ICSSP 2013 Proceedings of the 2013 International Conference on Software and System Process
Pages 88-97
ACM New York, NY, USA ©2013

Modeling User Story Completion of an Agile Software Process

Dan X. Houston
The Aerospace Corporation
P.O. Box 92957
Los Angeles, California, U.S.A
1.310.336.5000
dan.houston@aero.org

Douglas J. Buettner
The Aerospace Corporation
P.O. Box 92957
Los Angeles, California, U.S.A.
1.310.336.5000
douglas.j.buettner@aero.org

ABSTRACT
Although some researchers have studied agile techniques of software development using simulation, simulation studies of actual agile projects are difficult to find. This report on an industrial case study seeks to address this need by presenting an experience of modeling and analyzing an agile software development process using discrete event simulation. The study, undertaken for software process improvement, produced analyses that provided project management with process capability information. First, a sensitivity analysis used a designed experiment to measure the dominant factors in user story productivity. Second, a response surface provided information on the process tolerance for defect rework. Finally, a scenario comparison supported a management decision on sprint usage.

Categories and Subject Descriptors
K.6.3 [**Management of Computing and Information Systems**]: Software Management – *software process.*

General Terms
Management, Measurement, Experimentation.

Keywords
Agile software process simulation; simulating agile process; user story completion; user story productivity.

1. INTRODUCTION
Software process simulation (SPS) has been a topic of academic interest since the 1980s as researchers have sought to gain better understanding of the dynamics of software development projects. At that time, software projects were plan-driven and SPS was able to describe these projects well, as demonstrated by a growing body of literature [1, 2].

In the 1990s, various alternative software development techniques arose in reaction to criticism of plan-driven processes. In 2001, some of the leading proponents of these techniques came together and, seeking commonality in their approaches, articulated four values of agile software development that were later elaborated into twelve principles [3]. Since then, agile software development has been embraced widely in industry, though at times without sufficient attention to the specific disciplines and techniques that gave rise to it.

Unfortunately, SPS and the agile movement both reflect a gap in the field of software development. SPS has remained largely an academic topic with relatively little application in industry; on the other hand, agile development arose in industry, and research on its tenets and practices followed its widespread industrial acceptance. Consequently, a very limited amount of work has been undertaken in applying SPS to agile projects [1, 2].

The Aerospace Corporation (Aerospace) has successfully employed process simulation to study a number of software development problems. Though its software process simulation studies are sometimes conducted as research projects, they are often conducted as engagements with customers and contractors to guide and support program decisions. For example, an Aerospace simulation study was used successfully to forecast system integration duration, to quantify the effects of purchasing expensive test equipment on project completion, and to support the contractor's software management in their project staffing decisions [4]. In [5], a simulation study of the effects of quality-inducing activities on rework effort and timing was used to confirm the factors that required retesting late in the development stage of a satellite's embedded software, thus reducing risks of fielded defects. A related study [6], using a different model, profiled defect density during development, used COQUALMO factors to conduct a cost-benefit analysis, and identified the best process improvements. As a final example, an Aerospace research study on concurrent development [7] demonstrated the degrees to which increasing concurrency can be accompanied by increased risks to product quality and project costs.

Contractors working for Aerospace customers have been adopting agile software development techniques and processes. As a trusted partner with its customers, Aerospace has studied agile development for years. Recently, Aerospace began exploring the intersection of SPS and agile software development with simulation of an agile process in this case study.

Just as agile processes are different from plan-driven processes, models of the two types of processes are quite different [8]. Plan-driven processes start with eliciting requirements (that describe functionality) and seek to implement them within a schedule, though the schedule is less important than the content and is subject to slippage. Agile processes start with a timeframe and seek to implement a set of user stories (that describe

*ICSSP'*13, May 18–19, 2013, San Francisco, CA, USA

functionality), though the number of user stories implemented is less important and subject to slippage. At the risk of oversimplification, one might sharpen this contrast by considering plan-driven processes as "functionally-boxed" and agile processes as "time-boxed." The contrast is exhibited in modeling: plan-driven processes are modeled as work products passing through transformational phases of various durations until the flow is completed, while agile processes are represented as work products passing through fixed transformational time periods. In both, flows are subject to resource constraints.

The distinct differences in modeling the two types of development processes accompany differences in modeling purposes. While modeling of plan-driven processes is often initiated with questions of duration and product quality, we found that modeling of an agile process was initiated by a question of productivity: "How many user stories can be implemented during a quarter?"

In this paper, we describe an Aerospace discrete event model and simulation study that was used to support management of a contractor's agile software project. Both the customer and the contractor had become concerned about the lack of predictability in contractor deliveries and they wanted to understand the sources of variation in the deliveries and best steps to improve delivery quality. The study focused on producing useful knowledge for the customer and contractor rather than general knowledge about agile development. Nonetheless, it produced more widely applicable subject matter and methodological lessons. In addition, this report seeks to contribute to bridging the gap between research and practice in SPS and agile development, as well as to offer an example of using SPS to identify both practical needs and improvement opportunities in an industrial agile software development.

The remainder of this paper is structured to report the study results starting with context among published studies and proceeding to conclusions. Results of a literature search are reported to provide context and background to the study. The study approach and organization are described, followed by a description of the subject process and modeling scope. Sections on model calibration and outputs elaborate on the model. Model usage is related in two sections, one on sensitivity analysis and the other on scenarios. Finally, conclusions and opportunities for further work are offered in the last section.

2. LITERATURE REVIEW

A literature search for simulation studies of agile processes produced a list of 23 publications reporting on 16 studies (Table 1). The majority of these studied eXtreme Programming (XP) practices. In fact, a 2008 review of empirical studies of agile software development methods found that most research in this area had focused on XP, one of the earliest and best-known of the agile methods. "The studies investigated XP almost exclusively …. A clear finding of the review is that we need to increase both the number and the quality of studies on agile software development." [9]

Though simulation was applied to software processes in the 1980s, and the first processes that came to be known as agile emerged in the mid-1990s, it was only in 2003 that the first papers appeared in which software process simulation was applied to the study of agile processes. Table 1 lists works in the open literature describing simulation models of agile processes. In some cases, multiple works describe the same modeling study. The table includes the agile method or practice studied, the type of simulation model employed, and the purpose of the study.

Of the modeled XP practices, the effects of Test-Driven Development (TDD) and Pair Programming (PP) on effort, size, quality and released functionality have garnered the most attention. In a series of papers, researchers at the University of Cagliari found that user story completion for an XP project was described by a power law distribution, and that the number of USs completed was negatively impacted by the use of TDD and PP to varying degrees, as was defect density in the resulting code [17, 18, 19, 20].

Other researchers have also modeled XP's PP and TDD effects [11, 13, 29]. As an example, Kuppuswami and colleagues created a generic XP model that contains other individual XP practices in order to investigate the effects of these practices on cost [10, 11]. Meanwhile, Ur and colleagues developed an open source Matlab discrete event model to compare various testing strategies to TDD and PP in particular, in addition to test automation [22].

Modelers have also investigated schedule pressure effects on the dynamics of iterative development cycles [28]. They found a "U-shaped" pattern with better performance using 50 working-day iterations as opposed to 20 working-day development iterations. Researchers have also investigated the amount of work completed by Scrum and Lean-Kanban as compared to the waterfall method, and suggested that agile methods would significantly outperform plan-driven methods [30].

Cao, Ramesh and Abdel-Hamid created an integrative system dynamics model of agile software development for investigating refactoring and PP practices [16]. They suggest that refactoring is cyclic in nature and increases in the later phases of development and that the number of user stories delivered is highly dependent on the level of refactoring.

All the works listed in Table 1 are products of academic research with varying usage of actual project data for model calibration. Of the sixteen modeling works listed, ten are concerned with XP practices. Twelve of the models are systems dynamics models (SDM). Clearly, most investigators are interested in the effectiveness of agile practices relative to content-driven processes. Some of the studies simply reported model development, but others reported conclusions regarding agile practice benefits and drawbacks. When investigators reported conclusions on practices, they ranged from attributing unqualified benefits to agile practices to stating that agile practices may be beneficial depending on circumstances.

3. STUDY APPROACH AND ORGANIZATION

In contrast with research efforts described in the previous section, this paper reports an industrial case study undertaken with an agile software development team to model their development process and propose process improvements.

An industrial study has an advantage of focusing on a single instance of a process and having direct contact with process owners and actors. These factors facilitate the modeling process through elicitation of process data—both qualitative data about the structure of the process and quantitative estimates of process parameters values—and ground it in actual practice. This focus may limit generalizations from a study, but an accumulation of

Table 1. Software Process Simulation Studies of Agile Methods

Source	Agile Method	Simulation Method	Study Purpose
Kuppuswami *et al.* [10, 11] Vivekanandan [12]	XP	SDM (Vensim)	Investigate effects of XP practices on software development productivity and cost of changes
Wernick & Hall [13]	Pair Programming	SDM	How does pair programming impact the long-term trend in software product evolution?
Mišić *et al.* [14]	XP	SDM	Analyze performance of XP. No results reported.
Cao [15] Cao *et al.* [16]	Refactoring & Pair Programming	SDM	Understand effects of agile development practices
Cau *et al.* [17]	XP, TDD	SDM	Measure effectiveness of XP practices, particularly productivity and quality using TDD
Turnu *et al.* [18] Turnu *et al.* [19]	TDD	Continuous using difference equations	Demonstrate the effects of adopting TDD
Melis [20] Melis *et al.* [21]	Pair Programming & TDD	Hybrid in Smalltalk	Evaluate the effectiveness of XP
Ur *et al.* [22]	Pair Programming & TDD	DES	Demonstrate that a model can be used to evaluate software process changes
Yilmaz & Phillips [23] Phillips [24] Phillips & Yilmaz [25]	Increments & iterations	ABM	Model the role of human organization in software development
Chichakly [26]	Nightly builds, test first	SDM	Compare effects of agile practices & waterfall process
Wu & Yan [27]	XP	SDM	Assess performance of XP
van Oorschot et al. [28]	Iterations	SDM	Examines the relationship between schedule pressure, level of agility, and project outcomes.
Yong & Zhou [29]	XP	SDM	Evaluate the effectiveness of XP
Cocco *et al.* [30]	Scrum and Lean Kanban	SDM	Compare Scrum and Lean-Kanban with a waterfall process.
Kong *et al.* [31]	Critical Feature Method	SDM	Measure effect of schedule pressure on maintenance effectiveness
Faynshteyn *et al.*[32]	Pair Programming	SDM	Compare program value between waterfall and pair programming

such studies can support refinement of and validation of academic studies.

The study was initiated by a member of the customer's program office, a member familiar with SPS. His request for a simulation study was made with the intent of improving the quality of deliveries to the customer while reducing their cost.

This study took place over a two month period, during a release cycle, and was structured around visits to the contractor's site. In the first visit with the project manager and the development team leads, software process simulation was explained. Models from previous studies were presented and the results of those studies described so that the manager and team leaders could begin to pose specific process questions that could be answered through the study. The team leads quickly focused on their productivity because they were concerned about activities that distracted them from software production work.

Though this concern may seem a simple one, they wanted insights for balancing competing demands, which they faced daily, and they wanted understanding of how poor design and quality enhancement choices would produce costs in terms of rework and unplanned work. With that purpose in mind, the elicitation process began with asking them to describe their software development process, including the workflow, the activities and sequence of events, the resources, and how resources are applied.

After the first meeting, the analyst developed a preliminary model in ExtendSim, considering level of abstraction, scope, types of data required, and the kinds of questions that could be answered with the model.

A diagram of the preliminary ExtendSim model was taken to the second meeting with the team leads where it was used to continue the elicitation of process information. Walking through the diagram produced clarifications and changes to the representation of the workflow structure; the diagram was marked up with these changes. That meeting produced a decision to focus on software production activities and the factors contributing to the project's ability to complete user stories during any given 3 month release cycle. That decision provided for simplification by leaving user

story and task definition out of the model scope. Also, two roles were found to be necessary, developers and team leads. This meeting also considered parameters necessary for the model. These were selected and values for some of the parameters were discussed. Assuming that most parameters values are triangularly distributed, most data was elicited in terms of minimum, maximum, and most likely values.

The team leads reached consensus on all descriptions of workflow structure and parameter values. Agreement was often tentative because they had not previously discussed some of the process issues, particularly quantification of parameter values like task durations. Nonetheless, they could recall instances and posit values.

Information recorded on the model diagram in the second meeting was used to develop the model further and produce a running version. Some data was entered into the model and it was examined for additional data needs.

The third meeting with the team leads walked through the revised model diagram, confirming the workflow structure and continuing the discussion of parameter values. At that meeting, one of the leads also provided results of prior releases.

After the third meeting, the model was refined and the data from prior releases was used to calibrate the parameter values. Once the model was calibrated, sensitivity analysis was performed and scenarios were run. These results were interpreted and compiled into a presentation for a briefing delivered in the fourth meeting.

4. PROCESS DESCRIPTION AND MODELING SCOPE

Modeling focused on the production of user stories during a release cycle, including inputs most likely to affect that production. Figure 1 is a model diagram showing two workflows, represented by the activities (blocks and block arrows) and arcs (block arrows indicate transitional activities: process entry, exit, or transition to a group). The primary workflow represents the development of software from user stories. Prior to the start of a release cycle, each user story is decomposed by team leads into tasks, usually on the order of 8 hours each. (Modeled tasks are not characterized further, so they are assumed to be ordered such that work can proceed.) These tasks are translated into working software through a set of three activities—Team Lead and Developer Work, Developer Work, and Check-In—that are enacted six times serially in two week sprints. After Check-In, software and test materials produced from the tasks are used in the Test activity. After Test, software is delivered to Acceptance Test at the customer site.

Discrepancy reports (DRs) can be produced during developer work, during check-in, during user story testing, or during acceptance testing. Regardless of the activity in which they are produced, they are sent back (light arcs in Figure 1) to the Developer Work activity, producing rework in that and subsequent activities.

The second workflow represents tasks that do not contribute to the production of software for the current release. This latter workflow includes activities such as proposal work, planning for future releases, and researching and developing new tools or process improvements. Management prioritizes these other tasks higher than development tasks and this second workflow is included because it draws on the same resources that are applied to the first workflow.

Developers are the resource primarily responsible for transforming user stories into software. They were modeled as a resource pool without distinguishing individual characteristics due to limitations imposed on the study. This modeling choice is based on the facts that a specific development process was being modeled and that the process employed a single pool of developers. Variation in individual productivity was accommodated with a random variable for task duration.

Team leads were included because they also work on development tasks in conjunction with developers. System and test engineers were not included because they do not have direct

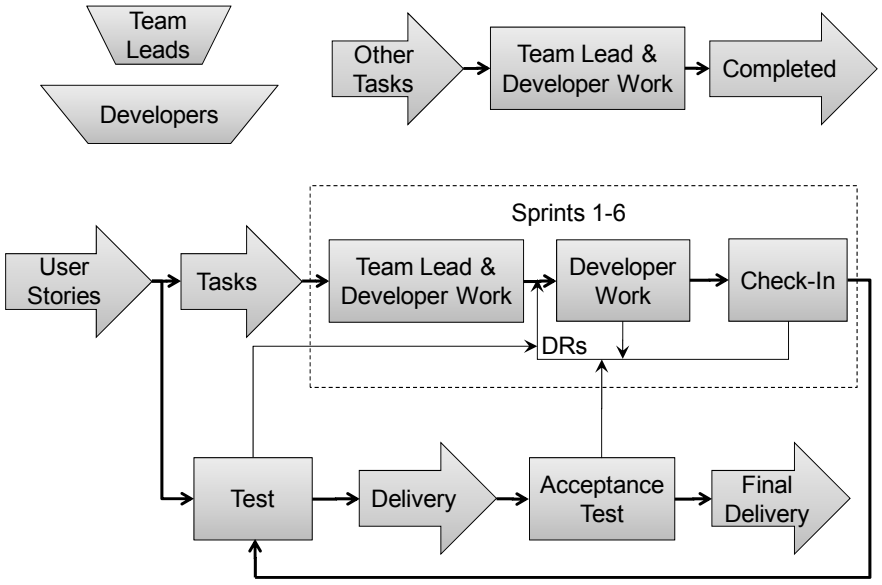

Figure 1. Model Diagram

Table 2. Model Factors

Model Factor	Type of Factor
Number of Software Developers	Constant for each release cycle
Number of Team Leads	Constant for each release cycle
Software Developer Allocation Rule	By task priority
Queue Prioritization	First in, first out
User Stories Priority	2
Number of DRs at Start	0
DR Priority	2
Number of Tasks/User Story	Triangular distribution
Required Resources per Task	1
Task Duration	Triangular distribution
DR Duration	*Uniform distribution*
DRs per User Story Found in Sprints	*Uniform integer distribution*
P(LateCheckIn)	0.5
LateCheckIn	Triangular distribution
Last User Story Sprint	5
System Test Duration	Triangular distribution
User Stories producing a DR	Constant fraction
DRs that are Blocking	Constant fraction
Acceptance Test Installation	Triangular distribution
Acceptance Test Duration	Uniform distribution
Number of DRs produced by Acceptance Test	Uniform integer distribution
Req'd Resources for each Acceptance Test DR	1
Fix Duration for Acceptance Test DRs	Triangular distribution
Acceptance Test ReInstallation	Triangular distribution
Acceptance Test ReTest Duration	Uniform distribution
Time Between Arrival of Other Tasks	*Uniform distribution*
Other Task Req'd Resources	*Uniform distribution*
Other Task Duration	*Uniform distribution*
Other Task Priority	1

contact with development tasks during release cycles and they are assumed to be available as needed, thereby not presenting a delay in user story production.

The duration of a release cycle is three months. Within each release cycle, six sprints of two weeks each are used for development. At the beginning of each sprint, each developer is assigned two tasks and as each task is completed, they receive a new task. A team lead works with each developer at the start of a task for an initial fraction of the task development time. Developers check-in tasks as they are completed. As tasks are batched to complete user stories, testing is conducted on the user stories. When user story testing is complete, the release is delivered to the customer site for acceptance testing. After acceptance testing and any rework from it, the release is delivered for installation.

5. MODEL CALIBRATION

Table 2 lists the model factors and types of values after calibration. All distributions in italics were triangular prior to calibration. However, the model was unable to reproduce the actual variation in number of user stories produced per release cycle. Therefore, five of the inputs were changed to uniform distributions to account for more of the observed process variation. Discrete event modelers often use triangular, uniform, or PERT distributions in the absence of recorded data. Distributions with a measure of central tendency (triangular and PERT) are preferred, but uniform distributions provide more variation and recognize that modes are unknown.

Prior to this study, the team demonstrated little interest in process efficiency. Consequently, values for only two process outputs, number of user stories produced and number of DRs fixed, had been recorded. Actual data for five releases was available for calibration. Actual outcomes were compared to the ranges of outputs from the model. Table 3 shows the results, by release, of calibrating the model to actual process performance. The table entries indicate the relationship of the actual process outputs relative to the output ranges produced by the model (within, lower than, or higher than the range produced by the model).

Table 3. Model Calibration Results

Release	1	2	3	4	5
Number of User Stories Produced	Within	Low	High	Within	Within
Number of DRs Fixed	Within	Within	Within	Within	Low

For the ten actual results, model output distributions covered 7 of them. For Release 2, the actual number of user stories produced was well below the model output and for Release 3, it was well above the model output. Even for some of the actual results covered by model results, the actual amount is near the edge of the model range. These results suggest that the development process has more variation than the model.

6. MODEL OUTPUTS

Running the calibrated model 500 times generates data to produce distributions for number of user stories completed, number of tasks completed, number of DRs completed, number of other tasks completed, effort applied to other tasks, final delivery times, developer utilization, and team lead utilization. Analysis of histograms of each of these outputs produced a number of observations, including the following:

- Releases are difficult to plan because the number of user stories produced for each release varies widely.
- Project management has a substantial opportunity for reducing rework effort by reducing the consistently high number of DRs.
- The large amount of effort given to other tasks came as a surprise because these tasks and their effort are not tracked.
- Final deliveries, after all rework, can take place between the third and fifth week after the end of a quarter. Management would like to decrease that delivery delay.
- Developer utilization is consistently very high, approximately 90%.

7. SENSITIVITY ANALYSIS

Six factors were selected for experimentation. They were known to be highly variable, team leads had expressed uncertainty about their actual values, and they were recognized as having a strong likelihood of affecting the three outputs of interest: (1) the

number of user stories developed in a release cycle, (2) the number of DRs completed during a release cycle, and (3) the final delivery time of a release. The factors selected for a statistical experiment are listed in Table 4. The ranges for these factors recognize extreme operating ranges. Task effort provides an example of a factor with a nominal value of 8 hours, but one that may actually have a much wider range of values, from 2 to 24 hours.

Table 4. Factors and Ranges of Values for Sensitivity Analysis

Factor	Range of Values
Effort required per development task (Task effort)	2 to 24 hours
Effort required per DR (DR effort)	2 to 80 hours
Number of DRs found per user story during sprints (N DRs dev)	0 to 12
Time between arrival of other tasks (TBA other)	3 to 10 days
Number of people required for each other task (N per other)	0.5 to 2 persons
Effort required per other task (Other effort)	2 to 80 hours

The extreme values of the ranges were used to produce 32 model configurations.[1] Each configuration was run four times. The analysis of variance performed on the resulting data indicated that all factors were significant; however some were much more influential than others with respect to each output. In Table 5, factors that have a significant influence on an output are indicated by a number in the output's column. The number is the rank order of the factor's influence. For example, development task effort is not significant for the number of user stories produced in a release cycle, but is most influential for the number of DRs completed and the completion time.

Table 5. Relative Influence of Factors on Primary Outputs

Factor	N User Stories Developed	N DRs Completed	Completion Time
Task effort		1	1
DR effort	1	3	
N DRs dev	3	2	1
TBA other	5	6	
N per other	4	5	
Other effort	2	4	

Table 6 provides more detail for the relative influence of factors and factor interactions on the number of user stories produced in a release cycle. The influence has been normalized relative to the influence of task duration. For example, DR effort is six times more influential than task effort in affecting the number of user stories produced in a release.

The model revealed many interactions between factors. An interaction is an effect on an output of a combination of factors, in which the influence of each interacting factor depends on the setting of the other factor(s). An interaction also represents a larger influence than each contributing factor has individually. Usually highly influential factors are the ones that produce interactions, but in some cases insignificant factors can participate in an interaction, such as with Task effort and DR effort above: the effort required for each development task does not exert a large influence on the number of user stories developed, but DR effort does. In addition, Task effort and DR effort interact to exert a substantial influence on amount of development. The number of DRs found in development also exerts a substantial influence on the production outcome. This means that any effort spent on reducing the number of DRs—and by extension, the effort applied to DRs—will yield benefits by affecting two factors and their interaction, which are all significant influences on user story development.

Table 6. Relative Influence of Factors on Number of User Stories Produced during a Release Cycle

Factor or Factor Interaction	Relative Influence on User Stories Completed
DR effort	6.0
Other effort	3.9
N DRs dev	3.8
N per other	3.3
Task effort * DR effort	3.2
N per other * Other effort	3.2
DR effort * N DRs dev	3.1
TBA other	2.8
TBA other * Other effort	2.7
TBA other * N per other	2.5
Task effort * N DRs dev	1.8
Task effort * N DRs dev * DR effort	1.6
Task effort	1.0

A response surface of the relationship between the number of DRs found during development (N DRs dev) and the effort applied to each DR (DR effort) provides further insight into the nature of DRs and their effect on production. Figure 2 is a response surface for this relationship as it affects user stories produced. In this figure, the mean values of the number of user stories produced are plotted against mean values of DR effort and of N DRs dev. The contours indicate that the number of user stories produced per release cycle can remain high as long as either the number of DRs or the effort per DR remains low, but as both factors increase user story productivity decreases. Specifically, this plot shows that user story production can tolerate 4 or fewer DRs per story until the effort per DR approaches 80 hours. However, when the number of DRs per user story increases beyond 4, then the user story production becomes very sensitive to the amount of effort applied to each DR.

The forgoing sensitivity analysis demonstrates that all experimental factors can be addressed to produce process improvements. Task sizing should be managed to produce tasks of about 8 hours effort each. However, management of other task effort and DR effort will provide the largest benefits.

8. SCENARIOS

In addition to the question of production capability, the development team raised a question about a potential change to their development process. The development team has treated Sprint 5 as the last sprint in which user story tasks are developed, allowing Sprint 6 to be devoted to rework. However, they have,

[1] This fractional factorial experiment used a 2^{6-1} (Resolution 6) design. This design provides discrimination of first and second order effects from higher order effects.

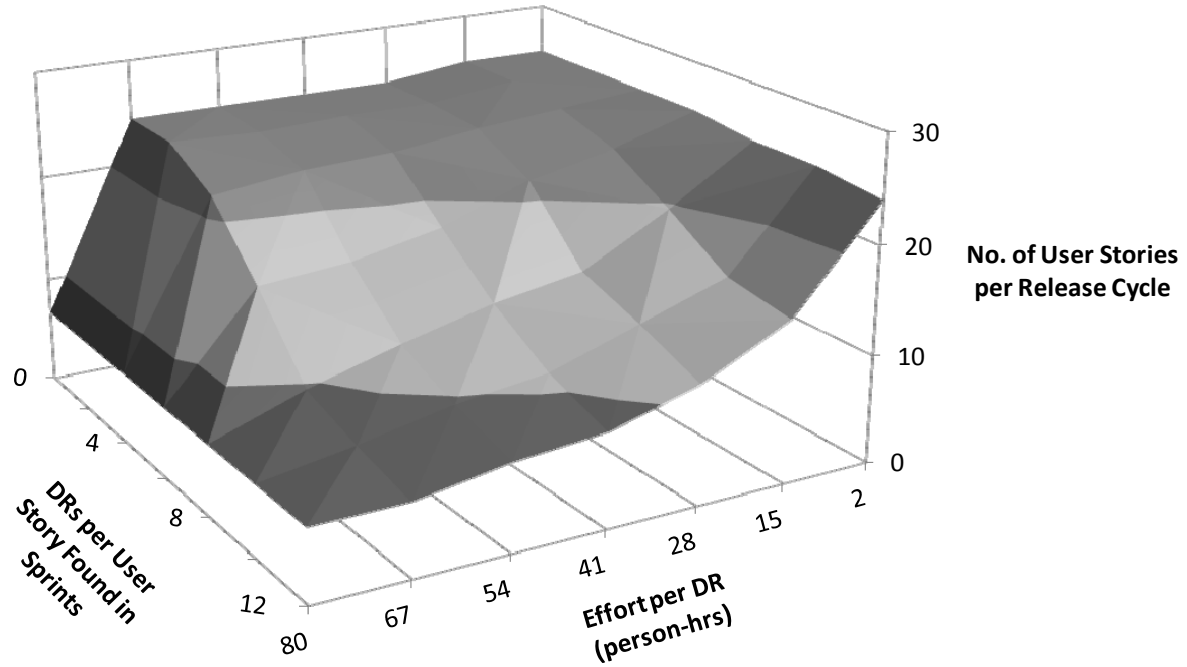

Figure 2. Effect of Interaction between DRs Found in User Story Development and Effort per DR

been discussing the use of both Sprints 5 and 6 for rework, making Sprint 4 the last one in which user story tasks would be developed. The effects of making Sprint 4 the last user story sprint were tested with the model by not allowing development task work after Sprint 4 and comparing results with the previous runs. When the last user story sprint was set to 4 rather than 5, the modeled release produced 3 fewer user stories on average, and completed 10 fewer DRs. Also, developer utilization dropped about 3% and team lead utilization dropped a little more than 3%.

To understand these results, it is helpful to recall the sources of rework. Some DRs are found in system testing and a few are found in acceptance testing, but most of the DRs worked in a release are found during work on development tasks. As a result, the number of DRs found during a release cycle is represented as a function of the number of user stories produced. Thus, fewer DRs will be completed when fewer user stories are produced and the 5th sprint capacity allocated to DRs could go unused.

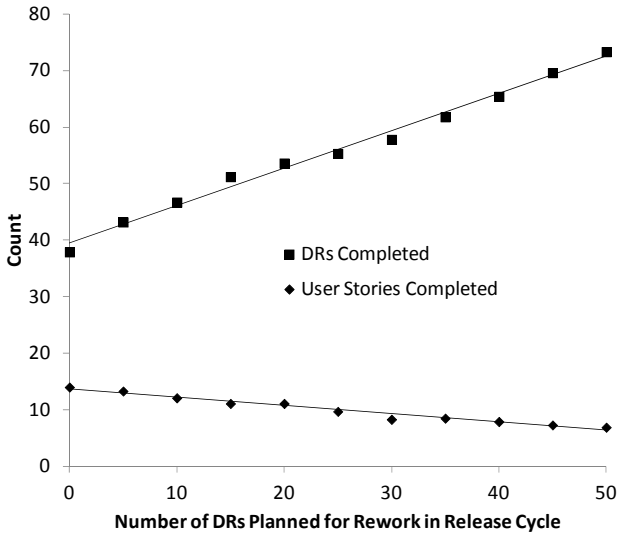

Figure 3(a). Work Completed vs. DRs Planned for Rework and Available at Start of Release Cycle

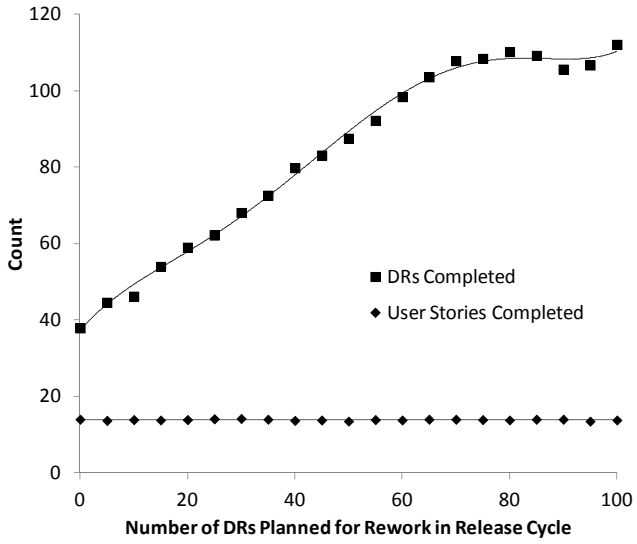

Figure 3(b). Work Completed vs. DRs Planned for Rework and Available at Start of 5th Sprint

Both the scenarios presume that no DRs are planned for rework in the release cycle. Though the model provides the option for planned DR work, neither scenario included it because no data was provided on how many DRs are typically planned for rework during a release cycle. In order to study this scenario further, a model experiment was run for a release cycle with last user story sprint of 4 and an increasing number of DRs included in the planned work. Two possibilities were considered: (a) planned DR work was available from the beginning of the release cycle, and (b) planned DR work was available at the start of the 5th sprint.

The following plots show that the timing of DR work is important when reserving the 5th sprint for rework. The upper plot, Figure 3(a), is from runs in which the DRs were available for rework from the beginning of the release cycle (each point is an average of 10 model runs). In that case, the number of user stories produced steadily decreases and the number of DRs completed steadily increases. The lower plot, Figure 3(b), is from runs in which the DRs were available for rework at the start of the 5th sprint (each point is an average of 10 model runs).

In the first case, the tasks and DRs compete for resources, so a trade-off occurs in production. In the second case, the DRs are held until the 5th sprint, so the number of user stories remains constant and the number of DRs completed rises with the planned DR work until the process capacity of the 5th and 6th sprints is reached and then levels off, which is at about 70 planned DRs in this example.

Therefore, a lesson from this scenario is that reserving a sprint for rework lessens user story production, but can work without affecting user story production further if the DRs planned for rework are held until the rework sprint.

9. CONCLUSIONS AND FUTURE WORK

The study produced useful insights into process improvement options for the development group. Specifically, two modeling challenges suggest very high variation in the development process being studied. First, uniform distributions, as opposed to distributions with a single measure of central tendency, were required to calibrate the model to actual performance. Second, even with calibration, output distributions did not cover all actual performance values or covered some values near a distribution's edge. The variation in the process may be due to input variation, or may be due to other factors that were not included in the model.

This study found two major sources of uncertainty in the subject process: other tasks that consume large amounts of developer effort and quality escapes. The effort estimated from the model for distractions to user story production is large and was questioned by the program manager. The inputs for this estimate came from members of the development team, but the question suggests that effort applied to other tasks should be recorded in order to produce a better estimate of its actual distribution.

The findings of this study were used to produce four major recommendations for process improvement.

1. Record a set of simple, low cost base measures that will provide better insight into process performance and provide a basis for future release planning with increased development predictability. These include number of developers and team leads per sprint, number of tasks per user story, estimated and actual effort per task and per DR, arrival time and applied effort for other tasks, and the activity in which each DR is found.
2. Reduce the number of DRs by choosing product quality-inducing activity for enhancement or implementation.
3. Manage effort applied to other tasks through an effort budget and task prioritization.
4. Adopt a paired comparison approach to task estimation [33, 34]. Though task sizing has not been a significant factor in user story completion, it has been significant for final delivery time. Reducing task size variation through improved task estimation may not be a high priority for the team, but it is a relatively low cost means of increasing process stability and predictability.

Though the recommendations were not surprising and may appear general, they were grounded in the group's own experience and elicited data, they addressed concerns expressed by the team leads at the time of the study, and they are based on quantified prioritization of factors. For these reasons, the recommendations helped the team to focus its process improvement efforts. In the release planning following presentation of the study results, the team took several actions. To improve team communication and status reporting, they added a work in progress board for tracking task assignments and task status, with updating at least weekly. To improve product quality, they added design reviews prior to implementation. To better manage distractions, they track "other tasks" on the work in progress board.

In addition to the process information provided to the customer and contractor, the study produced several methodological insights with wider applicability. Actors in a software development process often have difficulty sorting out important from unimportant issues, but the objectivity obtained by applying SPS clarifies the relative influence of factors. Though elicited data carries a degree of subjectivity, all the facts must fit together in a model such that it reproduces actual outcomes. When the pieces of the models do not fit together, then calibration is needed. This can mean identifying the most uncertain or subjective judgments and, with approval of the actors, revising them to reproduce actual process outcomes.

At the outset of this study, a manner of representing time-boxed development was not clear. One concern was the level of abstraction that could be achieved: Was it low enough to adequately represent the process dynamics and provide information useful to the team? On the other hand, was it high enough to overlook the highly variable and "noisy" actions of individuals? The foregoing results suggest that useful results do not necessarily depend on modeling developer characteristics, though doing so would likely provide additional valuable information.

Another concern was the limited data available and the heavy reliance on elicited data. However, as system dynamics modelers often point out, the structure of a process largely determines process outcomes. This observation certainly applies in a case study in which elicited data may be the most subjective element linking an observable process and recorded outcomes.

This study has demonstrated both a successful approach to and beneficial results from modeling and simulating an industrial agile software development process. The study required 200 hours from initial engagement through final briefing and distribution of a final report.

This work provides several possibilities for future work with the same project team. Model fidelity can be increased by adding the team's design reviews and by refining parameter values using new measures collected since the first engagement. Re-analysis can then be performed to assess whether the team is better able to control the scope of its deliveries. Model detail can also be added by representing individual developers with characteristics that can be used to condition productivity and task assignment. Similarly, modeling might be extended to other roles in the organization that indirectly influence software production, such as the project manager and user representatives. The scope of the model can also be increased to include planning activities of user story refinement and decomposition, and task estimation and assignment. The contractor has also expressed interest in a version of the model that can be used for early release planning.

While the quantitative results of this study are not expected to apply to other agile processes, the level of model abstraction and the generality of the recommendations suggest that other process may share similar concerns. Similar studies with other teams would provide opportunities to learn about the commonality of the issues found in this study. If other agile teams face similar challenges with regard to delivery predictability, rework, and distractions, then the time-boxed process model can be generalized to support a wider range of agile processes.

10. ACKNOWLEDGMENTS

This study was sponsored by The Aerospace Corporation's program funding and by its Software Acquisition Long Term Capability Development Project.

The authors thank the anonymous reviewers for their suggestions for refining this paper.

11. REFERENCES

[1] Zhang, H., Kitchenham, B., and Pfahl, D. 2010. Software Process Simulation Modeling: An Extended Systematic Review. International Conference on Software Process, Paderborn, Germany, July 2010. In: J. Münch, Y. Yang, and W. Schäfer (Eds.): ICSP 2010, LNCS 6195, 2010, 309-320. http://dl.acm.org/citation.cfm?id=1884258 Accessed Oct 11, 2012.

[2] Zhang, H., Kitchenham, B., and Pfahl, D. 2008. Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review. *International Conference on Software Process*, Leipzig, Germany, May 2008. In Q. Wang, D. Pfahl, and D.M. Raffo (Eds.): ICSP 2008, LNCS 5007, 345-356. http://dl.acm.org/citation.cfm?id=1789793 Accessed Oct 11, 2012.

[3] Manifesto for Agile Software Development http://agilemanifesto.org/ Accessed Nov 11, 2012.

[4] Houston, D., and Lieu, M. 2009. Modeling a Resource-Constrained Test-And-Fix Cycle and Test Phase Duration. *International Conference on Software Process*, Vancouver, Canada, May 2009. (ICSP'09), 211-221. DOI: 10.1007/978-3-642-14347-2_19 http://rd.springer.com/chapter/10.1007%2F978-3-642-14347-2_19?LI=true# Accessed Oct 11, 2012.

[5] Buettner, D. 2009. A System Dynamics Model That Simulates a Significant Late Life Cycle Manpower Increase Phenomenon. *International Conference on Software Process*, Vancouver, Canada, May 2009. In: Q. Wang, V. Garousi, R. Madachy, and D. Pfahl (Eds.): ICSP 2009, LNCS 5543, 402-410. DOI: 10.1007/978-3-642-01680-6_36 http://rd.springer.com/chapter/10.1007%2F978-3-642-01680-6_36?LI=true# Accessed Oct 11, 2012.

[6] Houston, D., Buettner, D., and Hecht, M. 2009. Dynamic COQUALMO: Defect Profiling over Development Cycles. *International Conference on Software Process*, Vancouver, Canada, May 2009. In: Q. Wang, V. Garousi, R. Madachy, and D. Pfahl (Eds.): ICSP 2009, LNCS 5543, 161-172. DOI: 10.1007/978-3-642-01680-6_16 http://rd.springer.com/chapter/10.1007%2F978-3-642-01680-6_16?LI=true# Accessed Oct 11, 2012.

[7] Houston, D., and Hantos, P. 2010. System Dynamics and Unified Life Cycle Modeling Approaches for Exploring Concurrent Engineering Risks. *Space System Risk Management Symposium*, El Segundo, CA, April 6-8, 2010.

[8] Hantos, P. 2012. Agile Software Development in Defense Acquisition – A Mission Assurance Perspective. Aerospace Report No. ATR-2012(9010)-2. March 23, 2012.

[9] Dybå, T., and Dingsøyr, T. 2008. Empirical studies of agile software development: A systematic review. *Journal Information and Software Technology* 50, 9-10 (August 2008), 833-859.

[10] Kuppuswami, S., Vivekkanandan, K., and Rodrigues, P. 2012. A System Dynamics Simulation Model to Find the Effects of XP on Cost of Change Curve. *Extreme Programming and Agile Processes in Software Engineering*, LNCS, 2675/2003, 1014, 4-20. DOI: 10.1007/3-540-44870-5_8 http://rd.springer.com/chapter/10.1007%2F3-540-44870-5_8?LI=true# Accessed Oct 11, 2012.

[11] Kuppuswami, S., Vivekkanandan, K., Ramaswamy, P., and Rodrigues, P. 2003. The Effects of Individual XP Practices on Software Development Effort. *ACM SIGSOFT Software Engineering Notes* 28, 6 (Nov 2003) 1-6. DOI: 10.1145/966221.966239 http://dl.acm.org/citation.cfm?id=966239 Accessed Oct 11, 2012.

[12] Vivekanandan, K. 2004. *The Effects of Extreme Programming on Productivity, Cost of Change and Learning Efficiency*. Ph.D. dissertation, Pondicherry University. http://shodhganga.inflibnet.ac.in/handle/10603/1018 Accessed Oct 11, 2012.

[13] Wernick, P, and Hall, T. 2004. The Impact of Using Pair Programming on System Evolution: a Simulation-Based Study. *Proc. 20th IEEE International Conference on Software Maintenance (ICSM'04)* 422-426. DOI: 10.1109/ICSM.2004.1357828 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1357828&tag=1 Accessed Oct 11, 2012.

[14] Mišić, V., Gevaert, H., and Rennie, M. 2004. Extreme Dynamics: Modeling the Extreme Programming Software Development Process. *Proc. 2004 International Workshop on Software Process Simulation (ProSim04)* 237-242.

[15] Cao, L. 2005. *Modeling Dynamics in Agile Software Development*. Ph.D. Dissertation, Computer Information Systems, Georgia State University. http://digitalarchive.gsu.edu/cis_diss/4/ Accessed Oct 11, 2012.

[16] Cao, L., Ramesh, B., and Abdel-Hamid, T. 2010. Modeling Dynamics in Agile Software Development. *ACM Transactions on Management Information Systems* 1, 1 (December 2010) 5:1-5:26. DOI: 10.1145/1877725.1877730 http://dl.acm.org/citation.cfm?id=1877730 Accessed Oct 11, 2012

[17] Cau, A., Concas, G., Melis, M., and Turnu, I. 2005. Evaluate XP Effectiveness Using Simulation Modeling. *Extreme Programming and Agile Processes in Software Engineering*, LNCS 3556/2005, 1142-1144. DOI: 10.1007/11499053_6 http://rd.springer.com/chapter/10.1007%2F11499053_6?LI=true# Accessed Oct 11, 2012.

[18] Turnu, I., Melis, M., Cau, A., Marchesi, M., and Setzu, A. 2004. Introducing TDD on a Free Libre Open Source Software Project: a Simulation Experiment. *Proc. 2004 Workshop on Quantitative Techniques for Software Agile Process* (November 2004) 59-65. Newport Beach, California. DOI: 10.1145/1151433.1151442 http://dl.acm.org/citation.cfm?id=1151442 Accessed Oct 12, 2012.

[19] Turnu, I., Melis, M., Cau, A., Setzu, A., Concas, G., and Mannaro, K. 2006. Modeling and simulation of open source development using an agile practice. *Journal of Systems Architecture* 52, 11 (November 2006) 610-618. DOI: 10.1016/j.sysarc.2006.06.005 http://www.sciencedirect.com/science/article/pii/S1383762106000634 Accessed Oct 12, 2012.

[20] Melis, M. 2006. *A Software Process Simulation Model of Extreme Programming.* Ph.D. dissertation, School in Information Engineering University of Cagliari, Italy. http://www.diee.unica.it/DRIEI/tesi/18_melis.pdf Accessed Oct 12, 2012.

[21] Melis, M., Turnu, I., Cau, A., and Concas, G. 2006. Evaluating the impact of test-first programming and pair programming through software process simulation. *Software Process: Improvement and Practice* 11, 4 (July/August 2006). 345-360. DOI: 10.1002/spip.286 http://onlinelibrary.wiley.com/doi/10.1002/spip.286/abstract Accessed Oct 12, 2012

[22] Ur, S., Yom-Tov, E., and Wernick, P.. 2007. An Open Source Simulation Model of Software Development and Testing. *Proc. 2nd International Haifa Verification Conference. In: Hardware and Software, Verification and Testing*, LNCS 4383, 124-137. DOI: 10.1007/978-3-540-70889-6_10 http://rd.springer.com/chapter/10.1007%2F978-3-540-70889-6_10?LI=true# Accessed Oct 12, 2012.

[23] Yilmaz, L., and Phillips, J. 2006. Organization-theoretic Perspective for Simulation Modeling of Agile Software Processes. *International Workshop on Software Process Simulation* 2006. In: *Software Process Change*, LNCS 3966, 234-241. DOI: 10.1007/11754305_26 http://rd.springer.com/chapter/10.1007%2F11754305_26?LI=true# Accessed Oct 12, 2012.

[24] Phillips, J. 2006. *Team-RUP: An Agent-based Simulation of Team Behavior in Software Development Organizations*. Masters thesis, Auburn University. http://etd.auburn.edu/etd/bitstream/handle/10415/497/PHILLIPS_JARED_46.pdf?sequence=1 Accessed Oct 12, 2012

[25] Phillips, J., and Yilmaz, L. 2007. Team-RUP: Agent-based Simulation of Team Behavior in Software Development Organizations. *International Journal of Simulation and Process Modelling* 3, 3, 170-179. DOI: 10.1504/IJSPM.2007.015240 http://www.inderscience.com/info/inarticle.php?artid=15240 Accessed Oct 12, 2012.

[26] Chichakly,K. 2007. Modeling Agile Development: When is it Effective? *Proc. 25th International Conference of the System Dynamics Society*. http://www.systemdynamics.org/conferences/2007/proceed/papers/CHICH380.pdf Accessed Oct 12, 2012.

[27] Wu, M., and Yan, H. 2009. Simulation in Software Engineering with System Dynamics: A Case Study. *Journal of Software* 4, 10 (December 2009) 1127-1135. DOI: 10.4304/jsw.4.10.1127-1135 http://ojs.academypublisher.com/index.php/jsw/article/view/041011271135/1447 Accessed Oct 12, 2012.

[28] van Oorschot, K., Sengupta, K., and van Wassenhove, L. 2009. Dynamics of Agile Software Development. *Proc. 27th International Conference of the System Dynamics Society*. http://www.systemdynamics.org/conferences/2009/proceed/papers/P1264.pdf Accessed Oct 12, 2012.

[29] Yong, Y., and Zhou, B. 2009. Evaluating Extreme Programming Effect through System Dynamics Modeling. *Proc. International Conference on Computational Intelligence and Software Engineering (CiSE 2009)* 1-4. DOI: 10.1109/CISE.2009.5365556 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5365556 Accessed Oct 12, 2012.

[30] Cocco, L., Mannaro, K., Concas, G., and Marchesi, M. 2011. Simulating Kanban and Scrum vs. Waterfall with System Dynamics. *Agile Processes in Software Engineering and Extreme Programming*, LNBIP 77, 117-131. http://rd.springer.com/chapter/10.1007/978-3-642-20677-1_9?null Accessed Oct 12, 2012.

[31] Kong, X., Liu, L., and Chen, J. 2011. Modeling Agile Software Maintenance Process Using Analytical Theory of Project Investment. *Proc. International Conference on Advances in Engineering*, Procedia Engineering 24,138-142. http://www.sciencedirect.com/science/article/pii/S1877705811054683 Accessed Oct 12, 2012.

[32] Faynshteyn, L., Mišić, V., and Mišić, J. 2012. Analyzing the Cost and Benefit of Pair Programming Revisited. *Journal of Information Technology and Applications*, 2, 1 (June 2012) 14-21. http://www.jita-au.com/Public/PDF/JITA_Vol%202_Issue1.pdf Accessed Oct 12, 2012.

[33] Miranda, E. 2001. Improving Subjective Estimates Using Paired Comparisons. *IEEE Software* 18, 1 (January/February 2001) 87-91. http://doi.ieeecomputersociety.org/10.1109/52.903173

[34] Miranda, E., Bourque, P., Abran, A. 2009. Sizing User Stories Using Paired Comparisons. *Journal Information and Software Technology* 51, 9 (September 2009) 1327-1337. 10.1016/j.infsof.2009.04.003