

Coordinating Models and Metrics to Manage Software Projects

David Raffo
Portland State University
School of Business
PO Box 751
Portland, OR 97207 USA
503-725-8508
 davidr@sba.pdx.edu

Warren Harrison
Portland State University
Computer Science
PO Box 751
Portland, OR 97207 USA
503-725-3108
 warren@cs.pdx.edu

Joseph Vandeville
Northrop Grumman
Software Engineering.
P.O. Box 9650
Melbourne, FL 32902 USA
407-951-5287
 vandejo1@mail.northgrum.com

ABSTRACT

In previous work we developed techniques for modeling software development processes quantitatively in terms of development cost, product quality, and project schedule using simulation. This work has predominately been applied to the software project management planning function. The goal of our current work is to develop a "forward looking" approach that integrates metrics with simulation models of the software development process in order to support the software project management controlling function. This "forward-looking" approach provides predictions of project performance and the impact of various management decisions. It can be used to assess the project's conformance to planned schedule and resource consumption. This paper reports on work with a leading software development firm to create an approach, that includes a flexible metrics repository and a discrete event simulation model.

Keywords

Software Process, Project Management, Simulation Modeling, Software Measurement Repositories

1 INTRODUCTION

Effective project management is critical to the success of software development projects. For large projects, software project management is often broken into two separate functions: planning and control. *Planning* is forward looking. It tells us what needs to be done, when it is to be done, how it is to be done and who is going to do it. Planning usually occurs prior to embarking upon a

project or early in the project life cycle. *Controlling* is intended to keep events on course by identifying and correcting deviations from the plan. It has a more narrow and immediate focus. It is intended to alert managers to significant deviations from plan while the project is in process.

The planning function can address a variety of useful questions regarding the potential performance of a company's software development process. Questions related to predicting effort, cost, schedule, and product quality; developing forecasts of staffing levels over the duration of the project, addressing resource constraints and resource allocation, predicting service-levels for product support, analyzing risks and so forth [KeMR 99].

The control function is different in focus, timing, and scope. It involves the on-going management of a project. It compares the likely final state of the project (actual performance) with respect to cost, quality and schedule with the state called for by the original plan. Deviations are reported to the project manager who then may choose to take some specific action to bring the project back into control. Timely and accurate data are necessary to provide an accurate picture of where the project currently is and to make a prediction of where the project is likely to go. The ability to quantitatively monitor and assess software projects helps support the Quantitative Process Management and Software Quality Management Level 4 Key Process Areas (KPAs) of the Capability Maturity Model (CMM) [Humphrey 89, Paulk+ 93].

In a domain where "gut feel" and subjective estimates are common, software project managers have often looked for

tools and an approach to provide quantitative data on current project status and quantitative estimates on potential project outcomes.

In previous work we developed techniques for modeling software development processes quantitatively in terms of development cost, product quality, and project schedule using simulation. This work has predominately been applied to the software project management planning function [Raffo 96; RVM 99]. The goal of our current research is to develop a “forward-looking” approach that integrates metrics with simulation models of the software development process in order to support the software project management *control* function. The forward-looking approach provides predictions of project performance and the impact of various management decisions. By combining metrics and predictive models it provides a more comprehensive picture than can be achieved using metrics alone. In addition, the predictive models can support managers as they attempt to replan and bring the project back on track. A key element of this approach is the development of a flexible metrics repository which links corporate databases with software process simulation models. This paper reports on work with a leading software development firm to create an approach that includes a flexible metrics repository and a discrete event simulation model based on the company’s software development process.

Section 2 provides a background regarding various models used to predict performance of software development processes. Primary attention is given to software process simulation models, which support the project planning function by capturing process-level issues. In addition, the expanded information needs required to support the control activity and the need for the metrics repository are shown. Section 3 presents the metrics repository and how it supports the controlling function by providing up-to-date information in a flexible manner. Section 4 provides an overview of a discrete event simulation model that has been developed. A distinction is made between the representation used by the discrete event simulation model and other process modeling approaches that makes the discrete event paradigm highly compatible with the metrics repository described in section 3. Section 5 discusses the controlling activity, outcome-based control limits (OBCLs) and decisions supported by our approach. Section 6 discusses potential benefits, conclusions and future work.

2 MODELING APPROACHES USED TO SUPPORT THE PLANNING ACTIVITY

Planning is forward looking. A variety of models and methods have been applied to support project planning.

These models and methods have been used to predict various aspects of performance of software development projects. Some of these approaches have been static models applied to one dimension of performance such as product quality or reliability [MIO 87, Musa 75, Trivedi 82]. Other approaches utilize high-level models to estimate costs and predict project-level performance. These models typically use summary inputs of the project and abstract out details of the underlying software development process. These models may also capture multiple dimensions of performance such as cost and schedule. Well known examples of these types of models include COCOMO [Boehm 81, Boehm+ 95] and SLIM [Putnam 78, Putnam 80].

In recent work, Raffo developed the Process Tradeoff Analysis (PTA) Method. This method builds on previous work by Kellner et al. at the Software Engineering Institute (SEI) [KH 88, KH 89, Kellner 89A, Kellner 89B, Kellner 90] by developing a quantitative approach to evaluating potential process changes in terms of development cost, product quality, and project schedule [Raffo 96]. The core of the PTA method addresses evaluating process alternatives quantitatively by developing stochastic state-based simulation models of each process alternative. These models explicitly capture process-level details including complex interdependencies among process components. The PTA Method has been applied to real-world process change problems at leading software development firms. [Raffo 96, RK 96, RVM 99].

Recently, other researchers have also modeled software development processes at a more detailed level than COCOMO and SLIM models using a variety of software process simulation techniques such as discrete event simulation, system dynamics or continuous simulation, knowledge-based systems, and scheduling simulation. [Scacchi 99, Madachy 94, Tvedt 95]

These models capture project-level issues which is a critical feature needed to support planning activities. In addition, the state-based and discrete event simulation process models [KoMR 98, RK 95, Raffo 96] are stochastic and can provide a quantitative assessment of risk.

The above approaches have the advantage of utilizing high-level parameters that are typically available for software development projects. Using these parameters and inputs, which are often captured at the end of a project, useful information and project-level estimates can be predicted for future projects.

However, in order to

1. Support planning decisions related to the project and processes being used,

2. Evaluate alternative variations of different processes and make choices (e.g. formal inspections or reviews, different ways to conduct integration test, and so forth), and
3. Allocate resources among different process sub-tasks,

more detailed models which capture process-level and product-level issues are needed.

Project-status metrics are used to provide indicators for the project. However, these metrics provide only snapshots of specific individual aspects of project status. By incorporating these metrics into a life-cycle model of the software development process, a more complete multi-dimensional performance picture can be created.

By capturing the details related to actually executing software projects, software process simulation models take a very significant step forward in supporting project planning and control activities. This step forward is attained by modeling the software development process to a finer level of granularity and utilizing lower-level project data. However, the timeliness of data sources for these models (i.e. data obtained from past projects) has remained the same. In order to provide an accurate picture of current project status, up-to-date project information is needed.

The work presented in this paper describes an approach for integrating metrics with simulation models of the software development process. We describe necessary support tools such as a flexible metrics repository and discuss the interface between the repository and the simulation models.

3 SOFTWARE METRICS REPOSITORY

Up-to-date project and process information is necessary to support planning and control decisions about a project. The metrics repository stores the necessary information and provides the critical link between raw project metrics and model parameters. Since the model needs to provide a timely view of the project at all times, the repository must facilitate the collection of data on a "real-time" basis. However, because of the natural evolution of models and their corresponding information needs and the different levels of granularity in which data may be captured, the repository must also possess an incredible level of flexibility.

The repository is based upon a "transformation view" of the software development process. Artifacts such as specifications, designs and code are "transformed" by the

application of a "transformation process" into a new artifact. For instance, a design artifact may be transformed into a code artifact by the application of a "programming transformation".

Artifacts possess certain properties, such as "size", "volatility", "complexity", etc. and the transformation process possesses other properties such as "resources consumed", "errors made", etc. The transformation process as well as the artifacts themselves can be represented in the following simplified entity-relationship diagram (figure 1).

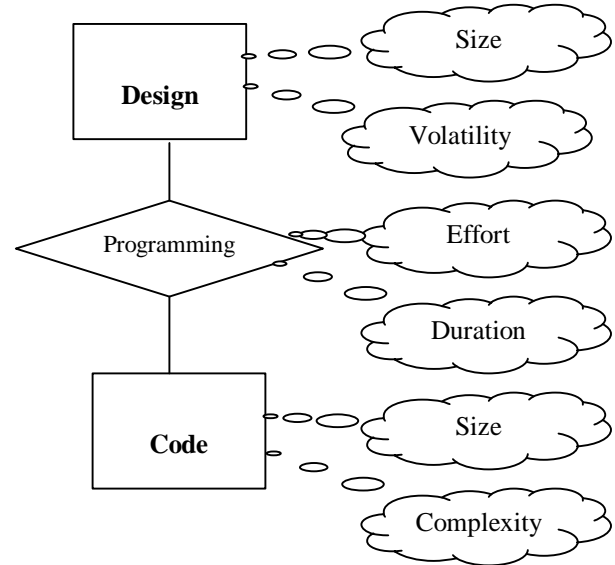


Figure 1 – Entity-Relationship Diagram of a Portion of the Transformation Process

In short, this model denotes that an Artifact is related to another Artifact through some "transforms" relationship. In order to provide an historical record of the state of the project as it progresses through (or is "transformed by") the development process, we record snapshots of project characteristics each time a *significant* "transformation" occurs. The threshold of *significance* can be adjusted to record snapshots at whatever level of detail is necessary. For instance, snapshots could be taken every time a changed piece of code is checked back into the revision control system, or they could be taken every time the project proceeds to the next phase of the process. The level of significance which is selected will have an impact on the degree to which the repository can be used to facilitate control activities (more detail supports greater control).

It is quite easy to characterize the entire development process as a series of transformations. The process begins

with a requirements specification, which can be considered the initial artifact. The artifact is transformed into a design artifact through the application of a "design transformation" (i.e., this represents the design activity). Should additional granularity be desired, the "design activity" could be decomposed into several transformations, such as transforming the requirements artifact into an architectural design artifact. The architectural design artifact could then be transformed into a structure chart artifact, which could finally be transformed into a detailed design artifact.

The detailed design artifact may be transformed into code through a programming transformation. The code artifact could be transformed into an "inspected code" artifact through the inspection transformation. Inspected code may be transformed into "final code", "tested code", "delivered code", etc.

A notable point is that the relationships between artifacts and transformations are many-to-many. That is, a design artifact may be transformed into several different code artifacts. For instance, a structure chart artifact may be transformed into a code artifact for every node in its tree. Likewise several different artifacts may be transformed into a single artifact. A code artifact and a maintenance request artifact may be transformed into another, revised code artifact.

As each artifact or transformation is recorded, the repository attempts to characterize the state of the project at each snapshot by maintaining information about the transformation and the artifact(s) produced by the transformation. This information is intended to characterize particular "domains" associated with the transformation and the artifact. *It should be recognized that the manner in which we operationalize a particular domain might differ from project to project, and organization to organization.*

For instance, suppose a company has some large projects with formal inspections and data collection. At the same time, the same company also has smaller projects where the data is stored by project, but only to a certain depth. These differences can be accommodated by the level of significance that triggers a "transformation" as well as including a description of the measurement scheme used along with each measurement (figure 2).

The domains of interest associated with a transformation include: the resources expended in the process of doing the transformation, the number of defects found, the number of faults injected, as well as the duration for each transformation.

The resource expenditure domain may involve calendar days, staff hours, equivalent FTE, etc. As a standard

method of operationalizing this domain, we propose actual staff hours expended (including both paid and unpaid overtime). However, if this information is not available, then some other measure of resource expenditure with a description of the counting rules, exceptions, etc. would be acceptable. Clearly, as we move farther and farther away from a "canonical measure of resource consumption", models built upon this data become less and less reliable. The number of faults found and made during a transformation should be noted. In general, it is safe to say that a transformation either inserts faults or finds them. For instance an inspection transformation will find faults, while a coding transformation will create them.

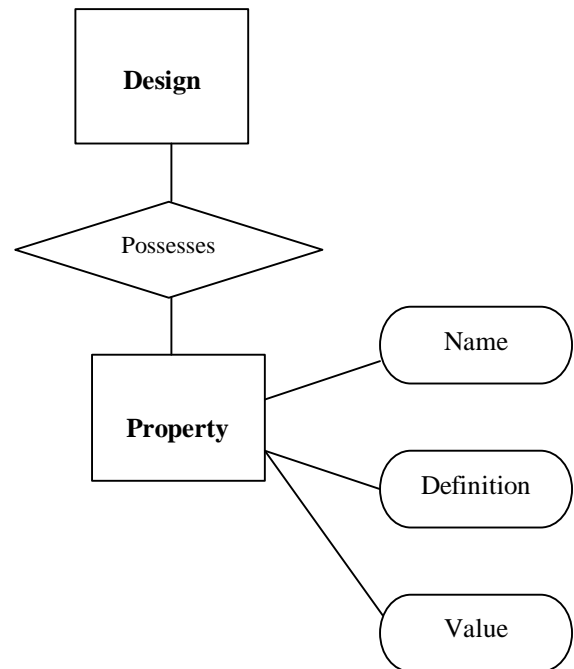


Figure 2 – Entity-Relationship Diagram Showing Design Artifact Properties

The domains of interest associated with each artifact (i.e., code unit, specification, design, etc.) involve size, complexity, volatility and quality (we might better term this "correctness"). Each of these can be operationalized in a variety of ways. Rather than selecting sophisticated and complicated measurement techniques, we have instead chosen to promote the use of simple, easily understood and captured metrics. However, the repository accommodates multiple metrics, and we encourage the collection of additional, more sophisticated metrics as well. Table 1 lists some domains and possible methods of operationalizing them.

Artifacts such as design documents, bug reports, change

requests, etc. are not addressed specifically in this table. However, depending on their level of formality, they should be able to be characterized as a "text document" or a "code document".

Table 1 – Potential Metrics Domains	
Domain	Possible Operationalizations
Size, Code (traditional programming languages)	Source Lines: number of non-blank lines in each artifact. <i>Alternates:</i> Software Science N and V; number of statements; number of subroutines/functions
Size, Text Documents	Text Lines: number of non-blank lines in each artifact. Figures should be counted based on the percentage of the page they cover. <i>Alternates:</i> number of sentences; number of paragraphs; number of pages; number of "key words" such as Shall
Complexity, Code (traditional programming languages)	Decision Count: number of decision statements in the code such as IF, WHILE, REPEAT, CASE, etc. <i>Alternates:</i> Software Science E; number of local and global variables; level of control-flow nesting
Complexity, Text Documents	Readability: Flesch Reading Ease Score applied to the document. <i>Alternates:</i> Flesch-Kincaid Grade Level Score; Gunning Fog Index
Volatility, Code	Number of changes since the last version of the artifact. <i>Alternates:</i> percent of total changes to the artifact that occurred since the last version of the artifact; change in units of each size measure listed above since the last version of the artifact; percent of total changes in units of each size measure since the last version of the artifact.
Volatility, Text Documents	Number of "TBDs" in the artifact. <i>Alternates:</i> Number of changes since the last version of the artifact in units of each size measure listed above; percent of total changes to the artifact that occurred since the last version of the artifact, in units of each size measure listed above.
Quality, Code	Number of faults (not failures) found in the artifact. <i>Alternates:</i> number of test cases (from test plan) passed and failed so far;
Quality, Text Documents	Number of faults (not failures) found in the artifact. <i>Alternates:</i> none

While the repository supports flexibility of data, it also provides a framework for maintaining the current state of

the project. Depending on the level of granularity and frequency of recording, a project manager should be able to determine the current artifacts being developed, as well as all the pertinent information currently available about preceding artifacts. However, data collection, especially of resource consumption properties is an inherently manual process and the data can only be as timely and reliable as the data-collection personnel make it. Therefore, some of the properties will be implemented as "computed fields" by providing links to artifacts such as the project defect tracking system and inserting "update triggers" in transformation sources such as revision control systems and makefile scripts.

The repository is currently implemented using Microsoft Access™ with linkage to the PR-Tracker™ bug tracking system. However, numerous other implementation approaches are under investigation including PROLOG and other commercial DBMS packages. The repository must be concerned both with linking to various data-capture mechanisms, as well as interfacing with the simulation model.

4 COMBINING PROCESS MODELS WITH THE DATA RESPOSITORY

Given the discussion in section 3, it is clear that a detailed artifact-based process model of software development projects would be compatible with the metrics repository. This section briefly describes our on-going work with Northrop Grumman Corporation to develop an artifact- or entity-based model that provides a compatible interface to the metrics repository described in section 3 and is linked through parameters that are generated by a set of database queries. These parameters are updated during various significant project *transformations* to provide timely information that can be used by the model to make improved predictions.

Since mid-1996, Northrop Grumman has been sponsoring research into the use of stochastic simulation models to support software process improvement and quantitative project management issues. The goal of this research is to develop a quantitative simulation model of the software development process that can be used in a forward-looking, predictive fashion to simulate the impact of proposed process changes prior to deployment on software projects. This work is being conducted in collaboration with Portland State University and the Software Engineering Research Center (SERC), a NSF sponsored Industry/University Cooperative Research Center. A new discrete event simulation model has been developed to simulate the activities and artifacts of one of Northrop Grumman's large-scale software development projects. This model contains cost, schedule and quality

data that have been collected from past projects. This research project has been expanded in scope to explore the potential new capabilities of integrating up-to-date metrics information as can be provided by the metrics repository described above with the discrete event simulation model of the software development process.

Northrop Grumman’s SBMS Melbourne site develops software for airborne radar surveillance and battle management systems. The portion of the software development process modeled consists of traditional software life cycle activities. These activities consist of 71 distinct development steps. The selection of steps was based on availability of distinct metrics. If two activities could not be distinguished using their metric data, then they were treated as a single activity. For example, if process measurements do not distinguish between compiling code and fixing any resulting compilation errors, then this is treated as a single process step.

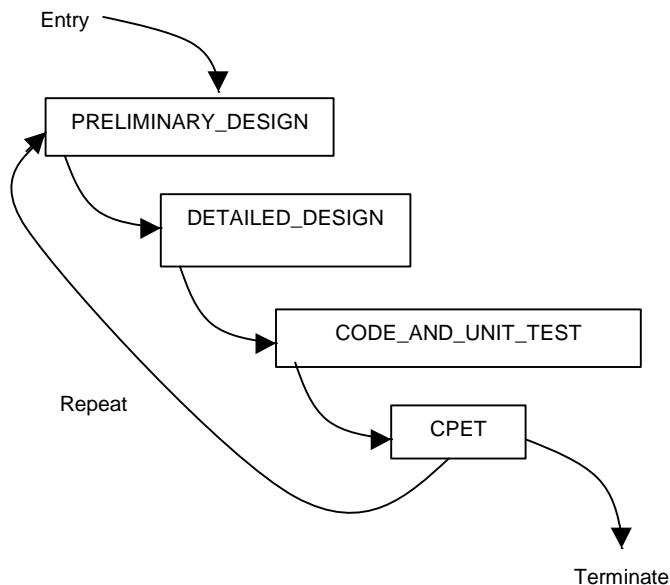


Figure 3 –TJU Project Life Cycle Process Model

The model can be viewed as a hierarchy of process steps. At the highest level, the model consists of four life cycle phases (See figure 3). **Preliminary Design** consists of software requirements analysis and high level software architectural design. It consists of designing CSCIs (Computer Software Configuration Items), CSCs (Computer Software Components) and identifying CSUs (Computer Software Units). **Detailed Design** consists of designing individual CSUs. **Code and Unit Test** is the implementation phase which consists of writing computer code, compiling units, testing individual units, and the first level of unit integration. **CPET** (Computer Program

Engineering Test) consists of internal integration and testing activities which are not formally witnessed by the customer.

Each of the four life cycle phases is decomposed in the model into several main tasks, and these in turn are decomposed into sub-tasks. The architecture of the simulation model replicates the architecture of the actual software development process in that some activities are executed sequentially, and some concurrently through the use of multiple entities. The overall project schedule (duration) for each simulation run is equal to the duration of the critical path. However, the critical path for the project results from the precedence relationships (flexible or fixed) that are encoded into this process architecture as well as the individual process step durations (determined based upon input contract constraints or expended effort) and process branching decisions that occur as each simulation run is executed.

Earned value is accrued by each activity as the project progresses. The amount of earned value assigned to each activity is based upon planned earned value allocations that are input before the simulation model is run. The model can help update earned value allocations by estimating the amount of effort associated with each sub-task.

When the simulation model is run, parameters for each execution are drawn from populations of data that were collected from the various project teams. Using multiple runs, the model provides the mean and variance of performance results that may be experienced from team to team. Hence, the results of the simulation are stochastic, capturing the inherent uncertainty associated with real-world development. The process model was developed using ExtendTM from Imagine Thatⁱ.

At SBMS Melbourne the model has been used to analyze several process change problems and to evaluate alternative process configurations for a significant new project bid proposal. The analyses have included flexible “what if” assessments of an initial review of a process change ideas.

One key distinction between the system dynamics and state-based simulation models as compared to a discrete event simulation model is the handling of process artifacts. In the discrete event simulation model, individual artifacts are represented and each artifact is able to retain distinct attributes. In other words, rather than representing a generic code or design artifact, in the discrete simulation model, we represent a particular code module with a certain size, complexity, number of defects, and so forth. It is clear that this added detail is highly compatible with the structure and output of the

metrics repository. The added detail also provides substantial scope for addressing a number of interesting questions such as: How does the process react if only 20% of the modules contain 80% of the defects? How does the process react if a few code modules are very large or highly complex rather than uniform throughout? What is the effect of a high or low level of fan-out of code modules from design? In short, this representation enables us to look at important questions related to core project management issues. Using the updated information from the repository enables improved accuracy in the predictions rather than basing the predictions upon initial project estimates of key model parameters (e.g. e.g. size and so forth). As will be discussed in section 5, this approach supports active planning and re-planning activities described earlier as part of the management controlling function.

4 THE CONTROL ACTIVITY

The purpose of this section is to illustrate the capability that can be achieved by linking the flexible metrics repository to the software process simulation model to provide real-time metrics combined with short-term performance prediction.

The control function is different from the planning function in terms of focus, timing, and scope. It involves the on-going management of a project. It compares the likely final state of the project (actual performance) with respect to cost, quality and schedule with the state called for by the original plan. Deviations are reported to the project manager who then may choose to take some specific action to bring the project back “in control”.

We introduce the concept of predicted project performance (i.e. predicted by the model) as being “in-control” or “out of control” – meaning the project “is” or “is not” adhering to the plan within a reasonable amount (reasonable bounds) for the performance measures under consideration (in our model these performance measures are cost, quality, and schedule). This is different from the definition typically used in statistical process control (SPC) where control limits are derived statistically from previous measurements of the process. In SPC, the control limits are determined independently of the desired outcome. We define outcome-based control limits (OBCLs) which are used as guides for assessing whether the project is “in-control” from a project management perspective. OBCLs can correspond to internal performance goals or can reflect contract performance requirements.

The decision as to whether a project is “out of control” requires a) constant monitoring of the current state of the project and b) an objective, accurate and meaningful way to compare the current state to the planned state.

Software process simulation models address this issue very well. Not only can process models identify changes that will have a significant impact to the project, they also can distinguish deviations from the plan that will *not* affect the project. For instance, although coding on a particular module may begin late, it may have no noticeable impact on the project if it is not on the critical path.

By incorporating up-to-date metrics data as described in Section 3 with the model described in section 4, estimated parameters become more accurate, the time frame for the estimate is reduced and more is known about the actual status of the project. In this mode, the model would predict the likely end of project cost, quality, and schedule performance. This performance would be compared to the outcome based control limits. If the project is within the OBCLs, confidence is increased that the current approach will achieve the desired performance targets.

On the other hand, if performance is outside of acceptable bounds, management is alerted that action needs to be taken. The project manager may have a large number of possible actions that could be taken to bring a project back into control. Information on which options are most likely to be successful, and their relative cost and risk is essential.

When a significant deviation between planned and actual behavior is identified, the project manager can take several steps. First, he can attempt to determine whether the deviation is significant. We propose the use of a computer simulation of the project to help predict the final outcome of the project, given the deviation from the plan. The result of the simulation will help the manager decide if the project is truly in trouble.

If the deviation suggests that the project may be in trouble the project manager can change aspects of the simulation representing various process steps and explore the results of process changes in response to the control deviation. Potential actions to bring the process back under control can be analyzed and compared for effectiveness, risk and cost.

5 CONCLUSIONS AND FUTURE WORK

The work presented here utilizes quantitative simulation models of the software development process in a new way that supports on-going management planning and control and supports corporate efforts for achieving levels 4 and 5 of the CMM. This work is closely tied to our previous process modeling research, which predicts the impact of

process changes in terms of cost, quality and schedule.

We have discussed the planning activity and shown that previous work in the area of software process modeling has been done at a sufficient level of detail using past project data to predict potential future performance. These detailed process models provide additional capabilities over cost estimation models due to their fine-grained depiction of the process. However, although this finer grained process depiction is necessary, it is not sufficient for supporting the control activity. Detailed and timely project data is an essential ingredient to support the control function.

The metrics repository provides a current record of the state of the project as it progresses through (or is "transformed by") the development process. This is accomplished by recording snapshots of project characteristics each time a *significant* "transformation" occurs. The repository attempts to characterize the state of the project at each snapshot by maintaining information about the transformation and the artifact(s) produced by the transformation. This information is intended to characterize particular "domains" associated with the transformation and the artifact. Project snapshots are used to develop timely model parameters using SQL queries.

Using current metrics and timely model parameters, process simulation models evaluate the current state of the project and predict the expected software project outcome. The predicted project outcome is checked to see whether it falls within the predetermined limits. If the expected performance is outside of these limits, management can be alerted to a potential problem and take corrective action. This is the goal of methods that are based on Statistical Process Control (SPC) but it cannot be achieved using individual metrics alone. The predictive model is then used to evaluate the outcomes of potential management decisions to bring the project back "in control" and help the project meet its pre-determined goals.

This approach is significant and directly 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.

In future work, we plan to develop a Process Tradeoff Analysis Testbed which will feature the flexible metrics repository and general software process simulation blocks which can be flexibly connected and configured to accommodate a variety of process types and variations.

The testbed will support rapid model development and configuration of a company's metrics data into the metrics repository.

ACKNOWLEDGEMENTS

This work has benefited from discussion, comments and participation by Bob Martin, Joe Vandeville, Dick Hotz and Dean Knudson. Their 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 for supporting this research effort.

REFERENCES

- 1 [Boehm 81] Boehm, "Software Engineering Economics", *Prentice Hall*, 1981.
- 2 [Boehm+ 95]B. Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," *Annals of Software Engineering*, vol. 1, pp. 57-94, 1995.
- 3 [Humphrey 1989] Humphrey, Watts, "Managing the Software Process", Addison Wesley, 1989, ISBN 0-201-18095-2
- 4 [Kellner 89A] M. Kellner, "Software Process Modeling : Value and Experience," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, SEI Technical Review 1989.
- 5 [Kellner 89B] M. Kellner, "Software Process Modeling Experience," presented at The 11th International Conference on Software Engineering, Pittsburgh, Pennsylvania, USA, 1989.
- 6 [Kellner 90] M. I. Kellner, "Experience with Enactable Software Process Models," presented at The Fifth International Conference on The Software Process, Kennebunkport, Maine, USA, 1990.
- 7 [KeMR 99] Marc Kellner, Raymond Madachy, and David Raffo, "Software Process Simulation Modeling: Why? What? How?" *Journal of Systems and Software*, Forthcoming.

- 8 [KH 88] M. I. Kellner and G. A. Hansen, "Software Process Modeling," Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-88-TR-9, May 1988.
- 9 [KH 89] M. I. Kellner and G. A. Hansen, "Software Process Modeling : A Case Study," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences-Vol II -Software Track, IEEE Computer Society Press, 1989, pages 175-188.
- 10 [KoMR 98] D. Kocaoglu, R. Martin, and D. Raffo, "Moving Toward a Unified Continuous and Discrete Event Simulation Model for Software Development", Proceedings of the Silver Falls International Workshop on Software Process Simulation Modeling (ProSim'98), Held in Silver Falls, Oregon, June 22-24, 1998.
- 11 [Madachy 94] Madachy, R., "A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment", Ph.D Dissertation, Department of Industrial and Systems Engineering, University of Southern California, December 1994.
- 12 [MIO 87] J. D. Musa, A. Iannino, and K. Okumoto, "Software Reliability – Measurement, Prediction, Application", McGraw Hill Book Company, 1987
- 13 [Musa 75] J. D. Musa, "A theory of Software Reliability and It's Application", IEEE Transactions on Software Engineering, 1(3):312-327, Sept. 1975.
- 14 [Paulk+. 93] Paulk, M.C., Curtis, W., Chrissis, M.B., Weber, C.V., "Capability Maturity Model for Software, Version 1.1", Technical Report SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA., February 1993.
- 15 [Putnam 78] L. H. Putnam, "General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transaction on Software Engineering*, vol. SE 4,4, pp. 345-361, 1978.
- 16 [Putnam 80] L. H. Putnam, *Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers*. New York: The Institute of Electrical Engineers, Inc. 1980.
- 17 [Raffo 96] 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.
- 18 [RK 95] "Using Quantitative Process Modeling to Forecast the Impact of Potential Software Process Improvements", Proceedings of the 10 th International Forum on COCOMO and Software Cost Modeling, Held in Pittsburgh, Pennsylvania, October 1995.
- 19 [RK 96] D. M. Raffo and M. I. Kellner, "Field Study Results Using the Process Tradeoff Analysis Method",. Proceedings of the 1996 Software Engineering Process Group Conference, Held in Atlantic City, New Jersey, May 20-23, 1996.
- 20 [RVM 99] D. M. Raffo, J. V. Vandeville, and R. Martin, "Software Process Simulation to Achieve Higher CMM Levels", Journal of Systems and Software, forthcoming.
- 21 [Scacchi 99] W. Scacchi, "Experience with Software Process Simulation and Modeling", Journal of Systems and Software, forthcoming.
- 22 [Trivedi 82] K.S. Trivedi, "Probability and Statistics with Reliability, Queuing, and Computer Science Applications", Prentice Hall, Inc., Englewood Cliffs, NJ, 1982.
- 23 [Tvedt 95] Tvedt, J., "A System Dynamics Model of the Software Inspection Process", Technical Report TR-95-007, Computer Science and Engineering Department, Arizona State University, Tempe, Arizona, 1995.

ⁱ Extend is a registered trademark of Imagine That, Inc., San Francisco, California (http://www.imagine_that.com)