SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2007; **12**: 461–473
Published online 30 May 2007 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/spip.332

Assessing Hybrid Incremental Processes for SISOS Development

Research Section

Raymond Madachy*,†, Barry Boehm and Jo Ann Lane
University of Southern California Center for Systems and Software Engineering, 941 W. 37th Place, Los Angeles, CA, USA

New processes are being assessed to address modern challenges for Software-Intensive Systems of Systems (SISOS), such as coping with rapid change while simultaneously assuring high dependability. A hybrid agile and plan-driven process based on the spiral lifecycle has been outlined to address these conflicting challenges with the need to rapidly field incremental capabilities in a value-based framework. A system dynamics model has been developed to assess the incremental hybrid process and support project decision making. It estimates cost and schedule for multiple increments of a hybrid process that uses three specialized teams, and also considers the mission value of software capabilities. It considers changes due to external volatility and feedback from user-driven change requests, and dynamically reestimates and allocates resources in response to the volatility. Deferral policies and team sizes can be experimented with, and it includes trade off functions between cost and the timing of changes within and across increments, length of deferral delays, and others. We illustrate how the model can be used to determine optimal agile team size to handle changes. Both the hybrid process and simulation model are being evolved on a very large scale incremental SISOS project and other potential pilots. Copyright © 2007 John Wiley & Sons, Ltd.

KEY WORDS: software process modeling; system dynamics; software-intensive systems of systems; hybrid processes; agile processes; plan-driven processes; incremental development; spiral model

1. INTRODUCTION AND BACKGROUND

Our experiences in helping to define, acquire, develop, and assess 21st century Software-Intensive Systems of Systems (SISOS) have taught us that traditional acquisition and development processes do not work well on such systems (Boehm *et al*. 2004, Boehm 2005). At the University of Southern California (USC) Center for Systems and Software Engineering (CSSE) we are using simulation modeling to help formulate and assess new processes to meet the challenges of these systems.

The systems face ever-increasing demands to provide safe, secure, and reliable systems; to provide competitive discriminators in the marketplace; to support the coordination of multicultural global enterprises; to enable rapid adaptation to change; and to help people cope with complex masses of data and information. These demands will cause major differences in the current processes (Boehm 2005).

We and others have been developing, applying, and evolving new processes to address SISOS challenges. These include extensions to the risk-driven

* Correspondence to: Raymond Madachy, University of Southern California Center for Systems and Software Engineering, 941 W. 37th Place, Los Angeles, CA, USA
† E-mail: madachy@usc.edu

spiral model to cover broad (many systems), deep (many supplier levels), and long (many increments) acquisitions needing rapid fielding, high assurance, adaptability to high change traffic, and complex interactions with evolving Commercial Off-the-Shelf (COTS) products, legacy systems, and external systems.

The distinguishing features of a System of Systems (SOS) are not only that it integrates multiple independently-developed systems, but also it is very large, dynamically evolving, and unprecedented, with emergent requirements and behaviors and complex socio-technical issues to address. Thus we have developed a system dynamics model because the methodology is well-suited to modeling these dynamic phenomena and their interactions.

1.1. Previous Simulation Work

System dynamics has been applied to software processes starting with the seminal model in (Abdel-Hamid and Madnick 1991) of software project dynamics for a single waterfall cycle. Since then, there have been over 90 publications using system dynamics for software processes (Madachy 2007). None have specifically focused on SISOS issues, nor concurrently addressed hybrid processes, incremental (or iterative) development and the effects of requirements changes.

Software requirements volatility and change control dynamics were investigated by Ferreira (2002) and Ferreira *et al*. (2003). Data in that research showed an average of 32% requirements volatility in over 200 projects, which was captured in their model. Iteration management, as a way to reduce cycle time in the midst of change, was addressed by Ford and Sterman (2003), where they modeled the 90% syndrome and impact of concealing requirements changes. The use of fire fighting techniques to handle late changes is described in (Repenning 2001), which illustrates how the late rework leads to perpetual fire fighting in a multi-project development environment. Both (Ford and Sterman 2003) and (Repenning 2001) were for high technology applications, but not in the specific context of software projects.

A few efforts have simulated incremental or iterative software development. The research by Tvedt (1996) modeled concurrent incremental development and the impact of inspections. The cost and schedule impacts of different increment options were assessed with the model. Another incremental development model to assess cost and schedule was developed by Sycamore (1996), though both of these models were limited in their cycle variability since they used replicated structures to represent the different increments. Iterative development was modeled with system dynamics by Powell *et al*. (1999) and Powell (2001) to assess both concurrent software engineering and staged-delivery as methods to improve cycle time. Requirement changes were included in the iterative model by Powell (2001) and based on substantial empirical data. However, none of these efforts considered personnel makeup to best handle changes.

A qualitative model of agile processes is described by Fernández-Ramil *et al*. (2005), but the modeling of agile processes with system dynamics is an area of current and future work, as described in (Madachy 2007). This paper presents the first published results of a system dynamics model applied to either hybrid or agile processes.

The model, presented here, builds on the concepts of requirements volatility and incremental or iterative development in previous research. But it goes a step further by showing the effects of a hybrid process used in incremental development to move away from fire fighting to a more controlled process for accommodating changes quickly while balancing software cost, schedule and value issues.

1.2. The Scalable Spiral Model

The outlines of a hybrid plan-driven/agile process for developing SISOS product architecture are emerging. It is a risk-driven balance of agility and discipline (Boehm and Turner 2004). In order to keep SISOS developments from becoming destabilized from large amounts of change traffic, it is important to organize development into plan-driven increments, in which the suppliers develop to interface specs that are kept stable by deferring changes, so that the systems can plug and play at the end of the increment. But for the next increment to hit the ground running, an extremely agile team needs to be concurrently doing continuous market, competition, and technology watch, change impact analysis, COTS refresh, and renegotiation of the next increment's prioritized content and the interfaces between the suppliers' next increment interface specs.

Softw. Process Improve. Pract., 2007; **12**: 461–473
DOI: 10.1002/spip

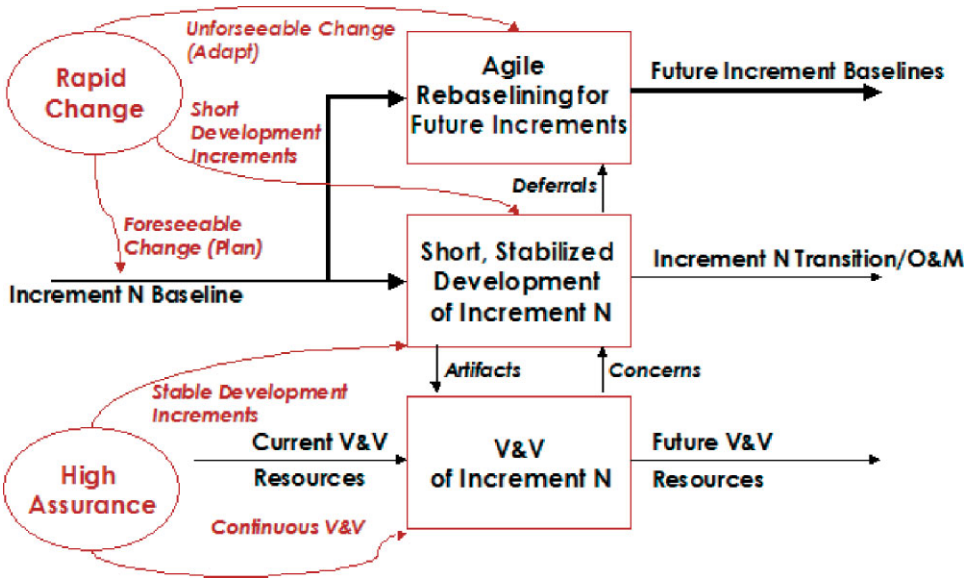

Figure 1. The scalable spiral process model: increment activities

The spiral model was introduced in 1986 and later elaborated for WinWin extensions (Boehm *et al*. 1998). It has continued to evolve to meet the needs of evolving development processes. We have been converging on a scalable spiral process model for SISOS that, for partial implementations to date, has scaled well from small e-services applications to super-large defense systems of systems, and multi-enterprise supply chain management systems.

Figure 1 shows a single increment of the development and evolution portion of the model. It assumes that the organization has developed:

- A best-effort definition of the system's steady state capability;
- An incremental sequence of prioritized capabilities culminating in the steady state capability;
- A Feasibility Rationale providing sufficient evidence that the system architecture will support the incremental capabilities, that each increment can be developed within its available budget and schedule, and that the series of increments create a satisfactory return on investment for the organization and mutually satisfactory outcomes for the success-critical stakeholders.

As seen in Figure 1, the model is organized to simultaneously address the conflicting challenges of rapid change and high assurance of dependability. It also addresses the need for rapid fielding of incremental capabilities with a minimum of rework, and the other trends involving integration of systems and software engineering, COTS components, legacy systems, globalization, and user value considerations.

The hybrid process uses a three-team cycle (lean, plan-driven, stabilized developers; thorough V&Vers; and agile, pro-active rebaseliners) that plays out from one increment to the next.

The need to deliver high assurance incremental capabilities on short fixed schedules means that each increment needs to be kept as stable as possible. This is particularly the case for very large systems of systems with deep supplier hierarchies, in which a high level of rebaselining traffic can easily lead to chaos. The risks of destabilizing the development process make this portion of the project into a waterfall-like build-to-specification subset of the spiral model activities. The need for high assurance of each increment also makes it cost effective to invest in a team of appropriately skilled personnel to continuously verify and validate the increment as it is being developed.

However, 'deferring the change traffic' does not imply deferring its change impact analysis, change negotiation, and rebaselining until the beginning of the next increment. With a single development team and rapid rates of change, this would require a team optimized to develop to stable plans and specifications to spend much of the next increment's

DOI: 10.1002/spip

scarce calendar time performing tasks much better suited to agile teams.

The appropriate metaphor for addressing rapid change is not a build-to-specification metaphor or a purchasing-agent metaphor but an adaptive 'command-control-intelligence-surveillance-reconnaissance' (C2ISR) metaphor. It involves an agile team performing the first three activities of the C2ISR 'Observe, Orient, Decide, Act' (OODA) loop for the next increments, while the plan-driven development team is performing the 'Act' activity for the current increment. These agile activities are summarized below:

- Observing involves monitoring changes in relevant technology and COTS products, in the competitive marketplace, in external interoperating systems and in the environment; and monitoring progress on the current increment to identify slowdowns and likely scope deferrals.
- Orienting involves performing change impact analyses, risk analyses, and trade off analyses to assess candidate rebaselining options for the upcoming increments.
- Deciding involves stakeholder renegotiation of the content of upcoming increments, architecture rebaselining, and the degree of COTS upgrading to be done to prepare for the next increment. It also involves updating the future increments' Feasibility Rationales to ensure that their renegotiated scopes and solutions can be achieved within their budgets and schedules.

A successful rebaseline means that the plan-driven development team can hit the ground running at the beginning of the 'Act' phase of developing the next increment, and the agile team can hit the ground running on rebaselining definitions of the increments beyond.

As much as possible, usage feedback from the previous increment is not allowed to destabilize the current increment, but is fed into the definition of the following increment. Of course, some level of mission-critical updates will need to be fed into the current increment, but only when the risk of not doing so is greater than the risk of destabilizing the current increment.

1.3. System Dynamics Introduction

System dynamics is a simulation methodology for modeling continuous systems. Quantities are expressed as levels, rates and information links representing feedback loops. It provides a rich and integrative framework for capturing myriad process phenomena and their relationships. System dynamics is well-suited to deal with the complexities of SOS because it captures dynamic feedback loops and interacting phenomena that cause real-world complexity (Madachy 2007).

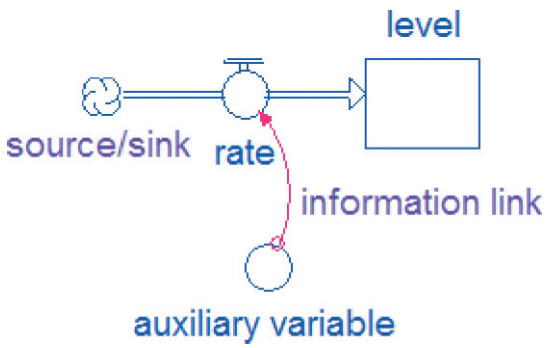

Figure 2. System dynamics model notation

Figure 2 serves as a model diagram legend showing the notation for system dynamics elements in a simple system. These notations and following brief descriptions of the elements may help understand the model described in Section 2.

Levels are the state variables representing system accumulations over time, and are also called stocks. They can serve as a storage device for material, energy, or information. Contents move through levels via inflow and outflow rates. Levels are a function of past accumulation of rates. Levels in this model represent the team sizes and software capabilities in different stages.

Sources and sinks represent the levels or accumulations outside the boundary of the modeled system. Sources are infinite supplies of entities and sinks are repositories for entities leaving the model boundary.

Rates are also called flows; the 'actions' in a system. They effect the changes in the levels. Rates may represent the decisions or policy statements. Rates are computed as a function of levels, constants and auxiliaries. This model has rates for capability changes, work production and staffing changes.

Auxiliaries are algebraic converters of input to output, and help elaborate the details of stock and flow structures. An auxiliary variable must lie in an information link that connects a level to a rate. Auxiliaries often represent 'score-keeping'

Softw. Process Improve. Pract., 2007; **12**: 461–473
DOI: 10.1002/spip

variables such as the parameters used in this model to calculate the required staffing levels.

Information links are used to represent the information flow as opposed to the material flow. Rates, as control mechanisms, often require links from other variables (usually levels or auxiliaries) for decision making. Information links can represent closed-path feedback loops between elements.

2. MODEL OVERVIEW

The primary portion of the system dynamics model diagram showing increment activities and the teams is given in Figure 3. It is built around a cyclic flow chain for capabilities and uses arrays to model multiple increments. The flow chains for the increment activities show multiple layers of

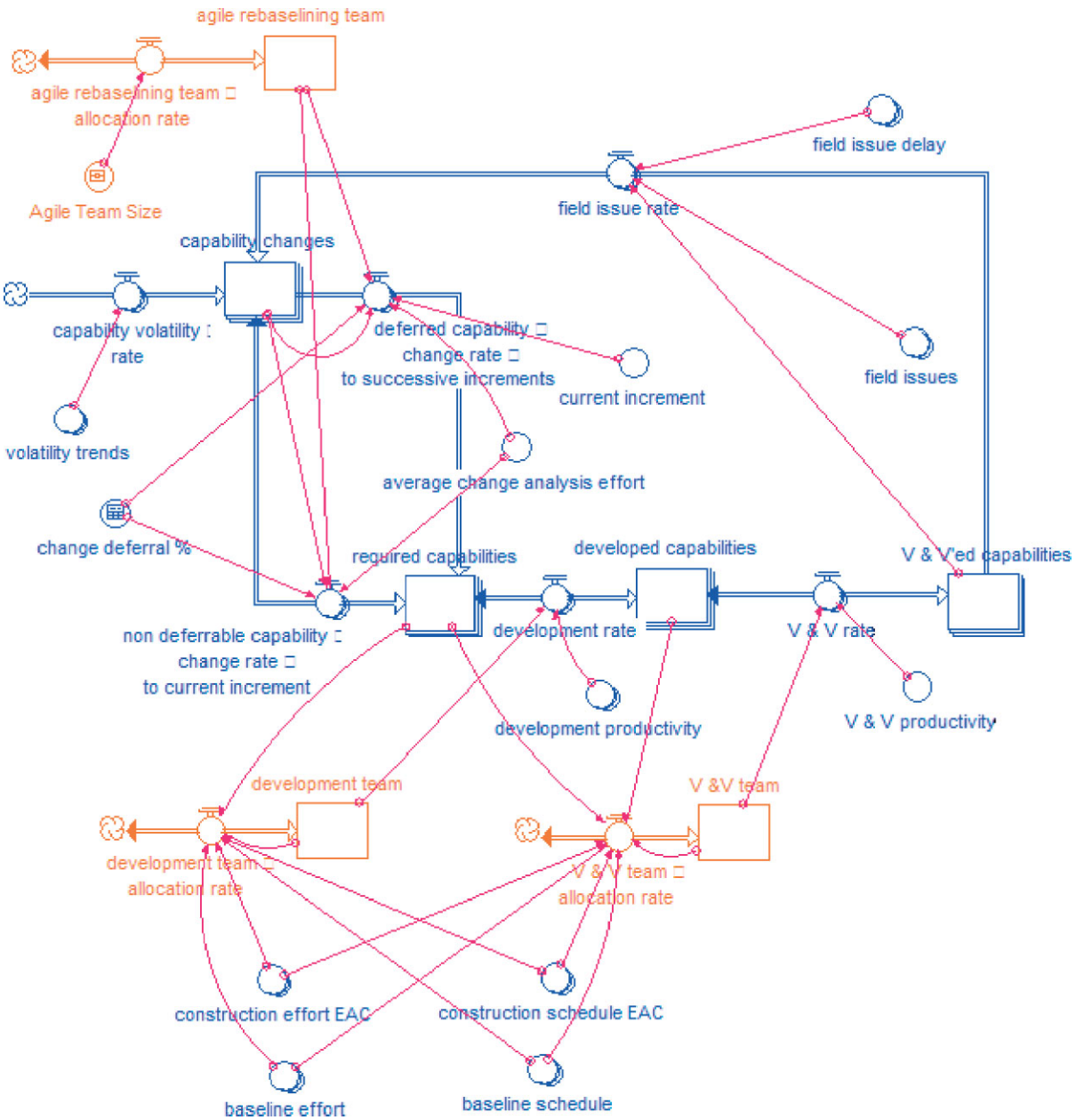

Figure 3. Model diagram

DOI: 10.1002/spip

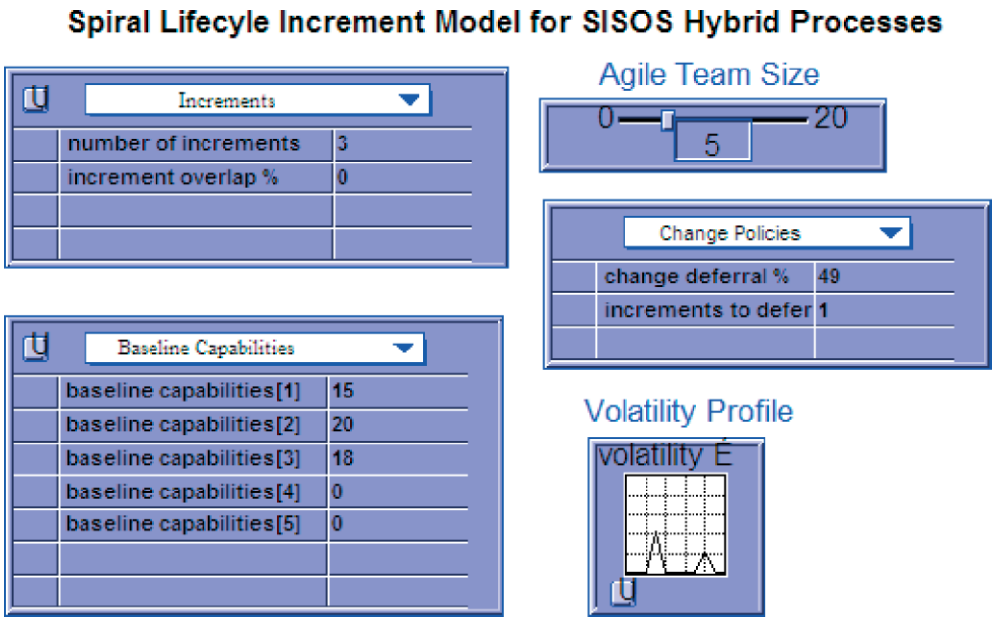

Figure 4. Simulation inputs

levels and rates; these identify array elements that correspond to the increments. Thus, the flow chain and its equations are arrays of five to model five increments (this preset number can be changed to model more or less increments). Note that multiple flows that appear to be flowing into the same level may be flowing into different levels with different array indices, depending on the logic of the increment allocation of capabilities.

Unanticipated changes arrive as aperiodic pulses via the *volatility trends* parameter. This is how they actually come on the projects *versus* a constant level of volatility over time. The user can specify the pulses graphically (see the input for Volatility Profile in Figure 4) or use formulas. The *capability volatility rate* will flow the changes into the corresponding increment for the current time.

From there, they arrive in the level for *capability changes* and are then processed by the agile rebaselining team. They analyze the changes per the *average change analysis effort* parameter. Their overall productivity is a function of the *agile team size* (as specified by the user in Figure 4) and the average analysis effort.

The *change deferral percentage* is a policy parameter to specify the percentage of changes that must be deferred to later increments via *deferred capability change rate to succeeding increments* to *required capabilities* for the appropriate increments. The remaining ones are nondeferrable that flow into the *required capabilities* for the current increment via the rate *nondeferrable capability rate change to current increment*. The deferral policy parameter is also shown in the inputs in Figure 4.

The two arrayed flows between the *capability changes* and *required capabilities* are complicated multiplexers that cannot be fully visualized on the diagram. In both the paths, there are five flows (or pipes) between them that capabilities may go through and the capabilities can move between the different increment flows. For different reasons, capabilities may be assigned, deferred, or delayed to any of the five increments, and the full set of possible paths cannot be seen.

When an increment starts, the *required capabilities* are developed by the development team at the *development rate* and flow into *developed capabilities* (all using the flow chain array index corresponding to the proper increment).

Similarly, the *developed capabilities* are then picked up by the V&V team for their independent verification and validation. They do their assessment at the *V & V productivity rate* and the capabilities flow into *V & V'ed capabilities*.

The rates in the flow chain between *capability changes*, *required capabilities*, *developed capabilities* and

DOI: 10.1002/spip

V & V'ed capabilities are all bidirectional. This is a provision for capabilities to be 'kicked back' or rejected by the various teams and sent back up the chain. For example, there are times when the developers have major concerns about a new capability and send it back to the rebaselining team. Likewise the V&V team might find some serious defects to be reworked by the developers.

Finally, there are user-driven changes based on field experience with the system. These are identified as *field issues* that flow back into the *capability changes* per the *field issue rate* at a constant *field issue delay* time. The *field issues* parameter represents the amount of concern with the fielded system and accounts for a primary feedback loop.

The agile baselining team is shown in the top left of the diagram. The size of the team can be specified as a constant size or a varying number of people over time via the inputs in Figure 4. The *agile rebaselining team allocation rate* flows people in or out of the team to match the specified team size over time.

The development and V&V teams are shown at the bottom. Their allocation rates are based on the construction effort and schedule for the required capabilities known to date. Currently the productivities and team sizes for the development and V&V are calculated with a Dynamic COCOMO (Boehm *et al*. 2000) variant. They are equivalent to COCOMO for a static project (the converse situation of this model context) and continuously recalculated for changes. However, this aspect of the model whereby the team sizes are parametrically determined from size and effort multipliers will be refined so that the constraints can be put on the development and V&V staff sizes. See Section 2.2 for more details on the staffing algorithms.

2.1. Trade off Functions

There are several functional relationships in the model that effect trade offs between deferral times and cost/schedule. For one, it is costlier to develop software when there is a lot of volatility during the development. If required, capabilities are added to an increment being developed; the overall effort increases due to the extra scope as well as the added volatility. The effort multiplier in Figure 5 is used to calculate the construction effort and schedule based on a volatility ratio of total required capabilities to the baseline capabilities. This example shows a simple linear relationship, though the graphical

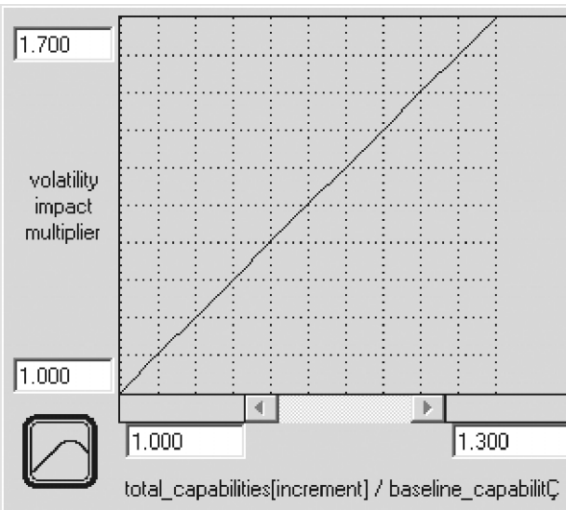

Figure 5. Volatility effort multiplier

construct allows it to be user-defined and become nonlinear (as in Figure 6 described next).

The volatility factor in Figure 5 is an aggregate multiplier for volatility from different sources. It works similarly to the platform volatility multiplier in COCOMO II (Boehm *et al*. 2000), the USC-CSSE software cost estimation model. The difference in this context is that there may be many more sources of volatility (e.g. COTS, mission, etc.). This multiplier effect only holds for an increment when changes arrive midstream. If new changes are already in the required capabilities, when an increment starts, then it has no effect.

Additionally, the later a new capability comes in during construction, the higher the cost to develop it. This is very similar to the cost-to-fix defects whereby the costs increases exponentially. Figure 6 shows the life cycle timing multiplier based on a ratio of the current time to the entire increment schedule.

Under normal circumstances, there is an additional cost of delaying the capabilities to future increments because there is more of a software base to be dealt with and integrated into. Therefore, we increase the cost of deferring to future increments by an additional 25% relative to the previous increment (this parameter is easily changed).

2.2. Dynamic Resource Allocation

In response to changes in the capabilities, the model calculates the personnel levels needed for the new

DOI: 10.1002/spip

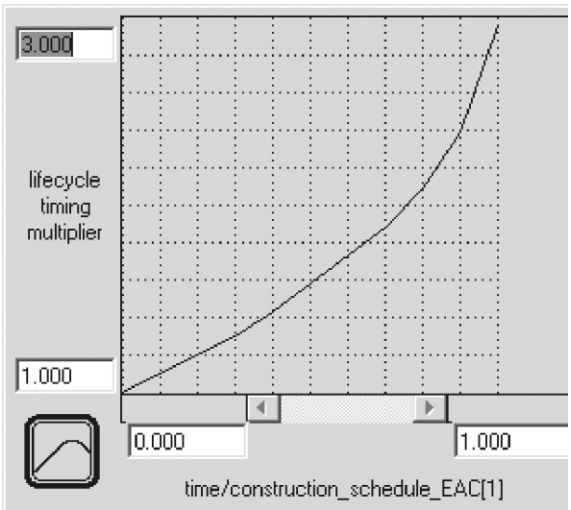

Figure 6. Life cycle timing effort multiplier

increment size and interpolates for the amount of work done. Part of the structure for continuous calculation of staffing levels is shown in Figure 7 where EAC stands for 'Estimate-At-Completion'. Baseline effort and schedule refer to the original plan without any volatility. The parameters for construction effort and construction schedule are the ones used to determine the staffing allocation rates shown at the bottom of Figure 3.

An example of the delta staffing for the development team when new capability changes come into the system is:

development team allocation rate

= (development_effort_fraction

× construction_effort_EAC(increment)/

× development_schedule_fraction

× construction_schedule_EAC(increment))

− development_effort_fraction

× baseline_effort(increment)/

× (development_schedule_fraction

× baseline_schedule(increment).

The development effort and schedule fractions are the calibrated portions of effort and schedule for development with respect to all of the construction, and the V&V allocation algorithm works similarly.

If the increment has just started, then the interpolated staffing level will be closer to the higher level needed for the new EAC. If the increment is mostly done, then it doesn't make sense to increase staff to the EAC level because almost all the work is done anyway.

On the SISOS project we are primarily applying the model to, there are already over 2000 software development people on-board, and most contractors have already hired their steady state levels of personnel. Thus it is reasonable to assume a nearly constant level of construction staffing with no major ramp-ups, and a step function staffing profile that approximates labor with average levels and nonoverlapping phases.

The step function staffing profile is improved with a Rayleigh curve staffing version of the model that intrinsically changes the staffing when changes occur with no interpolation necessity. It also assumes overlap of activities, and the starting

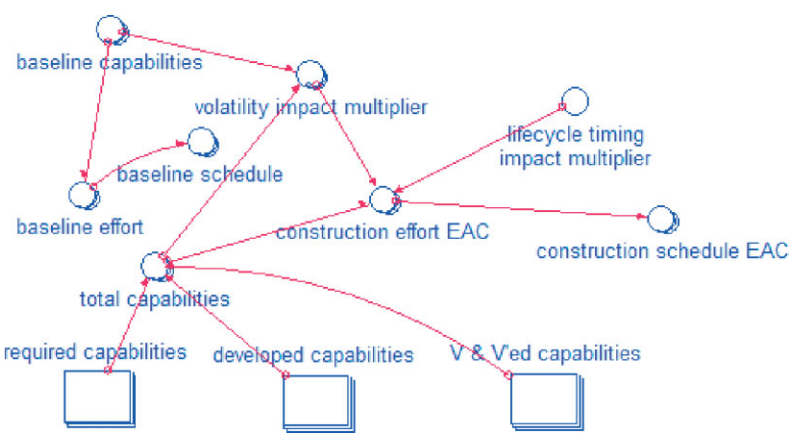

Figure 7. Staffing calculation parameters

DOI: 10.1002/spip

point of V&V can be specified with respect to development. For illustration, in this article, we will continue to use the step function version since the results are generally easier to follow and interpret.

2.3. Parameterizations

Since this is a macro model for very large systems, a capability is a 'sky level' requirement measure. It is defined as a very high level requirement and we have made that equivalent to 10 KSLOC for the purpose of estimation. The construction effort and schedule is currently calculated with a Dynamic COCOMO approach using the COCOMO II.2000 calibration (Boehm *et al*. 2000).

The volatility impact multiplier is an extension of COCOMO for the SISOS situation. It is extrapolated from the current model and partially based on expert judgment. Other parameterizations relying on expert judgment include the average change analysis effort, life cycle timing multiplier and amount of field issues. They will be updated based on empirical data we are collecting.

3. SAMPLE SCENARIO AND TEST CASE RESULTS

We have put the model through multiple scenarios to assess project options. Here, we demonstrate a seemingly simple scenario and show how it plays out to be a rather complex trade off situation. The scenario being demonstrated is for developing two successive increments of 15 capabilities each, with a nondeferrable change coming midstream which has to be handled. The agile team size is varied from 2 to 20 people in the test cases.

A capability change comes in at month 8 and is processed by the agile team. The change is nondeferrable as it needs to be in Increment 1. An illustration of how the system responds to such a volatility pulse in Increment 1 is given in Figure 8. In the figure legend, '1' refers to the increment number 1. An unanticipated set of changes occurs at month 8, shown as a *volatility trend* pulse. The changes immediately flow into the level for *capability changes*, which then starts declining to zero as an agile team works it off per the average change analysis effort of four person-months.

The change is nondeferrable and it becomes incorporated into the Increment 1, so the *total capabilities* for the increment increases. As the new capabilities become required for Increment 1, the development staffing responds to the increased scope by dynamically adjusting the team size to a new level.

However, the different team sizes will analyze the change at different rates. Figures 9–11 show results

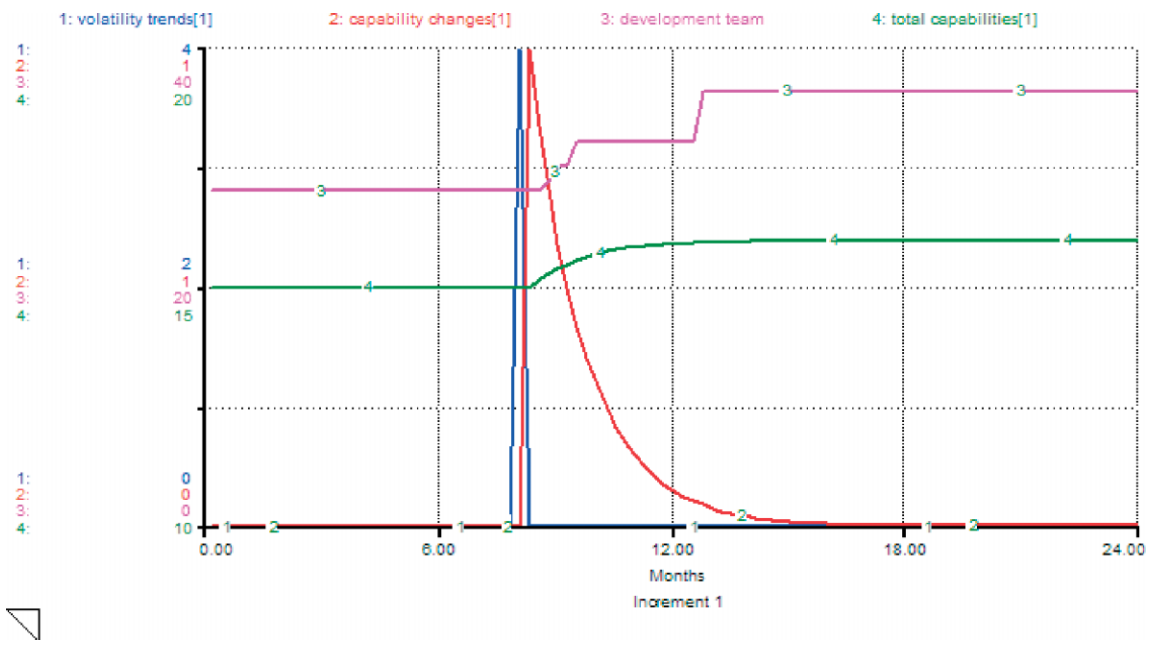

Figure 8. System response to volatility – increment 1

DOI: 10.1002/spip

of the test cases for varying the agile team size. If the team size is too small then it will not even make it into Increment 1. The agile team cannot even process the change in time for Increment 1 with two to four people. In these cases, the new capability for Increment 1 is processed too late and goes into Increment 2 instead. But with six people on the agile team it can make it in time. The larger team size will process the change and incorporate it faster; hence, the effort and schedule for Increment 1 improves with an agile team size of six or higher.

A summary of the resulting dynamics for these cases are:

- Two agile people take the longest time to process the change. The capability is ready for implementation late in increment 2 and incurs a substantial effort penalty per the life cycle timing effort multiplier. Its cost is greater than if the capability was ready at the start of Increment 2. Increment 2 also incurs the volatility multiplier effect and costs more than its original baseline. Increment 1 executes to its baseline plan.
- Four agile people process the change faster, but still not in time for Increment 1. The Increment 2 life cycle timing impact is not as great as the case for two people. Increment 1 again matches its original baseline plan.
- Six people can process the change fast enough to get it into Increment 1, but it comes in late and incurs the late life cycle timing effort multiplier. Increment 2 does not vary from its baseline
- Eight people are faster than six, so the Increment 1 losses are reduced and Increment 2 is still at its baseline.
- Ten people similarly improve upon the team of eight. Increment 1 construction effort is reduced relative to the team of eight because the change comes in sooner. Increment 2 remains unchanged.
- As the agile team grows from 10 to 20 people, the impact of late life cycle changes decreases less than the added effort due to more agile people. With larger agile team sizes, the effect of the life cycle timing effort multiplier in Figure 6 moves from the steep portion at the top down to the shallow part where changes come early in the increment.

Figure 9 shows the components of effort as a function of the agile team size. The effort for the constant size agile team increases linearly as its size grows. It is the smallest relative component of the effort when under ten people. The Increment 1 construction effort (Development + V&V) at two and four agile people is the baseline effort since the capability change did not reach it. It is the highest cost at six people due to the late life cycle effort multiplier effect, and that penalty is reduced at eight and ten people since it comes in sooner. Above ten people, the total cost raises slowly due to the agile team size increasing and impact of lateness decreasing, as previously described.

Roughly converse effects are shown in the effort function for Increment 2. At two and four people, the cost of Increment 2 is very high because the small agile team takes a long time to analyze the change. It comes in latest with the smallest team incurring a very high penalty for being late in the life cycle. Six through twenty people they can process the change before Increment 2 starts, so the effort is the same for those cases. The Increment 2 effort exceeds its original baseline due the added volatility of the single capability.

Figure 10 shows the test case schedule results for both increments and overall. They are analogous to the effort functions shown in Figure 9 for the two increments. Their shape mirrors the effort curves at approximately the 1/3 power. This empirical correlation between software effort and schedule is described in (Boehm *et al*. 1998).

Investigating Figure 9 and Figure 10 may lead to the erroneous conclusion that four people is the optimum, since it produces the least overall cost and schedule. The total effort for four agile

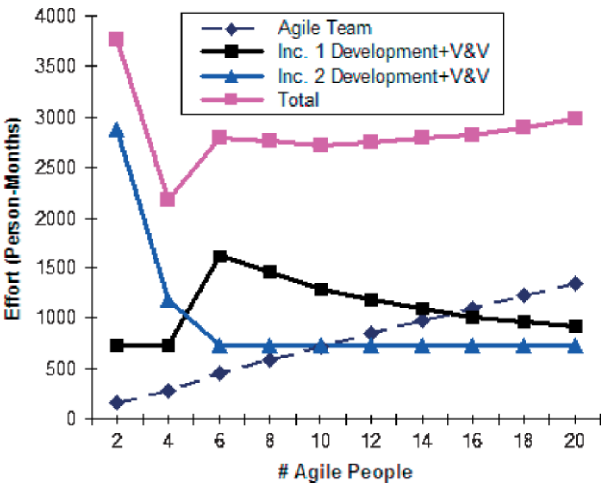

Figure 9. Effort *versus* Agile team size

Softw. Process Improve. Pract., 2007; **12**: 461–473
DOI: 10.1002/spip

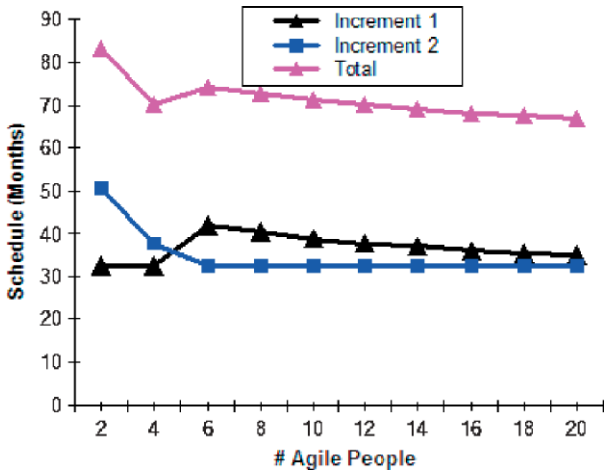

Figure 10. Schedule *versus* Agile team size

people may look promising since the change was effectively deferred and did not incur life cycle timing losses. However, the bigger picture must consider the mission value losses incurred by the smaller teams.

3.1. Software Value Considerations

These results are incomplete without casting them in a value-based framework. The model outputs described so far cover development cost and schedule impacts, but not the mission value of the software intended for the increments. Our project context of an extremely large government defense project places value on the utility of the mission capabilities in terms of wartime performance measures. However, the general technique also is applicable to commercial projects where the software utility is more directly quantified in terms of monetary return on investment (ROI). An example of a system dynamics model, for commercial business trade offs of process and product decisions using business value measures such as market share, sales and ROI, can be found in (Madachy 2005)

The mission capability values will be simplified using the conservative assumption that they are at least equivalent to their cost of development. Presumably, in any software venture, the desired capabilities are worth at least their cost, otherwise there is no reason to take on the project. Though there are exceptions to knowingly lose money on development as an investment to gain better chances for larger future gains (e.g. internal product research and development for a potentially large external market).

In our project case, the required capabilities are direly needed to address real battle threats, and they are carefully planned out in time. Real mission value will be lost if they are not delivered on time. Our assumption of their value being equal to development cost is unquestionably very conservative; in reality, their utility values are judged to be several times more. Their utility does not expire 100% at the planned time of delivery, but for simplification, we will assume so in our analysis and still demonstrate a valid picture of the situation. Our two simplifying assumptions also balance out each other a bit, by underestimating the mission value and possibly overestimating the full discrete loss at intended delivery time. Now we have accounted for a discrete mission value loss in Increment 1, but there are even more time-dependent losses to account for since Increment 2 stretches out. This analysis can be further refined to account for the loss of value as a diminishing function over time, and the results will vary a bit.

Figure 11 shows the value-based analysis for handling the capability change in terms of all costs, and can be considered a minimum view of the actual value loss per our conservative assumption previously described. With a more global value consideration, it is evident that four people are no longer the optimum. According to these results, a larger team size is optimum and the losses decrease very slightly when increasing the team size up to ten people. The region of slightly rising effort between 10 and 20 people produces the sweet spot cost minimum at ten people.

These results account for the life cycle timing multiplier, volatility multiplier, increment delay losses and a quantification of software capability value. The model shows that a sufficient level of agile rebaseliners is necessary, or the cost and schedule for the project increases substantially. The losses are even worse when considering the mission value. Enough must be on-board and productive enough to analyze the changes in a timely manner. Otherwise, there could be a backlog of work to worry about at the beginning of a later increment that could have been resolved earlier by the agile team, or there are other losses. If we further refine upwards our conservative estimate of the mission value, the results will lead to the same decision.

DOI: 10.1002/spip

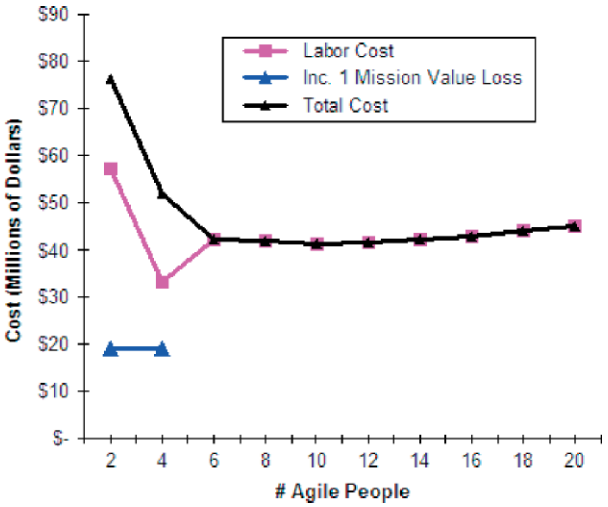

Figure 11. Cost *versus* Agile team size

Without explanation of these dynamic effects across increments, this apparently simple case of a single change may seem confounding at first with the multiple maxima and minima in the results. When adding the consideration of mission value, the results become even more cogent, explainable and grounded in practicality. Another corresponding example is to show the results for a single change destined for Increment 2 while holding all the experimental parameters the same. The results are similarly complex, and we are going forward with scenario permutations consisting of multiple capability changes.

4. CONCLUSIONS AND FUTURE WORK

Processes need to rethought for current and upcoming SISOS, and the outlined hybrid process based on the scalable spiral model appears to be an attractive option. The dynamic model will help to further refine the hybrid process and determine optimized variants for different situations.

Traditionally, it has been thought that agile processes are not suitable for very large software development projects, particularly ones with high dependability requirements. However, this approach addresses concerns by retaining large pockets of stability on the construction teams for development and V&V. We believe this hybrid approach would be applicable to programs across the size spectrum from small to very large. This is another aspect that we can eventually address with the model by descaling the simulated project and even changing its domain context.

This second major iteration of the model provides interesting results. It shows that if the agile team does not do their work rapidly enough, then developers will have to do it at a higher cost due to changes late in the life cycle, and mission losses may also occur. Further experiments are underway to vary the deferral percentages and perturbation smoothing policies, include rework, and constrain the staff sizes for development and V&V.

Both the hybrid process and the model will be further analyzed and evolved. Various improvements in the model are already identified and briefly discussed below, but further changes will come from the users of the model and additional empirical data will be used to calibrate and parameterize the model.

The set of test cases we demonstrated varied agile team size, but other dimensions to vary are the deferral percentage and perturbation smoothing policies. Additionally we are simulating all five increments with volatility occurring in more than one increment in experiments.

This version of the model uses step function staffing profiles that adjust dynamically to changes. Another version uses Rayleigh curves for more realistic staffing patterns that adjust on the fly to midstream changes. These models will be integrated to allow the user to specify the type of staffing.

In the current test cases, only the optimum personnel levels are used for the development and V&V, but in reality there may be staffing constraints. The model will be refined so that the users can constrain the development and V&V staff sizes. Another set of tests will compare trade offs between different agile team staffing policies (e.g. level of effort *vs*. demand-driven).

Patterns of changes and change policies will be experimented with. We will vary the volatility profiles across increments and demonstrate kickback cases for capabilities flowing back up the chain from the developers or V&V'ers. Additionally, we will model more flexible deferral policies across increments to replace the current binary simplification of allocating changes to the current or next increment.

We are implementing an increment 'balancing policy' so that large perturbations for an increment can be optionally spread across increments. The

DOI: 10.1002/spip

user can specify the number of increments and the relative weighting of capability across them. This balancing policy will tend to smooth large fluctuations in staffing and minimize costly ramp-ups (but always traded against lost mission value).

Value-based considerations should be part of the overall process analysis. We showed that accounting for mission/business value of software capabilities made the results more practical, coherent, and explainable in terms of finding trade off sweet spots. We will refine the value-based considerations to quantify the time-specific value of capabilities instead of assuming full discrete losses.

Parts of model have been parameterized based on actual empirical data, but not the change traffic. We will be getting actual data on volatility, change traffic trends and field issue rates from our USC affiliates and other users of the model. Data for the change analysis effort and volatility cost functions will also be analyzed.

After we get change data to populate the model and make other indicated improvements, we plan to use it to assess increment risk for a very large scale SISOS program. It will also be used by contractors on the program in addition to our own independent usage to assess process options.

We also plan to apply it to other projects we are involved with, and the model will be supplied to our USC-CSSE industrial affiliates for assessing and improving their processes. Some of these users will also provide us with additional empirical data for parameterizing the model.

REFERENCES

Abdel-Hamid T, Madnick S. 1991. *Software Project Dynamics*. Prentice-Hall: Englewood Cliffs, NJ.

Boehm B. 2005. Some Future Trends and Implications for Systems and Software Engineering Processes, USC-CSE-TR-2005-507.

Boehm B, Turner R. 2004. *Balancing Agility and Discipline*. Addison Wesley: Boston, MA.

Boehm B, Brown A, Basili V, Turner R. 2004. Spiral acquisition of software-intensive systems of systems. *CrossTalk*, May 4–9.

Boehm B, Egyed A, Kwan J, Port D, Shah A, Madachy R. 1998. Using the WinWin spiral model: a case study. *IEEE Computer* **31**: 33–44.

Boehm B, Abts C, Brown A, Chulani S, Clark B, Horowitz E, Madachy R, Reifer D, Steece B. 2000. *Software Cost Estimation with COCOMO II*. Prentice-Hall: Upper Saddle River, NJ.

Fernández-Ramil J, Capiluppi A, Smith N. 2005. Understanding open source and agile evolution through qualitative reasoning. *Proceedings of the 6th International Workshop on Process Modeling and Simulation*. IEE: St Louis, MO.

Ferreira S. 2002. Measuring the effects of requirements volatility on software development projects. PhD Dissertation, Arizona State University.

Ferreira S, Collofello J, Shunk D, Mackulak G, Wolfe P. 2003. Utilization of process modeling and simulation in understanding the effects of requirements volatility in software development. *Proceedings of the 4th International Workshop on Software Process Simulation and Modeling*, Portland, OR.

Ford D, Sterman J. 2003. Overcoming the 90% syndrome: Iteration management for reduced cycle time in concurrent development projects. *Concurrent Engineering Research and Application (CERA) Journal*, **11**: 177–186.

Madachy R. 2005. Software process and business value modeling. *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling*. IEE: St. Louis, MI.

Madachy R. 2007. *Software Process Dynamics*. Wiley/IEEE Computer Society Press: Hoboken, NJ.

Powell A. 2001. Right on time: measuring, modelling and managing time-constrained software development. PhD Dissertation, University of York.

Powell A, Mander K, Brown D. 1999. Strategies for life cycle concurrency and iteration: a system dynamics approach. *Journal of Systems and Software* **46**(2–3): 151–161.

Repenning N. 2001. Understanding fire fighting in New product development. *Journal of Product Innovation Management* **18**: 285–200.

Sycamore D. 1996. S improving software project management through system dynamics modeling. MS Dissertation, Arizona State University.

Tvedt J. 1996. An extensible model for evaluating the impact of process improvements on software development cycle time. PhD Dissertation, Arizona State University.

DOI: 10.1002/spip