) because it will not appreciably change the number of errors discovered and corrected in Unit Test. It does, however, allow more errors to escape (although not shown on the figures).

4.3 Broad Range Sensitivity Analysis Results

The DOE analysis indicates that project duration is likely to increase when Re-inspections are eliminated in a low error injection environment. Naturally, this result depends on several parameters that were held constant during the experiment. We will illustrate the use of BRSA by investigating how much these parameters can change without changing this conclusion.
We note that increases in project duration appear to be caused by the additional time needed to rework an increased number of errors in Unit Test. By studying the logic of the model, we determine that rework time depends on the number of errors detected and the time to rework each error. The time to rework each error depends on the error rework rate (a user specified function) and the *relative difficulty* value specified for unit test. The number of errors detected depends on the *effort* (referred to as inspection effectiveness in the DOE analysis), the *error generation rate,* the *error detection rate*, and the *error density multiplier*.

Because of their potential significance to the results, the five model parameters shown in italics above were selected to illustrate BRSA. Each parameter was varied over a broad range of plausible values taken from the literature. The details for each parameter are discussed below. Ten replications were run for each of these values, with everything else held constant, for a total of 50 model runs.

Figure 4 shows the mean of the 10 runs for project duration, as a function of the parameters as they were varied over a broad, but plausible range. We discuss each parameter and the interaction between parameter following the figure.

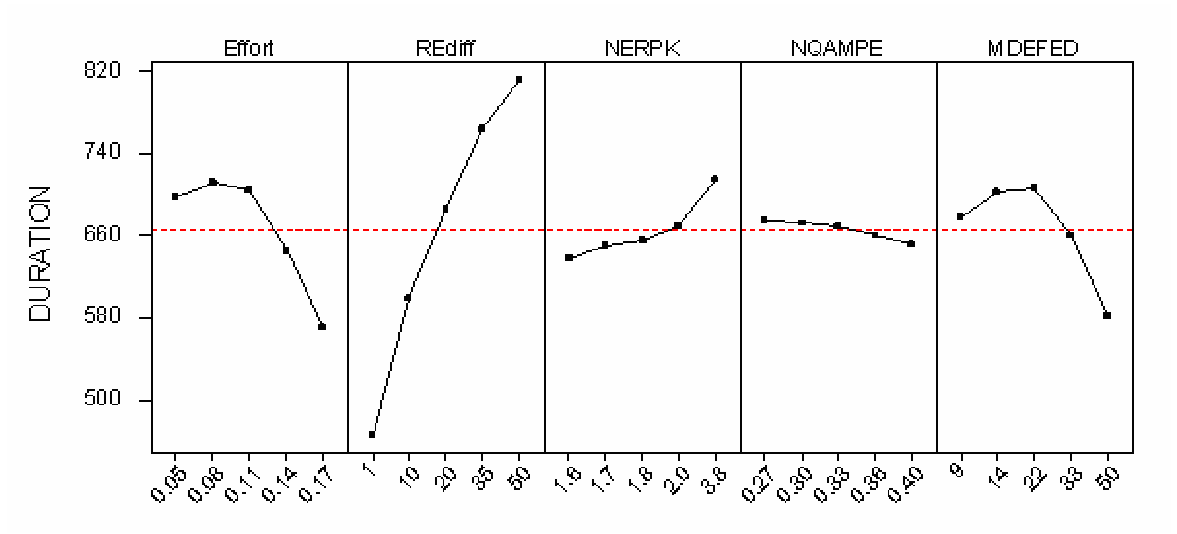

Figure 4 -- Sensitivity of Project Duration to Different Parameters

Effect of inspection effectiveness (Effort)
Inspection effectiveness (effort) is modeled as the fraction of project resources allocated to inspection. At low values of inspection effectiveness (effort), small increases lead to increases in the overall duration because more errors are identified and corrected early in the project without reducing the number of errors detected and corrected later in the project.

For example, imagine a project that injects 1000 errors in design, and then detects 500 errors in inspections and 200 errors in Unit Test. If inspection effectiveness (effort) is increased and the project detects 700 errors in inspections, it would still detect 200 errors during Unit test. The time to detect and correct the additional 200 errors found during inspection would simply increase project duration (and reduce the escaped errors).

However, as we can see from Figure 4, as inspection effectiveness (effort) increases, eventually the number of errors available for detection will drop below the detection and correction capability of Unit Test. At the higher inspection effectiveness (effort) values, correcting the errors in inspection leads to decreases the project duration (in addition to reducing escaped errors) because errors are detected earlier when the number of hours required to correct the errors is less.

Effect of unit test rework relative difficulty (REdiff)
Boehm claims that it is more difficult to fix an error when it is detected late in the development process. The increase in relative difficulty increases with the size of the project. In small projects, the relative difficulty may increase by a factor of 4 from design to unit test. In large projects, it may be more than 20 times as difficult [2]. Fagan [7, pg. 270] used a multiplier of 10 to 100 times for the relative difficulty. Figure 4 shows the effect of this relative difficulty on project duration. Relative difficulty relates the unit test rework effort to the effort required to fix the same error in design. A value of 20 on the X-axis means that it would take 20 times the effort to rework an error in unit test. Relative difficulty is graphed in Figure 4 as REdiff vs. duration. The results are highly sensitive to this parameter, as would be expected, and in a nearly linear fashion. This is not surprising, and means that REdiff can safely be assumed not to confound the decision regarding Re-inspections because regardless of the value used for Rediff the direction of the impact will be same; only the magnitude will change.

Effect of error generation rate (NERPK)
The error generation rate directly affects the number of errors available for detection. Abdel-Hamid and Madnick [1] described the error generation function as an S-shaped curve that began at 25 errors per KDSI and ended at 12.5 errors per KDSI (a ratio of 2 to 1). Several authors have discussed the ratio between design and coding errors. Table 1 (originally compiled by Abdel-Hamid and Madnick[1]) shows the ratios and the sources.

Ratio of Design Errors to Coding Errors	Reference
3.8 to 1	Martin, 1983
2.0 to 1	Alberts, 1976
1.8 to 1	Jones, 1981
1.7 to 1	Boehm, 1981
1.6 to 1	Thayer et al, 1978

Table 2-- Design vs. Coding Errors

Using these ratios and an ending (lower) value of 12.5 errors per KDSI, we generated a family of S-shaped curves that creates the range of different error profiles suggested by Table 2. These curves were entered into the model as five different values for the Nominal Errors per thousand lines of code function (NERPK). As shown in Figure 4, duration increases as the error generation rate increases. The increase in duration is roughly proportional to the increase in error generation rate, but the effect is relatively modest overall. This is not surprising, and indicates that NERPK can also safely be assumed not to confound the decision regarding Re-inspections.

Effect of error detection rate (NQAMPE)
Error detection rates are also described as a function that describes the QA manpower (in staff-days) needed to detect an error. The function is an S-shaped curve that assumes a higher value early in the project, but drops later to reflect the assumption that coding errors are easier to detect than design errors.

Using the reference originally cited by Abdel-Hamid and Madnick [1], we assume a range of values from .27 staff-hours per error to .4 staff-hours per error. As the QA manpower needed to detect an error increases, the number of detected errors drops. The reduced number of detected errors reduces the amount of rework and thus reduces the project duration. This is plotted in Figure 4 as NQAMPE. The effect is quite modest, and not surprising. NQAMPE can also safely be assumed not to confound the decision regarding Re-inspections.

Effect of error density multiplier (MDEFED)
Abdel-Hamid and Madnick modeled error detection with an error density function that increased the difficulty of error detection as errors were removed. Specifically, MDEFED increases error detection difficulty by as much as a factor of 50 times when only one error remains. We preserve the shape of the function, but vary the maximum multiplier from 9 to 50. When the multiplier maximum is smaller, the difficulty of detecting an error is smaller, so that enough errors are detected to allow duration to increase slightly. However, when the multiplier is large, the increased detection difficulty reduces the number of errors detected, and thus reduces duration (but at the cost of more escaped errors). This is plotted in Figure 4 as MDEFED. Product duration is very sensitive to this parameter, and the relationship is highly non-linear, reinforcing the proposition that increasing inspections cannot be counted on to improve performance in all cases.

5. CONCLUSIONS

In order to fully analyze potential SW process changes, one must embed SW development process simulation models within an experimental framework, and then use that framework to fully understand the implications of these complex simulation models. This requires that simulation, data management, and statistical analysis tools be effectively linked together in order in order facilitate DOE analysis to reveal interaction effects and BRSA to better understand non-linear aspects.

The example provided, the contemplated removal of a re-inspection step in a complex SW development process, shows a combination of both anticipated and unanticipated results, which underscores the importance of running a set of well designed experiments and analyzing the results statistically. Should one remove re-inspection steps to save time? It depends, or course, on several factors.

When the error injection rate is reasonably low, DOE reveals an interaction effect between inspection effectiveness and whether or not re-inspections are done (Figure 3). Skipping re-inspections when inspection effectiveness is high is likely to *increase* overall project duration due to the additional time required to correct the errors when they are detected later in the process. But, if the inspection effectiveness is low, the capacity in Unit Test to detect and correct errors is overwhelmed regardless of the inspections, and therefore skipping re-inspections reduces overall project duration.

However, if the error injection rate is high, the interaction effect is no longer significant, in which case skipping re-inspections would *reduce* project duration regardless of whether or not the inspection effectiveness is high or low.

BRSA further indicates that the results are highly sensitive in a *non-linear* fashion to the relative difficulty to correct errors, and to the inspection effectiveness. The results are also highly sensitive in a *linear* fashion to the error density multiplier. This suggest that additional attention should be focused on these particular relationships, and that the decision whether to skip or perform re-inspections, for example, depends very much on specific details of the software development environment.

6. ACKNOWLEDGEMENTS

This work has benefited from modeling support and participation from Siri-on Setamanit. Her contributions are gratefully acknowledged. In addition, the authors are grateful to the Software Engineering Research Center, a National Science Foundation Industry/ University Collaborative Research Center and Northrop Grumman Corporation for supporting this research effort.

7. REFERENCES

[1] Abdel-Hamid, T. and Madnick, S., *Software Project Dynamics: An Integrated Approach*, Prentice-Hall software Series, 1991, ISBN 0-13-822040-9

[2] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981, ISBN 0-13-822122-7.

[3] Boehm, B., Clark, B., Horowitz, E., Madachy, R., Shelby, R., Westland, C., "Cost Models for Future Software Life Cycle processes: COCOMO 2.0" *Annals of Software Engineering*, (1995).

[4] Chen and Lamb, "Interactive Implementation of Optimal Simulation Experiment Designs", *Proceedings of the 1998 Winter Simulation Conference*, 1998, pp 707-712.

[5] Donzelli and Iazeolla, "Hybrid Simulation Modelling of the Software Process", *Journal of Systems and Software*, Volume 59, Number 3, December 2001.

[6] Extend, developed by Imagine That, Inc., http://www.imaginethatinc.com.

[7] Fagan, M. E., "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, vol. 15, no. 3, 1976

[8] Hood and Welch, "Experimental Design Issues in Simulation with Examples from Semiconductor Manufacturing", *Proceedings of the 1992 Winter Simulation Conference*, 1992, pp 255-263.

[9] Host, Regnell, Dag, Nedstam and Nyberg, "Exploring Bottle Necks in Market Driven Requirements Management Processes with Discrete Event Simulation", *Journal of Systems and Software*, Volume 59, Number 3, December 2001

[10] Houston, et al, "Behavioral characterization: finding and using the influential factors in software process simulation models", *Journal of Systems and Software*, V 59, 2001, pp 259-270.

[11] Jones, C., *Applied Software Measurement*, 2nd Edition, Mc Graw-Hill, 1996.

[12] Kellner et al., "ISPW-6 Software Process Example", *Proceedings of the First International Conference on Software Process*, Held at Redondo Beach, California, October, 22-22, 1991, IEEE Computer Society, 1991, pp 176-186.

[13] Law and Kelton, *Simulation Modeling and Analysis*, 3rd Ed., McGraw-Hill, 2000.

[14] Lehman and Ramil, "The Impact of Feedback in the Global Software Process", *Journal of Systems and Software*, V 46, No. 2/3, April 15, 1999.

[15] Madachy, R.J., " A Software Project Dynamics Model for Process Cost, Schedule, and Risk Assessment", Ph.D. Dissertation, Dept. of Industrial and Systems Engineering, University of Southern California, December 1994.

[16] Madachy, R., "System Dynamics Modeling of an Inspection-Based Process", *Proceedings of the Eighteenth International Conference on Software Engineering*, IEEE Computer Society Press: Berlin, Germany, March 1996, pp. 376-386.

[17] Martin, R.H., "A Hybrid Model of the Software Development Process", Ph.D. Dissertation, Dept of Engineering Management, Portland State University, March 2002.

[18] Martin, R.H. and D.M. Raffo, "Application of a Hybrid Process Simulation Model to a Software Development Project", *Journal of Systems and Software*, Vol. 59, 2001, pp. 237-246.

[19] Porcaro, D., "Simulation Modeling and DOE", *IIE Solutions*, September 1996, pp 24-30.

[20] Powell, Mander and Brown, "Strategies for Life Cycle Concurrency and Iteration", *Journal of Systems and Software*, V 46, No. 2/3, April 15, 1999.

[21] Raffo, D. and Kellner, M., "Predicting the Impact of Potential Process Changes: A Quantitative Approach to Process Modeling," *Elements of Software Process Assessment and Improvement*, IEEE Computer Society Press, 1999.

[22] Raffo, Vandeville, and Martin, "Software Process Simulation to Achieve Higher CMM Levels," *Journal of Systems and Software*, Vol. 46, No. 2/3 (15 April 1999), pages 163-172.

[23] Tvedt, J., "An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time", Ph.D. Dissertation, Arizona State University, Tempe, Arizona, 1996.

[24] Wernick and Lehman, "Software Process White Box Modelling for FEAST/1", *Journal of Systems and Software*, V 46, No. 2/3, April 15, 1999.

8. BIOGRAPHIES

Wayne Wakeland
Dr. Wakeland is an Associate Professor of Systems Science at Portland State University, where he teaches system dynamics, discrete system modeling, manufacturing simulation, business process modeling & simulation, agent based simulation, and organizational theory and dynamics. His research interests include software process simulation, biomedical simulation, sustainability, supply chain logistics, and optimization in conjunction with simulation. Dr. Wakeland has also held IT and manufacturing management positions at Leupold & Stevens, Epson, Magni Systems, Photon Kinetics, and Tekronix. He was also a research associate on an NSF-funded project on technology assessment. Dr. Wakeland received a B.S. in Engineering and Master of Engineering from Harvey Mudd College, and a Ph.D. in Systems Science from Portland State University.

Bob Martin
Dr. Martin has been doing and managing software development since 1967. He has worked for Computer Sciences Corporation, E-Systems, and Tektronix. His consulting company, "Software Management Consulting" has provided decision models to British Petroleum, Alyeska Pipeline, Motorola, and Northrop-Grumman. Dr. Martin received a B.S. in Mathematics from the University of Central Florida, and a M.S. in Engineering Management and Ph.D. in System Science/ Engineering Management from Portland State University.

David Raffo
Dr. Raffo is an Associate Professor of Technology Management and Information Systems at Portland State University. His research interests include: software process modeling and simulation, strategic software engineering, and process improvement. He has over twenty-five refereed publications in the field of software engineering and has received research grants from the National Science Foundation, the Software Engineering Research Center (SERC), NASA, IBM Corporation, Tektronix Corporation, and Northrop-Grumman Corporation. Prior professional experience includes programming as well as managing software development projects. He has also taught in Purdue University's Software Engineering Retraining Program (SERT) through the Department of Computer Science. Dr. Raffo received his Ph.D. at Carnegie Mellon University.