Paper appeared in: Martin, R. H. and D. M. Raffo, "A Model of the Software Development Process Using Both Continuous and Discrete Models", International Journal of Software Process Improvement and Practice

A Model of the Software Development Process Using Both Continuous and Discrete Models

Robert Martin
bobm4@teleport.com

Dr. David Raffo
davidr@sba.pdx.edu

Portland State University

Portland, Oregon

Abstract

Software process models have been simulated using system dynamics and discrete modeling paradigms. System dynamics models describe the interaction between project factors, but do not easily represent queues and discrete process steps. On the other hand, discrete event models describe process steps, but may not have enough events to represent feedback loops accurately. We develop a combined model that represents the software development process as a series of discrete process steps executed in a continuously varying project environment. We demonstrate the feasibility of this model by combining a discrete event model of the ISPW6 software process example with the system dynamics model developed by Abdel-Hamid and Madnick [Abdel-Hamid and Madnick 91]. The combination of these two modeling paradigms creates the opportunity to examine new problems and issues that are highly relevant to software project managers.

1. Introduction

Over the past few decades, many authors have documented the difficulty encountered in managing software projects. In 1980, Putnam [Putnam 80] claimed that cost overruns of two to three hundred percent were typical. DeMarco [DeMarco 82] stated that fifteen percent of software projects failed to deliver anything and that projects with cost overruns of less than thirty percent were considered successful. Recently, Shein [Shein 96] reported that only sixteen percent of software projects were successful.

Attempts to understand the software development process have concentrated on identifying factors that influence the process and relating these factors to estimates of time and effort. In the COCOMO model, Boehm [Boehm 81] identified 15 factors that influenced the estimates. Jones [Jones 86] identified 20 factors in his model.

While these models have achieved some success in predicting the duration and effort

of the project, they do not consider dynamic interaction of factors inherent in the process. Moreover, these models do not capture the details pertaining to the actual process that is followed by the project. The complexity and diversity of software development projects have caused researchers to turn to simulation models.

Software process simulation models have been used to capture dynamic interactions inherent in software development projects as well as process level issues. Two main modeling paradigms found in the literature are system dynamics (continuous simulation) and discrete event simulation. System dynamics models describe the interaction between project factors, but do not easily represent queues and discrete process steps. Discrete event models describe process steps, but may not have enough events to represent feedback loops accurately. In this paper, we present a combined model that represents the software development process as a series of discrete process steps executed in a continuously varying project environment. We demonstrate the feasibility of this model by combining a discrete event model of the ISPW6 software process example with the system dynamics model developed by Abdel-Hamid and Madnick [Abdel-Hamid and Madnick 91]. The combination of these two modeling paradigms creates the opportunity to examine new problems and issues that are highly relevant to software project managers.

2. Background

Discrete models have been developed using tools like GPSS [Schriber 74], SLAM [Pritsker 79], and SIMAN [Pegden 90]. These models represent the development process as a series of entities flowing through a sequence of activities. In discrete event simulation, activities schedule future events. System Dynamics models are time-based, describing the system in terms of continuous functions that advance by some fixed time step. We will examine both types of models in more detail, looking at how each model presents the software development process.

2.1 Discrete Models

Advantages

We use the term “discrete models” to refer to the group of models that advance time because of a discrete event. Since nothing of importance to the model happens between event times, the model can advance the simulation clock from event to event with no loss of information. This type of model is efficient and particularly appealing when the process is viewed as a sequence of activities. Discrete models are often used to model a manufacturing line where items or “entities” move from station to station and have processing done at each station. Discrete models easily represent queues and can delay processing at an activity if resources are not available. In addition, 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 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 for each activity.

In software development, there has been a long history of viewing the development process as a sequence of discrete activities. To enable more precise management, process models like the waterfall model or the spiral model have been proposed as a generic set of activities that can be used to track the progress of the development effort. Tausworthe’s Work Breakdown Structure approach [Tausworthe 80] is a clear example of a systematic description of the development process as a sequence of discrete activities. More recently, the Capability Maturity Model [Paulk 95] stresses the importance of a description of the process as a detailed sequence of repeatable activities.

Because a discrete model allows each entity to contain unique values for attributes, the model can capture the variation in code difficulty and programmer capability. The effects of increased complexity on effort and error rates or the impact of different

programmer capabilities on coding duration may be computed from attributes in a discrete model.

Values for attributes may be constant or may be sampled from distributions. The stochastic nature of the variables captures the uncertainty present in measuring the attributes.

Finally, a discrete model allows us to represent 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.

Disadvantages

As mentioned above, a discrete model advances time only when an event occurs. This means that continuously changing variables are only updated at the event times. While the time between discrete events may be days or weeks in a software project, the model of the 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

In addition, because discrete models are based on the idea of a sequence of activities, it may be awkward to represent simultaneous activities. While activities can occur in parallel, it is difficult to represent the idea of an entity in two activities simultaneously. Imagine a code module in which some parts are in coding while other parts are in unit test. To capture this in a discrete model, we are forced to model sub-components of the module so that each sub-component can be in only one activity at a time.

2.2 System Dynamics Models

System dynamics modeling tools such as STELLA [HPS 98], POWER-SIM [Powersim 95] and DYNAMO [Richardson 81] represent the development process as a system of differential equations. System dynamics were first applied to project management by Roberts [Roberts 64]. Richardson [Richardson 81] gave a detailed project model as an example in his book. Abdel-Hamid and Madnick [Abdel-Hamid and Madnick 91] extended this work to software projects in 1990. Later work by Madachy [Madachy 94] modeled a more detailed development process. Tvedt and Collefello [Tvedt 95] used system dynamics to model the software inspection process step. Abdel-Hamid and Madnick's model was able to reproduce several well-known project characteristics, suggesting that much of project behavior was a consequence of relationships between factors.

Advantages

System dynamics models describe the system in terms of "flows"' that accumulate in various "levels". The flows can be dynamic functions or can be the consequence of other "auxiliary" variables. As the simulation advances time in small evenly spaced increments, it computes the changes in levels and flow rates. For example, the error generation rate may be treated as a "flow" and the current number of errors could be treated as a "level". This allows the model to capture the stability or instability of feedback loops. A system dynamics model can be valuable in finding the levels where a model can become unstable, or in predicting the unanticipated side effects of a change in a system variable.

In a software project, experience is recognized as a factor in productivity. Since the experience level changes continuously as the project progresses, a system dynamics simulation can model the resulting change in productivity. At the same time, fatigue, schedule pressure and attrition will affect productivity, but in an adverse manner. Continuously simulating the interaction of these variables can produce a dynamic estimate of net productivity.

Disadvantages

While the system dynamics model is an excellent way to describe the behavior of project variables, it is a more difficult way to describe process steps. While it is possible to represent discrete activities in a system dynamics model, the nature of the tool implies that all levels change at every time interval. If the process contains sequential activities, some mechanism must be added to prevent all activities from executing at once. For example, if we modeled the software process as define, design, code and test activities, as soon as some code was defined, design would start. If we wanted to model a process that completed all design work before coding started, we would have to create an explicit mechanism to control the sequencing.

Abdel-Hamid and Madnick integrate a single "software development rate" to model tasks being completed. Their model implies that "tasks" are developed, verified by Quality Assurance (QA), and tested, but no explicit process steps are described.

Because system dynamics models deal with flows and levels, there are no individual entities and thus no entity attributes. For a software process model, this means that all modules and all developers are equal. If we wanted to model the effect of a few error prone code units on the development process, we would not be able to specify which units were error-prone.

Finally, a system dynamics model does not easily allow us to model the uncertainty inherent in our estimates of model parameters. If we estimate that coding each module will take from 3 to 6 weeks, we would have to translate that estimate into equivalent coding rates. A discrete model could sample from a distribution using a different time for each module. The system dynamics model either must sample the rate at each time step or must use the same rate for each model run.

2.3 Need for a combined model

Many characteristics of software development projects influence the choice of modeling tools. Large projects are often broken down into a series of inter-related tasks that consume resources as they are completed. The duration and effort of the tasks are seldom predicted without some uncertainty. Code modules may vary in size and complexity and programmer's experience and ability may differ. The project environment is often dynamic. Staffing, experience, schedule pressure and fatigue all may vary over time.

It would be desirable to use a continuous simulation tool to model the dynamic environment, and a discrete simulation tool to model tasks and resources. A combined model would allow investigation of the effects of discrete resource changes on continuously varying productivity. In this way, continuously changing error rates could influence the duration of discrete code inspection tasks.

In the next section, we describe a model of the software development process that combines the advantages of a system dynamics model with the precision and detail of a discrete event process model. We also discuss several key issues related to integration of the two modeling paradigms. The integration presents a number of interesting issues and insights and shows the opportunity for addressing important new management questions.

3. Issues with Model Integration

In order to demonstrate the feasibility of hybrid model, we combined the system dynamics model developed by Abdel-Hamid and Madnick [Abdel-Hamid and Madnick 91] and a discrete event model based on the ISPW-6 Software Process Example. [Kellner et al. 91]

These models differ in their assumptions and levels of abstraction. In the following sections, we discuss these differences and how they were reconciled in a combined model.

3.1 Tasks

In order to develop a combined system

dynamics and discrete event simulation model, we need to reconcile the differences in how each paradigm represents tasks or the development work to be done.

Many existing abstractions of the software development process that can be found in the literature represent the process as a series of tasks [Thayer 81, Reifer 79, and Tausworthe 80]. These tasks represent activities that are performed on software components.

A system dynamics model describes "flows" or rates that vary over time and change "levels".
Thus, sequential activities in a process model must be represented as either a flow or a level. A task is represented by the level of its outputs, e.g. modules coded or errors detected. The levels are adjusted every time increment by integrating flows (e.g. coding rate or inspection rate). Because every flow is integrated every time increment, additional mechanisms must be used to create sequential behavior. Abdel-Hamid and Madnick use a third order delay to cause the QA rate to lag behind the development rate. Testing is forced to occur late in the project by specifying the manpower allocation externally. Within software production, the nominal error rate function, an input variable, changes halfway through the project to reflect a lower error rate during coding. While these mechanisms produce valid behavior, they require that some of the important project behavior be specified a priori. To combine the models, we must determine the process implied by these mechanisms.

These Abdel-Hamid and Madnick model characteristics imply that a series of identically sized (in terms of effort) components are designed, sent to quality assurance, coded, sent to quality assurance again and then tested. Design and coding are sequential, but QA proceeds in parallel (slightly delayed) controlled by manpower allocation. Rework must occur after each task is processed by QA, because rework manpower is a function of the number of detected errors which are produced by QA. While this process is implied by the model assumptions, it is not directly represented or readily apparent in the model. Quantities like number of errors or amount of rework are not explicitly associated with tasks, but are dynamic variables driven by the development rate. Any inference about the amount of rework required by a specific task must be done in the interpretation of the model results. Thus, a task is defined simply as a unit of code, but is used throughout by the model mechanisms as a representation of sequential activities. In the combined model, we must assure that the activities we explicitly represent in the discrete portion model are consistent with the implied activities of the continuous portion.

In contrast, in a discrete event or state-based model, a task is a discrete component of the system. This task can have different size, complexity, number of defects and so forth. The task is then transformed by the various software development process steps (e.g. design, coding, inspections, testing, and rework among others).

In this work, our objective was to combine two existing models to demonstrate the feasibility of developing a combined system dynamics and discrete event simulation model. For the discrete model, we used the model of the ISPW-6 process example created by Raffo and Kellner [Raffo 96; Raffo and Kellner 99], which is an event driven model of the activities performed during the software development process. Equations for each process step calculate activity effort, duration and defects from user supplied distributions. In keeping with the definition of the ISPW-6 example, the model describes the activity necessary to make a change to one unit of code. Estimates for effort are obtained by assuming a constant number of staff for each activity. The sampled duration is then multiplied by the staff to determine effort. The number of errors corrected affects the duration of some activities.

While this abstraction explicitly describes process steps, it does not address multiple units of code, the effect of resource constraints, productivity, variations in error rates and manpower allocation.

In order to resolve the differences between

the Abdel-Hamid and Madnick model [Abdel-Hamid and Madnick [91] and the discrete event ISPW-6 simulation model developed by Raffo and Kellner [Raffo 96; Raffo and Kellner 99], we assumed that we are developing a number of components that are of equal size and effort. We defined a "task" as a process step or activity performed on a component or item. Each activity in the process is assigned an earned value. This value represents the percentage of the total project effort allocated to a given process step for all components. Once the user has specified the total effort for the project and the number of components, we compute the effort required for a component in each process step. We use the distributions defined by Raffo to estimate the duration of each step. From the effort and duration, we then compute the expected number of staff required. Computing the needed staff simply lets us avoid allocating all resources to the first task.

The combined model maintains separate resource pools for development, QA, rework, and test engineers. The total number of staff in each pool varies continuously as staffing levels and manpower allocation decisions change due to a variety of factors during the project. As each task occurs, the model attempts to allocate the needed staff from the appropriate resource pool. If the model is unable to allocate the entire number of staff needed, it takes whatever is available. Staff are re-allocated at each continuous time step. It is possible to assign different priorities to components. The allocated staff and the current productivity are integrated for each time step until we reach the required work load (RWL) for a given activity for each component. This mechanism allows us to capture the effects of resource constraints while considering fluctuations in staffing levels and productivity. Thus, as each time increment passes, work is completed at a rate that is a function of the allocated staff ($A_i(t)$) and the current productivity level PA(t). The duration for an activity (D_i) is determined when the total amount of accumulated work ($RWLA_i$) is equal to the estimated required work load ($RWLE_i$) for a given activity for a given component. That is we integrate until

$$RWLE_i = RWLA_i$$

$$RWLE_i = E_{V_i} * E_T * P_p(Estimated\_at\_beginning)$$

$$RWLA_i = \int_0^{D_i} A_i(t) * P_A(t) dt(computed)$$

$RWLE_i = Estimated\_\mathrm{Re}quired\_work\_load\_for\_activity\_i(LOC$

$RWLA_i = Accumulated\_work\_load\_for\_activity\_i(LOC)$

$A_i(t) = Allocated\_staff\_at\_time\_t\_in\_Task\_i(staff)$

$P_A(t) = Actual\_\mathrm{Pr}oductivity\_at\_time\_t(LOC\_per\_staff\_day)$

$P_P = Planned\_\mathrm{Pr}oductivity(LOC\_per\_staff\_day)$

$E_{Vi} = Earned\_Value\_at\_task\_i(Percent)$

$E_T = Total\_Effort(staff\_days)$

$D_i = Sampled\_duration\_for\_task\_i(days)$

Note that when actual productivity equals planned productivity and allocated staff equals the expected staff, duration will equal the sampled duration, since the expected staff was based on expected effort divided by sampled duration. If actual productivity varies or resources are constrained, the duration will be different from the sampled duration.

As each activity is completed for each component, cumulative earned value is computed. This total influences motivation, staffing levels, error rates and schedule.

Quality

The quality model describes the way that errors are generated, detected and corrected during the development process. The hybrid quality model illustrates the model refinement needed to combine a system dynamics model with a discrete event model.

The system dynamics model describes the quality model in terms of a set of error rates. The error rates vary with schedule pressure, experience and project progress. They are integrated to determine the number of errors present at any time in the project.

A discrete model associates error generation and correction with specific tasks occurring at discrete times. Error rates used in these tasks are usually fixed throughout the task.

In a hybrid model, we can isolate the rate integration to specific activities. For example, the error generation rate describes the number of errors injected into the code. This rate will vary continuously due to changes in experience, schedule pressure and project progress. We would like the hybrid model to only increase generated errors during tasks like design or coding. We accomplish this by only integrating the error generation rate when an item is in the design or coding step. The nominal error generation rate per KLOC (thousand lines of code) is specified as an input function in the system dynamics portion of the model. This nominal rate is modified by functions that account for the effect of experience and schedule pressure. This is converted to errors per item. The error rate is integrated while the item is in the design or coding activity. The duration of the activity is determined by the method described in the sub-section on tasks.
If we let

$$E_G(t) = Errors\_Generated\_per\_KLOC$$
$$D_i = task\_duration\_for\_task\_i$$
$$T = Size\_in\_LOC\_for\_all\_items$$
$$n = Number\_of\_items$$
$$E_{Gi} = Errors\_generated\_in\_task\_i\_per\_item$$

Then the number of errors generated in a design or coding task is

$$E_{Gi} = \int_0^{D_i} \frac{E_G(t) * T}{n * 1000} dt$$

If the error generation function is constant during the task, the task is fully staffed, and productivity is normal, this integral reduces to the error generation rate times the item size.

When the error generation function fluctuates, the number of errors generated by the task will change. We also note that since the duration is sampled from a distribution, the number of errors will be a stochastic variable, influenced by resource constraints and productivity.

Detected errors are also based on detection rates supplied by the system dynamics portion of the model. The error detection rate is calculated as the nominal QA manpower required per error and is then modified by functions based on experience and error density. The discrete event model provides an estimate of the duration of the inspection, sampled from a normal distribution. Then,

$$E_{Dj} = \min\left(\int_0^{D_i} \frac{A(t)}{Q(t)} dt, E_{Gj} \right)$$

$$Q(t) = QA\_staff\_days\_per\_error$$
$$A_i(t) = Allocated\_staff\_for\_task\_i$$
$$D_i = Duration\_of\_the\_error\_inspection\_task$$
$$E_{Gj} = Errors\_generated\_in\_item\_j$$
$$E_{Dj} = Errors\_\mathrm{det}ected\_in\_item\_j$$

This approach only integrates the detection function during an inspection or unit test. The number of detected errors will be change if resources are constrained or if the QA manpower per error changes during the inspection.

We assume that all detected errors will be corrected, but that some new errors will be generated during the correction process. The nominal rework rate from the system dynamics model is modified to account for losses due to motivation and communication. The rework rate is given in rework person-days per error. Let

$$R(t) = \mathrm{Re}\, work\_effort\_per\_error$$
$$A_i(t) = Staff\_allocated\_to\_task\_i\_at\_t$$
$$E_{Di} = Number\_of\_errors\_found\_in\_item\_j$$
$$D_i = task\_duration\_for\_task\_i$$

Then, we calculate the duration of the rework task by

$$E_{Dj} = \int_0^{D_j} \frac{A_i(t)}{R(t)} dt$$

That is, using allocated staff and the re-work rate, we can calculate the errors re-worked each time step. When all detected errors have been reworked, we are done. During this interval, we decrease the generated errors as errors are corrected and we add a percentage of the re-worked errors to the generated errors to account for bad fixes.

4. Implementation

4.1 Combining Continuous and Discrete Clocks

We developed a hybrid model using EXTEND™, from ImagineThat, Inc. The EXTEND simulation software allows the user to develop customized blocks to implement unique functionality.

Because an event driven model uses a clock that is advanced when something happens, and a continuous model advances time at a small steady increment, we created an executive that can drive the continuous blocks at the required time increment while preserving the discrete scheduling. While implementations differ, a discrete event simulator usually maintains a next event queue pointing toward the next scheduled activity. As the activity is performed, new events may be scheduled and placed on the queue. When the activity is completed, the system clock is advanced to the time of the next event. To support hybrid models, we modified the scheduler to only advance the clock by the specified delta time. At each delta time increment, we then integrate all necessary equations until the next scheduled event time is reached. This approach is an approximation that works well if the delta time is sufficiently small. In order to accommodate the computational overhead of this approach, we then re-wrote the EXTEND block libraries to improve efficiency.

4.2 An Example Model

Figure 1 shows an overview of the combined model. The continuous portion of the model is well documented in Abdel-Hamid and Madnick's book [Abdel-Hamid and Madnick 91]. Blocks were created to implement the sectors of their model. Information is passed between blocks using the connections shown. We provide a brief description of each block below:

- HR – Human Resources. This block describes the changes in the workforce due to hiring, training, transfers, and attrition.
- MP ALLOC – Manpower Allocation. The

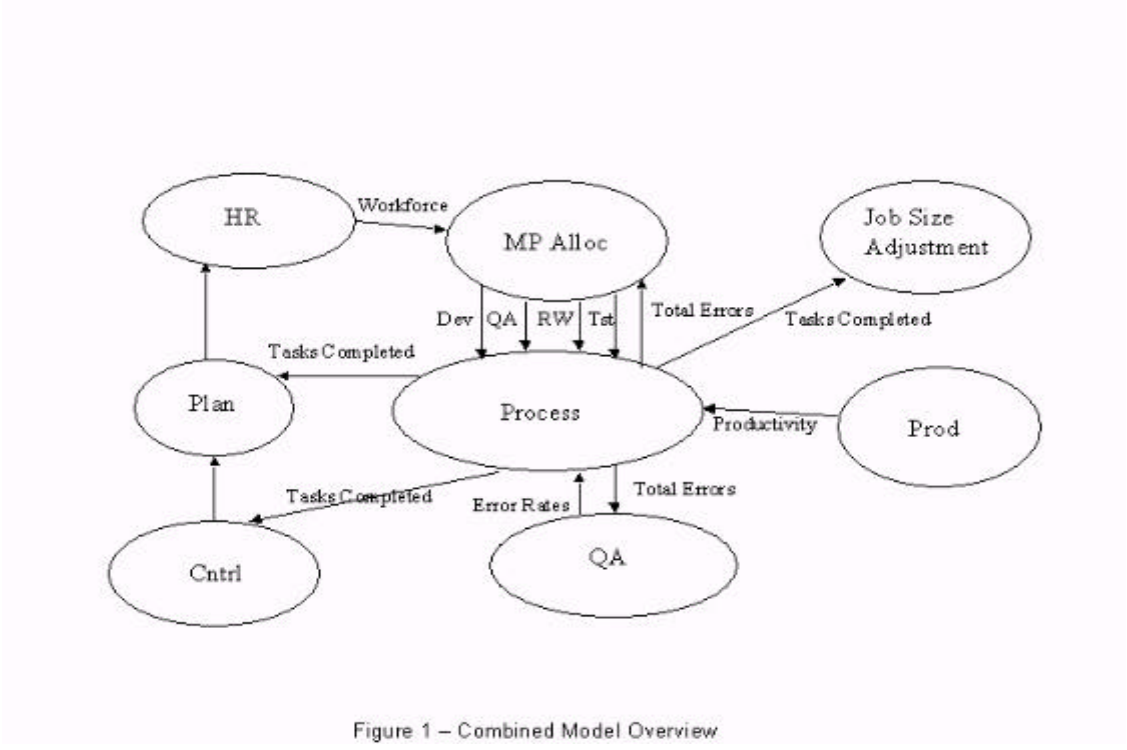

Figure 1 – Combined Model Overview

total manpower resource may be allocated dynamically to development, quality assurance, rework, and system test.

- JOB SIZE ADJ – Adjusts the size of the project to include new work that is discovered during the project.
- PROD – computes productivity as the dynamic variable based on experience, exhaustion, motivation, and communication losses.
- CNTRL – adjusts the scheduled completion date.
- PLAN – uses the completion date and amount of work remaining to create the target workforce level.
- QA – models error generation and rework rates.
- SYS TEST – models error detection and removal in system test. This sector is disabled in this version of the hybrid model.

The sector labeled "S/W Dev Proc" contains the discrete event process model. This block contains the individual process steps required to implement the ISPW-6 process example. This block is expanded in Figure 2. Double line connections show the way that items travel through the model. Single line connections show information flow.

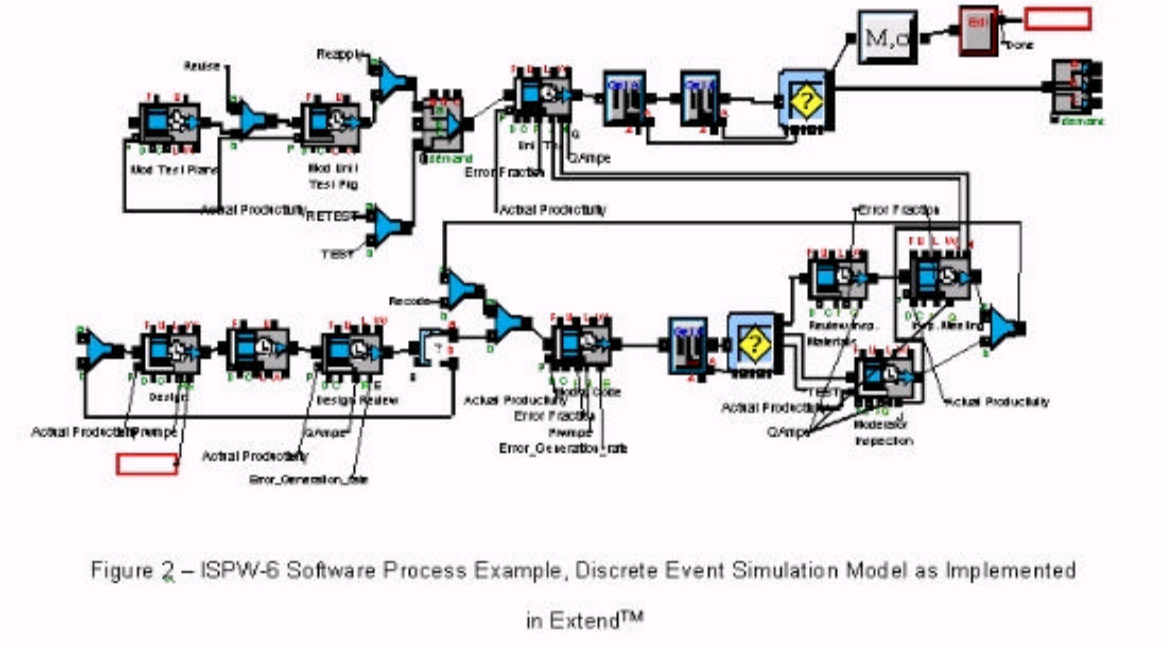

Figure 2 – ISPW-6 Software Process Example, Discrete Event Simulation Model as Implemented in Extend™

Process steps are shown for design, design review, coding, inspections, unit test plan preparation, and unit test. Project size, which is defined in constants on the top-level is divided into a number of components. Each of these components moves through the process steps. If the component fails a design review, it is sent back to design for error correction. Similarly, if a component fails in inspection or unit test, it is sent back to coding for re-work.

The allocated manpower levels from the "MP Alloc" block are passed to the discrete model where they may be allocated to process steps as needed. The total effort expended in all process steps, and the errors detected by all process steps are passed out of the block to be used in the system dynamics portion of the model.

5. Model Output

A hybrid model allows us to examine phenomena that are not reproducible in the continuous or discrete models. To illustrate this point, consider the way that manpower is utilized during the project. A continuous model can show us how the manpower

levels may vary because of changes in desired workforce, hiring delays and allocation decisions. A discrete model may show us that available manpower may be wasted if process bottlenecks cause staff to be idle waiting for components to become available.

Figure 3 below shows the manpower utilization from a typical run. The top line is the total available manpower during the project. The manpower varies as hiring, training and attrition vary continuously. The lower line shows the actual manpower allocated to tasks during the project. The lower line never quite reaches the upper line because some manpower is diverted to training. Gaps in the lower line show those times when manpower was available, but could not be allocated because no work was available.

duration.

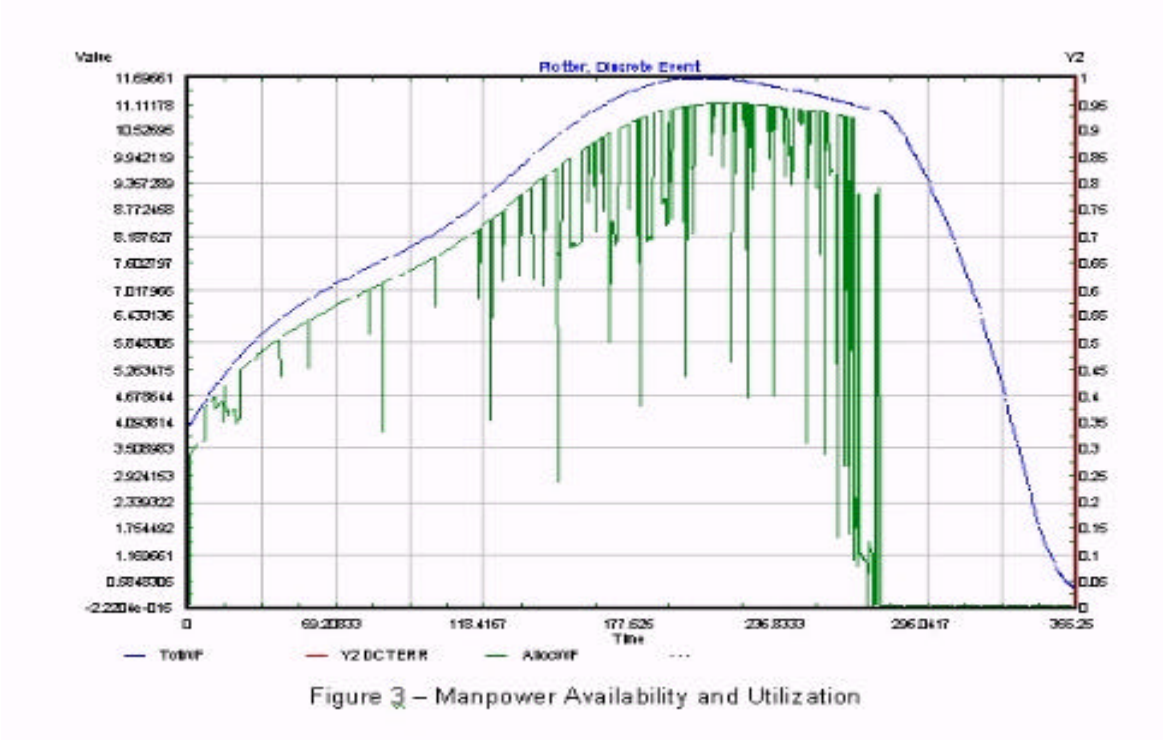

Figure 3 – Manpower Availability and Utilization

To test possible solutions to the manpower utilization, we may either experiment with changes to input variables that characterize the project or we might add or delete process steps. For example, since the model dynamically allocates manpower to QA based on project policy, insufficient QA manpower can leave development staff waiting for tasks being inspected. The model could be used to minimize this idle time by evaluating changes to the allocation policy giving more staff to QA. We could also attempt to remove the QA bottleneck by removing inspection steps in the discrete model, but the simulation would detect the increase in errors, allocate more effort to re-work, and possibly end up with increased

The ability to easily evaluate both types of solutions is a unique contribution of hybrid models.

6. Conclusions and Future Work

Previous models developed to predict the performance of software development operations concentrated on identifying factors that influenced the process and related them to effort and schedule. Simulation approaches capture the dynamic and uncertain nature of software development activities while providing improved insight into the process.

This paper describes the strengths and weaknesses of discrete and continuous simulation models. Ideally, we would like to use a discrete simulation tool to model tasks and resources, a continuous simulation tool to model the environment.

We demonstrate the feasibility of a combined model, by constructing a hybrid model that combines the continuous model proposed by Abdel-Hamid and Madnick with a discrete event model of the ISPW-6 process example. We then utilize this model to examine dynamic changes in project staffing and developer utilization levels incorporating factors that cannot be examined using either modeling paradigm individually.

Biographies

Robert Martin

Mr. Martin is a Ph.D. candidate at Portland State University. He received his M.S. in Engineering Management from Portland State in 1991 and his B.S. in Mathematics from the University of Central Florida in 1971. He has over 30 years experience in software management and development. He is President of Software Management Consulting, a Portland, Oregon based company that develops simulation models as components of decision support systems.

Dr David Raffo

David M. Raffo received his Ph.D. in Operations Management from Carnegie Mellon University. His current research is in the area of strategic software process management and software process simulation modeling. Raffo has twenty-five refereed publications in the software engineering and management fields. He has received research grants from National Science Foundation, IBM, Tektronix, Northrop-Grumman, and the Software Engineering Research Center (SERC). Currently, Dr. Raffo is an Assistant Professor of Operations Management and Information Systems in the School of Business Administration at Portland State University. He is Co-Director of Portland State University's Center for Software Process Improvement and Modeling.

Bibliography

[Abdel-Hamid and Madnick 91] Abdel-Hamid , T. and Madnick, S., "*Software Project Dynamics: An Integrated Approach*", Prentice-Hall software Series, 1991, ISBN 0-13-822040-9.

[Armenise 92] Armenise, P., Bandinelli, S., Ghezzi, C., Morzenti, A., "Software Processes Representation Languages: Survey and Assessment, *Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering*, Capri (Italy), June 1992.

[Boehm 81] Boehm, B. "*Software Engineering Economics*", Prentice-Hall, 1981, ISBN 0-13-822122-7.

[Davis 94] Davis, A.M., Sitaram, P., "A Concurrent Process Model of Software Development", Software Engineering Notes, ACM SIGSOFT, Vol. 19, no.2, pp. 38-51.

[DeMarco 82] DeMarco, T., "*Controlling Software Projects*", Yourden Press, 1982, ISBN 0-917072-32-4.

[Harel 90] Harel, D., et al. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, April 1990.

[HPS 98] High Performance Systems Inc. 45 Lyme Road Hanover, NH, 03755.

[Humphrey 89] Humphrey, W.S., and Kellner, M.I., "Software Process Modeling: Principles of Entity Process Models", *Proceedings of the 11th International Conference on Software Engineering*, IEEE, May 1989, pp. 331-342.

[Jones 86] Jones, C. "*Programming Productivity*", McGraw-Hill Book Company, 1986, ISBN 0-07-032811-0.

[Kellner 89] Kellner, Marc I. *Software Process Modeling: Value and Experience*, In SEI Technical Review, Pittsburg, Pa. :Software Engineering Institute, Carnige Mellon University, 1989, pp 23-54.

[Kellner et al. 91] Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterweil, L.J., Penedo, M.H., Rombach, H.D., "ISPW-6 Software Process Example", *Proceedings of the 6th International Software Process Workshop: Support for the Software Process*, IEEE Computer Society Press, 1991.

[Klingener 96] Klingener, J.F.,"Programming Combined Discrete-Continuous Simulation Models for Performance", Proceedings of the 1996 Winter Simulation Conference, Coronado California, December 8-11, 1996.

[Madachy 94] 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.

[Paulk 95] Paulk et al, *The Capability maturity Model: Guidelines for Improving the Software Process*, The SEI Series in Software Engineering, Addison-Wesley, 1995.

[Pegden 90] Pegden, C. D., Shannon, R.E., Sadowski, R.P*., Introduction to Simulation using SIMAN*, McGraw-Hill, 1990, ISBN0-07-049217-4.

[Powersim 1995] Powersim Corporation, 1175 Herndon Parkway, Suite 600, Herndon, VA, 20170.

[Pritsker 79] Pritsker, A. Alan B., and Pegden, C. D., *Introduction to Simulation and SLAM*, John-Wiley & Sons, 1979 ISBN 0-470-26588-4.

[Putnam 80] Putnam, Lawrence H., *Software Cost Estimating and Life-Cycle Control: Getting the software Numbers, A Tutorial for COMPSAC '80*, IEEE Computer Society's Fourth International Computer Software and Applications Conference, Oct 27-31, 1980.

[Raffo 96] Raffo, D.M., "*Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance*", Ph.D. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, 1995.

[Raffo and Kellner 99] Raffo, D. M and Kellner, M. I., "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.

[Reifer 79] Reifer, D.J., "The Nature of Software Management: A Primer" Tutorial: Software Management. Edited by Donald Reifer. IEEE Computer Society, 1979.

[Roberts 64] Roberts, E.B., *The Dynamics of Research and Development.* Cambridge, Massachusetts, 1964.

[Richardson 81] Richardson, G.P., and Pugh, A.L., *Introduction to System Dynamics Modeling with DYNAMO*, The MIT Press, Cambridge, Massachusetts, 1981, ISBN 0-262-18102-9.

[Schriber 74], Schriber, T., *Simulation Using GPSS*, John Wiley, 1974.

[Shein 96] Shein, E., "Process Patrol" PC Week, March 6, 1996, pp. 15-19.

[Tausworthe 80] Tausworthe, R. C., "The Work Breakdown Structure in Software Project Management", *The Journal of Systems and Software*, Vol. 1, 1980, pp. 181-186.

[Thayer , R.H., et al., "Major Issues in Software Engineering Project Management*. IEEE Transactions on Software Engineering*, Vol. SE-7, No. 4, (July, 1981).

[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