Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

AGENT-BASED SIMULATION FOR SOFTWARE PROJECT PLANNING

David Joslin
William Poole

CSSE Department
Seattle University
Seattle, WA 98122, U.S.A.

ABSTRACT

Estimates of task duration and resource requirements in software engineering are notoriously inaccurate, and as a result effective project management often must be very dynamic. In response to new information or revised estimates, it may be necessary to reassign resources, cancel optional tasks, etc. Project management tools that make projections while treating decisions about tasks and resource assignments as static will not yield realistic results. In this paper we describe some preliminary attempts to adapt a simulation-based planning algorithm developed for planning experimental activities of Mars rovers to the problem of planning for software project management. Simulation techniques offer the potential for modeling the way agents behave in project development, and the way a manager might adapt the project plan based on the project status at future points, resulting in a tool that more accurately reflects the realities of software project management.

1 INTRODUCTION

Effective project management is difficult and complex. Tasks of various types must be assigned to resources (including human resources) with different characteristics, taking complex dependencies, constraints and uncertainties into account, attempting to meet goals related to costs and time. The complexity and nature of project management make the use of simulation techniques an obvious choice. We can simulate the characteristics of the agents involved, the effects of external events, uncertainty about the duration of tasks, and so on.

The most straightforward application of simulation to project management is to simulate the application of a project plan to see how uncertainties about task duration, etc., affect the outcome. No matter how carefully we simulate the various elements involved, however, the simulation will be unrealistic if the plan is static.

Project management must be very dynamic because projects are very dynamic. An effective manager evaluates new information as it becomes available and, when necessary, modifies the current plan to adapt. Simulation of the consequences of an initial set of decisions, treating those decisions as static, is therefore unrealistic. At the same time, it is infeasible to anticipate every possible combination of future events and plan responses in advance. We can't simulate dynamic project management this way.

Our objective in this research is to extend some Artificial Intelligence (AI) planning techniques that have been recently developed for planning activities for Mars rovers (Joslin, Frank, Jónsson, and Smith 2005). A key consideration there was the desire to increase the autonomy of Mars rovers, or other future planetary rovers. In that respect the two problem domains are very different. If these algorithms can be applied to project management, their purpose won't be to take control away from the manager but rather to assist the manager in decision-making.

What the domains have in common is a high degree of uncertainty, and the need to adapt any plan to current circumstances as new information becomes available. In both cases it is infeasible to anticipate every possible future combination of events. For the rover planning work, we search a space of plan *strategies*, and use simulation to evaluate strategies. A strategy is not a static set of commitments, but rather a representation of high-level considerations that can be used to guide dynamic decision-making during simulated or actual execution. For planning experiments for a planetary rover, we implemented a simple strategy that identified key locations in order to guide the path the rover would take and that set parameters to control opportunistic decision-making during execution. More generally, the strategy can be thought of as setting priorities; for project management the analogous approach might include prioritizing tasks and resources. These priorities would guide decision-making during simulation, such as decisions about whether to reassign resources from one task to another in response to revised estimates.

Finding effective strategies can potentially be of great benefit to a manager by helping provide better risk assessment. Simulation of a static plan will not distinguish uncertain outcomes that can be easily accommodated by relatively small adjustments of the plan from uncertain outcomes that cannot. Simulation of a reasonable strategy for dynamically adapting plans can better reflect the way a manager would actually respond to change, and thus better uncover the critical issues in project planning. We don't, of course, claim that we will be able to create Artificial Intelligence algorithms that would be competitive with good managers, but if we can simulate *reasonable* responses to dynamic situations then we can increase the accuracy of the simulation.

We stress that this work is highly preliminary. This paper outlines our research agenda, and explains the reasons we are optimistic about being able to apply these ideas to the domain of project management.

2 SIMPLE EXAMPLES

This section describes a few simple examples that illustrate ways that uncertainty can make static decisions inefficient.

2.1 Resource Allocation

Suppose that in a software development project we have two tasks, *T1* and *T2*, both estimated to require six to twelve man-weeks of effort. Further suppose that they both are predecessors of another task, *T3*, and that both must be completed before *T3* can be started. If either *T1* or *T2* is on the critical path because of the relationship it has to *T3*, then they both must be on the critical path. Consequently, we might divide available resources evenly between them.

Of course, the estimates may have been wrong. Suppose that after three weeks have passed, it has become clear that *T1* is relatively easy, and its estimate is revised so that we now expect it to require an additional two to four more man-weeks. On the other hand, task *T2* has turned out to be difficult, and its revised estimate is now that it will take six to ten additional man-weeks.

If we currently have two developers assigned to each task, then based on the revised estimates we might want to consider moving one developer off of the first task and onto the second. That decision is not automatic. Increasing the number of developers from two to three isn't always (if ever) going to decrease the duration of that task proportionately. The "mythical man-month" problem is well-known, and there are also costs involved in the reassignment itself because of the time that will need to be spent bringing the reassigned developer up to speed on the new task, for example. Nevertheless, the resulting disruption may be worth the benefits of this adjustment of assignments and, if it is, a good manager will make that change. A simulation that assumes that the initial assignments will be maintained no matter how the uncertainties in the task durations play out is obviously going to be a flawed simulation.

2.2 Task Selection

Some software development projects have both a "core" set of objectives that must be delivered by a deadline no matter what, plus some additional objectives that are optional but desirable. For example, if a retail web site is developing web pages for a Christmas ad campaign, then having the core functionality for those pages delayed until after the Christmas shopping season is over would obviously make the project a failure. Time permitting, there may be any number of additional objectives that would be nice to have, but that are not absolutely necessary.

An initial schedule may appear to allow for all of the mandatory tasks, plus some number of optional tasks, but inaccurate estimates or unexpected events may force the manager to reassess the selected set of optional objectives. Underestimated requirements may require that optional tasks be sacrificed to ensure that the core functionality is completed on time. A simulation that does not reflect this contingency would be unrealistic, predicting project failure in scenarios for which it may be possible to modify the plan and ensure success.

Of course both of these examples are too simple to be interesting, but they illustrate very real effects of uncertainty that can be found embedded in large-scale real-world project planning.

3 BENEFITS OF SIMULATION

The motivation for wanting to incorporate simulation into project planning is clear. Simulation offers the possibility for representing the complexity that is necessary for realistic reasoning about a project, including the inherent uncertainty. For example, we can represent task duration as a probability distribution, given the current resource assignments, and use Monte Carlo simulation to get an estimate of the expected outcomes. We can represent uncertainty about resource availability, due to illness and other factors.

Employees are not interchangeable, and considering the individual characteristics of these resources can be very important to good project management. Employees will differ in their skill sets and levels of experience, of course. They can also differ significantly in the amount of time it takes them to "come up to speed" on a new task, differ on how well they can balance their time across multiple tasks, differ on whether they work most effectively individually or in small groups assigned to a task, and so on. Expected availability may differ from person to person. For example, the more experienced someone is, and the longer they have been with the company, the more likely it may be that they could be pulled away from a project to address a crisis elsewhere in the company.

Resources are even less interchangeable after work on a task has begun. Reassigning someone from one task to another, or adding an additional resource to a task after substantial progress has already been made on it, will usually be inefficient in the short term. It takes time to become familiar with the work done up to that point, and the people currently working on the task may need to divert some of their time to helping the new person become productive on that task.

Simulation offers the potential for representing the complexities of a software development project, the inherent uncertainties, and the individual characteristics of the agents involved and how they can be expected to affect the state of the project. However, as we have pointed out, simulation based on a static set of resource assignments, static estimates for task requirements, and static decisions about optional tasks is very unrealistic because as progress is made on tasks, uncertainties are reduced and revised estimates, better or worse, will often motivate revisions of the project plan. In the next section we discuss an approach that addresses these concerns.

4 STRATEGY-DRIVEN SIMULATION

Some AI planning algorithms build contingent plans that check conditions and branch accordingly (Ghallab, Nau, and Traverso 2004). For example, we could imagine a contingent plan that says that if the estimated completion of task *T2* slips beyond a certain date, we want to reassign a certain programmer from task *T1* to task *T2*. Where a relatively small number of highly-critical conditions can be identified, this can be an effective approach. As the number of branch points in a contingent plan grows, however, the size of the plan, and the time required to generate the plan, can grow exponentially. Furthermore, the more complex the simulation, the more difficult it will be to identify simple branch conditions.

Rather than searching the space of contingent plans, we propose to search the space of plan *strategies*. A strategy is defined relative to a decision-making procedure, such that the strategy guides that procedure in a reasonable direction without requiring extensive search. We can think of a strategy as identifying key factors from a global perspective. The *critical path* in a project, for example, captures something important about a project when we are analyzing static project plans. When the plan is dynamically adjusted in response to new information, the notion of a critical path is too simplistic.

At an abstract level the project management problem has some strong similarities to the Mars rover experiment planning problem (Joslin and Smith 2005)(Joslin, Frank, Jónsson, and Smith 2005). The Mars rover problem was modeled after current rovers and expectations of future rovers. More experiments are desirable than time, energy and other resources allow. Experiments are assigned utility values that reflect their relative importance, and the objective is to maximize the total utility of the experiments that we complete within a given time frame. Various constraints must be satisfied in order to achieve an experiment, possibly including location (i.e., an experiment may require that the rover be adjacent to a particular rock) and time (i.e., an image may be desired with sunlight from a certain angle, so the experiment may be required to be performed between certain hours of the day).

The duration of an action performed by the rover cannot be known precisely in advance. For example, the time that it takes to drive from point A to point B may depend on the terrain, soil consistency, navigation difficulty, and other factors. The only way to plan out all of the rover's actions in advance would be to use the worst-case estimates, resulting in an inefficient plan because the rover will need to sit idle whenever an action takes less than the worst-case time. We might construct a contingent plan, but as noted earlier this is practical only when the number of critical branch points is relatively small.

In (Joslin, Frank, Jónsson, and Smith 2005), rather than searching the space of possible plans or contingent plans, we searched a space of plan *strategies*. A strategy consisted of a set of weights and other information that reflected the "big picture" priorities that needed to be taken into account in order to achieve high-utility results. During execution a strategy would guide opportunistic decision-making. Computational resources on a rover are likely to be limited, but the computationally intensive effort goes into identifying an effective strategy. Applying that strategy is quick. At execution time, rather than trying to analyze the long-term implications of a decision, the strategy is assumed to reflect those long-term considerations, and a decision is made based on the priorities defined by the strategy.

After driving from point A to point B, for example, the rover would look at the current status – the time, energy available, etc. – and the experiments that could be performed next, and apply the strategy to decide whether to perform one of those experiments or to drive to a new location. If the drive was relatively quick then we may have the option of performing an experiment that would not have been feasible (due to a temporal window constraint on the experiment, or due to having less energy available after a longer drive) if the drive had taken longer. The strategy determines whether it seems more promising to perform that experiment, or to forego it because higher-utility experiments are likely to be available further along the rover's path, as determined using simulation.

The project management problem is similar in that we cannot know in advance exactly how long it will take to perform a task, and in a realistic setting therefore cannot know in advance what resource assignments we will want to have at some future point, what optional tasks we might want to pursue or abandon, etc. We propose to adapt the strategy-based approach. A strategy in this case would need to identify priorities and factors that should guide decisions about changing the project plan. For example, if the estimate for a task is revised upwards, as often happens, a good strategy should reflect the degree to which it is critical that resources be added to that task even if that means sacrificing other optional tasks. The simulation would then change (or not change) resource assignments accordingly.

A key difference between this domain and the rover domain is that the goal with rover experiment planning was increased autonomous operation. A strategy in that case would need to be effective at making reasonably good decisions across the range of likely outcomes for the operations performed by the rover. Certainly a good manager would never turn over project management decisions to software. Instead, the goal is to provide a decision-support tool. Consequently, a strategy for this domain only needs to reflect some reasonable approximation of the priorities that a manager might apply in dynamically adjusting a project plan. To the extent that it does so, the resulting simulation and analysis of risk factors, etc., can be much more realistic, and therefore more useful to the manager.

5 SEARCH FOR STRATEGIES

Strategies are evaluated by simulation. Given a set of resource assignments we want to use Monte Carlo simulation based on probability distributions defined for uncertain outcomes to generate a possible "next" world state. The granularity of that simulation would depend on the nature of the project. In our examples below we simulate one week of activity at each step, but that granularity could be larger or smaller depending on the characteristics of a particular project.

Part of the world state will be estimates about the future, relative to that world state. These estimates are what the strategy must use in order to decide whether and how to modify the current plan. Obviously it would be unfair for the strategy to be based on infallible estimates of the simulated future, so just as initial estimates are uncertain, when we simulate the revision of an estimate that revised estimate should also reflect realistic uncertainty.

We must execute multiple simulation trials to estimate the probability of various outcomes. We can estimate the expected utility using whatever definition of utility best reflects management objectives. We could be trying to minimize some cost calculation, minimize lateness if we have a deadline, or some combination of these and/or other factors. The "fitness" of a strategy is the expected utility estimated by this simulation.

We can apply a variety of AI optimization algorithms to search for strategies with high fitness values. For the rover planning experiments we used a simple genetic algorithm, and a similar approach should be effective here as well.

6 IMPLEMENTATION

Here we describe the current state of an on-going implementation. The purpose here is primarily to illustrate our approaches to resolving practical implementation issues, and to discuss intended directions for further implementation.

We defined a simple project consisting of both mandatory and optional tasks, based very loosely on a real-world example of a project involving the implementation of a Christmas ad campaign for a retail web site. In the particular problem instance discussed below, we have a team of eight developers, three mandatory tasks and eleven optional tasks, and a planning horizon of twelve weeks. The mandatory tasks must all be completed within that period in order for the project to be considered successful. Additionally, it is desirable to complete as many of the optional tasks as possible, so long as the mandatory tasks are not jeopardized.

6.1 Tasks

The "size" of a task is defined in *work units*. A work unit is equivalent to one man-week, for a typical worker familiar with the task. During startup phase a worker contributes to the task at a reduced rate. Currently we simplify this to a factor that reduces a developer's contribution during the first week in which they are assigned to a task. More realistically, this startup phase could be longer or shorter than one week, depending on how quickly the developer learns, how easily they adapt to new tasks, etc. Having too few or too many workers assigned to a task should reduce efficiency; for now we limit the number of workers that can be assigned to a task, with a maximum of either two or three depending on the task.

The startup penalty is assessed every time a worker is added to a task. If we were to remove a worker from a task (diverting them to something more urgent, until it is complete, for example), then add them back to the original task, the penalty would be assessed again. This is intended to reflect the fact that shifting assignments frequently is very inefficient, and a plan that requires frequent reassignments is unlikely to be an effective plan.

The startup period is not intended to reflect a learning curve period during which the worker learns a new skill. This should be represented by an explicit training period, after which the worker's new skill is reflected in the parameters for that worker. Ideally a simulation would allow for strategic decisions such as allowing time for training or sending someone to training classes in order to make them more valuable to the project once the training is complete. Hiring developers under temporary contracts would be another example of an option that may be available to a manager, and that may be important for risk mitigation. A strategy could reflect the tradeoffs of time and cost involved in deciding whether to augment a team with temporary contractors, so that this contingency could be considered in the simulation.

Currently we assume that developers are only assigned to one task at a time. We would want to allow the possibility of a resource being assigned to multiple tasks, but with some penalty factor applied to the worker's contribution level based on the number of tasks. This factor could be an individual characteristic of a developer, since some developers are much better at dividing their time effectively than others. It should probably also depend on the types of tasks involved.

6.2 Resources

The only resources currently represented are the software developers, and these are the agents in the simulation. The individual characteristics of those agents represented in the current simulation are the probability distribution of their contribution to the assigned task in any given week, and a factor representing how readily they can be expected to become fully productive on a newly-assigned task.

The contribution is represented as a uniform distribution over a specified range. A value of 1.0 represents the weekly contribution of a "typical" developer, once they have become familiar with the task. The minimum and maximum values were selected randomly during problem generation, resulting in a variety of characteristics. For example, one developer might contribute at a relatively low rate, but contribute at that rate very dependably. Another might contribute at a much higher rate some weeks, but with a high degree of variability. This sort of variation might be due to temperament and/or to other recurring responsibilities that require the employee's attention.

The startup factor represents the fact that we cannot expect to assign a developer to a new task and have them contributing at full capacity from the first day. On the other hand, some people learn more quickly than others. As described above, each resource has a startup penalty, applied to the first week after assignment to a task. If the penalty factor is 0.5, for example, then during that first week they are assumed to contribute at half of their normal rate.

A much richer representation of the individual characteristics of resources is desirable. Skill sets, for example, should be matched to task types, and the expected contribution of a developer is unlikely to be the same across all types of tasks.

6.3 Simulation and Uncertainty

The granularity of simulation is one week. At the beginning of each week we assign resources to tasks. We then simulate the contribution of each resource to the assigned task (if any), and then make new staffing decisions at the beginning of the next simulated week. Staffing decisions can depend on the current best estimate of required time to complete the task with some hypothetical staffing.

We have an initial estimate of the work units for a task. The estimate is a range, not a single value. When a simulation begins, we decide what the actual number of work units will be for that simulation run, and define the estimate range around that. The actual (simulated) value will fall within that estimate range – i.e., we assume that the estimate is valid – but the size of that estimate range, and where the actual value falls within that range, are random.

During the simulation we don't want to use the original estimate after a task has been partially completed. As progress is made on a task, hopefully we get a better idea of how hard that task is turning out to be. We don't want to use the actual value selected for the simulation, because that would assume perfect knowledge about the future. We also don't want to have the estimate converge gradually upon the simulated value, because that would be unrealistic and might risk "leaking" information about the future, i.e., the rate of change in the estimate could be extrapolated to make an accurate prediction about the future.

What we want is to simulate the revision of estimates. Task estimates tend not to change gradually, and more often than not they change at a point fairly close to the original estimate, i.e., it becomes obvious that the time remaining is not sufficient, and at that point the estimate is revised. The revised estimate is itself unlikely to be exactly correct. To model this we randomly select a point prior to the original estimate, with a preference for points closer to the original estimate. When that much work has been accomplished on a task, we revise the estimate based on the remaining actual (simulated) number of work units, again defining a randomized estimate range around that value. We again define a trigger point at which the new estimate will be revised. As we get closer to completion of a task, the error in the estimates decreases, which is as it should be.

6.4 Strategy

Resource allocation decisions are based on a rough approximation of the probability of successful completion of a task, given the current resource assignments and the time remaining before the deadline. To calculate this estimated probability we use the mean expected contribution of each assigned resource (ignoring the startup factor for now), and look at the range of estimates for the required work-units. Assuming a uniform distribution across that range, we can estimate a probability of completing the task prior to the deadline.

We have defined a very simple strategy consisting of three threshold values. One is the required probability of completion for mandatory tasks. We assign resources to a mandatory task until its estimated probability of completion exceeds the threshold, or we reach the limit on the number of resources that can be assigned to that task, or no more resources are available. If necessary, resources can be preempted from optional tasks. In effect, the higher the value for this threshold, the less initial risk we are willing to take, based on the current estimates. The less initial risk we are willing to take, the fewer resources we will be able to assign to optional tasks at that point.

Once resources have been assigned to mandatory tasks we look at the optional tasks, with a preference for tasks with a higher "utility rate," defined as the ratio of the task s's utility to the mean of the work requirement estimate range. Two threshold parameters guide resource assignment decisions for optional tasks. First, we attempt to assign resources to a task until it's probability of completion estimate exceeds one threshold. If that turns out not to be possible we may try to complete that task anyway. The second threshold related to optional tasks sets a minimum acceptable estimate of the probability of completion. If we cannot assign enough resources to exceed that threshold, or if revised estimates cause us to fall below that threshold and we cannot add resources to raise the probability above it again, then we abandon that optional task and make those resources available for other tasks. Resources are only preempted from optional tasks to satisfy mandatory tasks, never to improve another optional task.

This algorithm for resource assignment is executed at the beginning of each simulation period. This algorithm is a simple starting point that can obviously be improved. For example, it might make sense to abandon an optional task and preempt those resources for another optional task if both have relatively low estimates of their probability of success, i.e., we may have a preference for maximizing the probability of one successful optional task rather than continuing both and risking having both fail. We also do not currently allow preemption from a mandatory task to another mandatory task, but clearly there are times that this would make sense. The proximity of the deadline is also a factor that should be considered in these decisions.

More importantly, this strategy implementation is too simple. We have in mind a strategy that does a better job of identifying task priority than the current probability estimate. In similar algorithms, priorities have been assigned to individual tasks (Joslin and Clements 1998), or weights have been defined that allow priorities to be calculated dynamically (Joslin, Frank, Jónsson, and Smith 2005), and these approaches could also be combined. We also want to consider resource prioritization. Currently we use the average contribution of a resource to decide what resource to select, but better prioritizations would be possible.

7 EXPERIMENTAL RESULTS

Our simple simulation can apply a strategy (the three threshold values) over the weeks within the planning horizon. We run multiple simulation trials with randomized values for the actual workload requirements, the workload estimates, the estimate revisions and the points at which those revisions occur, and the weekly contribution of each resource to its assigned task. Resource preemption occurs when the strategy indicates that it should. The output of each simulation run tells us whether all of the mandatory tasks were achieved, the utility achieved for optional tasks completed, and the number of resource preemptions (if any) that occurred. Multiple simulation trials give us an estimate of the probability of project success (all mandatory tasks completed), and the expected utility when successful.

We ran simulations varying the three parameters systematically across the range of possible values, with and without preemption from optional tasks to mandatory tasks. For each set of parameter values, 100 simulation trials were run.

With resource preemption allowed, most parameter settings (86%) avoided project failure across all simulation runs, and the worst parameter settings had an estimated risk of project failure of 2% or less. Across all settings and all simulation runs, 90% of the time the project completed all three of the mandatory tasks and two additional optional tasks. Ten percent of the time only one optional task was completed, and small fraction of a percent of the time three optional tasks were completed.

In contrast, without allowing resource preemption, 98% of the parameter settings resulted in a prediction of the chance of project failure of 5% or worse, and 81% resulted in a chance of project failure of 10% or worse. Three percent of the settings resulted in failure rates of 20% or more.

When resource preemption was allowed, it was not necessary to switch a resource from one task to another very often in order to achieve the higher probability of success. The average number of resource preemptions over the life of the project ranged from 0.1 to 0.9, across the various parameter settings, with a median of 0.7. In other words, even in the worst case we never averaged more than one resource preemption over the life of the project. As noted earlier, the simulation was designed to recognize that reassigning a developer from one task to another does not mean that they can immediately start contributing to the new task at full capacity.

An algorithm such as a genetic algorithm could be used to search this space of parameters (or a more realistic parameter space that includes some notion of prioritizing tasks and/or resources). For example, we might search for parameter settings that maximize the chance of success on mandatory tasks, maximize the expected utility of optional tasks when all mandatory tasks are achieved, and minimize the expected frequency of resource preemption.

The current problem is too simplified to be of any practical interest, so further analysis of the results serves little purpose. The code would also need to be improved substantially before we could meaningfully claim that it reflects "intelligent" decision-making about resource preemption, accurate estimates of the probability of task completion, etc. We show these results of our initial investigation only as a way of illustrating some ideas about the direction we intend to take with this research.

8 RELATED WORK

The idea of using simulation methods in project management has been around for many years. Van Slyke (Van Slyke 1963) is widely cited as the first to use simulation for project network analysis. That and other early work looked at the use of simulation with PERT and related techniques (Hebert 1979). Uncertainty about task duration was modeled, and resource assignments were made using heuristics for "resource leveling" when the tasks eligible for execution required more resources than were available.

More recent work has extended that approach in a variety of ways. A fairly recent survey is found in (Pich, Loch, and De Meyer 2002). In looking at AI techniques applied to project management, the authors point out a distinction between "conditional planning," in which contingencies are built into the plan, and "execution monitoring," in which a static plan is executed, but if the plan fails at any point then the current plan is discarded and replanning occurs. We would describe our algorithm as not quite fitting either category. We do not use replanning in the simulation, but instead the strategy is applied based on the simulated conditions that hold at each point. On the other hand, a strategy is not a conditional or contingent plan in the usual sense, because it does not itself spell out any decisions. A strategy only implies those decisions in conjunction with the execution algorithm.

Another recent example is (Antoniol, Di Penta, and Harman 2004), in which simulation is used not only to represent the inaccuracy of task estimates, but also the possibility that a task may need to be abandoned, or that a task may need to be revisited at some point after it has been considered complete. Their research has some similarities to the work we have described in this paper. They use a genetic algorithm (GA) to search for what we would describe as an effective prioritization of tasks. They don't, however, appear to model any individual characteristics of the agents in the simulation, nor do task abandonment and rework represent management decisions in the simulation based on revised estimates.

The best-known commercial project management product is Microsoft Project (Microsoft Corporation 2003). Although that product does not directly allow for representing uncertainty about tasks, or for using simulation, a number of add-on products provide these capabilities. One example is SimEstimator (Orlando Software Group, Inc. 2004), a tool that allows project plans to be simulated, with uncertainty on task duration and other variables. The simulation appears to assume a static plan, however.

We are not aware of any commercial product that simulates the kind of dynamic decision-making we are interested in modeling. In general, the way that the most commonly used project planning and management software is used involves defining a list of tasks and the predecessor/successor relationships between them, assigning task durations (weeks, or man-weeks, for example), and assigning resources to tasks. Analysis tools may then provide feedback on that plan, with or without considering the various elements of uncertainty that are unavoidable in real projects, but the state of the art appears to be analysis and/or simulation that assumes a static project plan.

9 FUTURE DIRECTIONS

In this paper we have outlined a direction for future research, adapting some algorithms that show promise for using simulation techniques in planning under uncertainty and applying those algorithms to software project planning. The current implementation is not sufficiently accurate to be of practical interest. To be of practical interest, not only will the accuracy and detail of the simulation need to be increased, the algorithm would also have to be embedded in a decision-support tool. Unlike some other domains, such as the Mars rover experiment planning domain, the point of using simulation to support software project management is not autonomous operation. A good manager will not want to just receive a plan that defines a set of resource assignments, for example. Instead, they will want a tool that provides them with insight into what factors they should consider. Good parameter settings, particularly when the current simple approach is augmented to reflect prioritization of tasks and resources, need to be translated into qualitative guidelines about what aspects of a project seem to be the most critical.

Simulation can be particularly useful in this respect if it allows a manager to focus on critical scenarios. If search is used to find an effective set of prioritizations, a manager might then want narrow the focus to the simulation runs with those prioritizations in which the results were least successful. Aggregate data about the tasks, task/resource assignments, etc., that caused those particular simulation trials to turn out badly may be of interest. Looking at the events in individual simulations may be interesting to a manager as well. Seeing the sequence of events that let to a poor outcome may help a manager anticipate critical conditions earlier, or may suggest ways in which the simulation model needs to be refined.

Our intention is to continue developing this approach by implementing a more detailed and realistic simulation, to experiment with a richer representation of a "strategy," and to apply this technique to more realistic data. We have also identified a possible source of detailed historical project management data, and hope to use that to help validate our approach. We look forward to furthering these initial steps toward using simulation in decision-making support for software project management.

ACKNOWLEDGMENTS

The authors would like to thank Jeff Gilles for helpful discussions of this material.

REFERENCES

Antoniol, G., M. Di Penta, and M. Harman. 2004. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In *Proc. of the 10th International Software Metrics Symposium*, 172–183.

Ghallab, M., D. Nau, and P. Traverso. 2004. *Automated planning: Theory and practice*. Morgan Kaufmann.

Hebert, J. E. 1979. Applications of simulation in project management. In *Proc. of the 1979 Winter Simulation Conference*, ed. M. Spiegel, R. Shannon, and H. Highland, 211–219.

Joslin, D., J. Frank, A. Jónsson, and D. Smith. 2005. Simulation-based planning for planetary rover experiments. In *Proc. of the 2005 Winter Simulation Conference*, ed. M. Kuhl, N. Steiger, F. Armstrong, and J. Joines.

Joslin, D., and D. E. Smith. 2005. Squeaky-Wheel Optimization for planetary rover experiment planning. In *Proc. of the Intelligent Systems and Agents 2005 Conference (ISA2005)*.

Joslin, D. E., and D. P. Clements. 1998. Squeaky Wheel Optimization. In *Proc. of the 15th National Conference on Artificial Intelligence (AAAI-98), Madison, WI*, 340–346.

Microsoft Corporation 2003. Microsoft project 2003. `<http://office.microsoft.com/en-us/FX010857951033.aspx>`.

Orlando Software Group, Inc. 2004. Simestimator project management and risk analysis tool. `<http://www.osgi.com/simestimator.html>`.

Pich, M. T., C. H. Loch, and A. De Meyer. 2002. On uncertainty, ambiguity, and complexity in project management. *Management Science* 48 (8): 1008–1023.

Van Slyke, R. M. 1963. Monte carlo methods and the pert problem. *Operations Research* 11 (5): 839–860.

AUTHOR BIOGRAPHIES

DAVID JOSLIN is an assistant professor in the Computer Science and Software Engineering department at Seattle University. He received his PhD from the University of Pittsburgh in 1996. His research interests include AI planning and scheduling, and game AI algorithms. His e-mail address is `<joslind@seattleu.edu>`.

WILLIAM POOLE is a professor and the chair of the Computer Science and Software Engineering department at Seattle University. He received his PhD from the University of California, Berkeley. His research interests include software project management and process and the effects of global information technology sourcing. His e-mail address is `<bpoole@seattleu.edu>`.