SYSTEMS ENGINEERING
Research Center

Agile and Lean Systems Engineering: Kanban in Systems Engineering

Final Technical Report SERC-2011-TR-022

31 December 2011

Principal Investigator - Richard Turner, Stevens Institute of Technology
Co-Principal Investigator: Ray Madachy, Naval Postgraduate School

Team Members
Jo Ann Lane - University of Southern California
Dan Ingold, PhD Candidate - University ofSouthern California
Laurence Levine, PhD Candidate - Stevens Institute of Technology

Contract Number: H98230-08-D-0171

DO1, TTO2, RT 035

Report No. SERC-2011-TR-022
December 31, 2011

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
31 DEC 2011	**Final**	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Agile and Lean Systems Engineering: Kanban in Systems Engineering	**H98230-08-D-0171**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Turner /Dr. Richard	**RT 35**
	5e. TASK NUMBER **DO1 TTO2**
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Stevens Institute of Technology Naval Postgraduate School University of Southern California	**SERC-2011-TR-022**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
DASD (SE)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research project evaluates the use of on-demand (pull or kanban) scheduling approaches in systems engineering. In particular, this initial phase focuses on systems engineering where rapid response software development projects incrementally evolve capabilities of existing systems and/or systems of systems. It defines and models a kanban-based scheduling system and a services approach to systems engineering among software projects in such an environment. It then reports on simulation of their performance and those of traditional SE methods to understand if systems engineering functions are accomplished more effectively and efficiently, and whether the overall value of the systems of systems over time is increased.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**65**	

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology

This page intentionally left blank

ABSTRACT

This research project evaluates the use of on-demand (pull or kanban) scheduling approaches in systems engineering. In particular, this initial phase focuses on systems engineering where rapid response software development projects incrementally evolve capabilities of existing systems and/or systems of systems. It defines and models a kanban-based scheduling system and a services approach to systems engineering among software projects in such an environment. It then reports on simulation of their performance and those of traditional SE methods to understand if systems engineering functions are accomplished more effectively and efficiently, and whether the overall value of the systems of systems over time is increased.

ACKNOWLEGEMENTS

This work could not have been accomplished without the support of an extremely talented volunteer industry working group of experienced professionals with extensive experience in lean, kanban and systems engineering. This group included:

- David Anderson (David J. Anderson and Associates)
- Mike Burrows (David J. Anderson and Associates)
- Victor Dingus (TASC)
- Brian Gallagher (Northrop Grumman)
- Hillel Glazer (Entinex)
- Curtis Hibbs (Boeing)
- Suzette Johnson (Northrop Grumman)
- Dominic Lepore (Stevens Institute, Howe School)
- Don Reinertsen (Reinertsen & Associates)
- David Rico (Boeing)
- Garry Roedler (Lockheed Martin)
- Karl Scotland (Rally Software, UK)
- Alan Shalloway (NetObjectives)
- Neil Shirk (Lockheed Martin)
- Neil Siegel (Northrop Grumman)
- James Sutton (Jubata Group)

Thanks are also due to the members of the SERC Research Council, particularly Barry Boehm and Jon Wade, for their support.

TABLE OF CONTENTS

FIGURES AND TABLES

This page intentionally left blank

1 SUMMARY

1.1 PURPOSE OF RESEARCH

This research evaluates the use of on-demand (pull or kanban) scheduling approaches in systems engineering where rapid response software development projects incrementally evolve capabilities of existing systems and/or systems of systems. It is hypothesized that such systems could provide more effective integration and use of scarce systems engineering resources, enhance flexibility and predictability over complex master schedules, improve visibility and coordination across multiple projects, lower governance overhead, and achieve higher system-wide value earlier.

1.2 WORK ACCOMPLISHED

A general kanban-based scheduling system was defined and coupled with a service-oriented approach to systems engineering to develop an approach for integrating multiple related projects with a resource pool of systems engineers. A number of simulations have been developed to investigate whether the hypothesized benefits seemed likely to result from an in vivo implementation.

1.3 FINDINGS

In developing the simulations it became clear the complexity of the environment and the nature of both kanban scheduling and service-oriented systems engineering dictated a hybrid model with discrete-event, agent-based and continuous components. We have explored interactions between the modeling components. Current simulations make quantitative assumptions for the baseline cases and approximate well-tuned kanban processes executed by proficient practitioners and the results have been in line with expectations. However, in order to gain sufficient confidence for in vivo experimentation, we will need better project data to parameterize and calibrate the models for the sponsor's rapid response environment. Access to the current simulations is available online at http://softwareprocessdynamics.org/models/se_kanban/.

1.4 RESEARCH RESULTS

Beyond the findings as presented, the following research accomplishments have been achieved:

- Two peer-reviewed conference papers have been written; another is in process An international advisory working group has been established and has contributed to this work.
- Two peer-reviewed conference papers have been accepted; two others have been submitted
- One international conference workshop on the subject has been conducted and a

second has been accepted for conduct this spring
- Two doctoral candidates are using this work in their dissertation approaches

1.5 NEXT STEPS

In Phase 2 we will integrate the best aspects of each type of model into a demonstration toolset which can take actual or estimated data provided by an organization and indicate how using the defined kanban, service-oriented approach might improve their current process performance. The intent is then to provide that toolset to a variety of organizations with similar environments to gather additional baseline data to improve the simulations.

2 INTRODUCTION

Traditional systems engineering (SE) developed half a century ago, primarily driven by the challenges faced in the aerospace and defense industries. The environment was fairly uniform – hardware-driven, long lived, single mission. The result of this uniformity was practices that worked well in that specific context were seen as "best practices," and came to define the discipline of systems engineering. In the last few decades, system contexts have multiplied, and the speed of change in both needs and solution technologies has accelerated. This has led to an inherent loss of determinism—requirements are less tangible, more evolving, and sometimes emergent and systems are both complex and constantly adapting. The practice of systems engineering, with its roots in long-term, primarily hardware projects, has not kept pace.

Engineering principles involving agility and leanness have been adopted to address non-determinism in software systems. They use iterative and spiral concepts, require less traditional ceremony, maintain closer interaction with stakeholders, and are based on best practice, underlying theory and overarching principles. Combining agile-lean software experience with system engineering fundamentals can provide practical, principle-driven agile-lean systems engineering approaches for the design of complex or evolving hardware-software-human systems.

This research task examined one of those approaches, kanban (pull) scheduling techniques, to determine its applicability to systems and software engineering in a rapid response environment. The task developed a general kanban approach, a specific kanban-based process for supporting SE in rapid response environments, and simulated that process as well as traditional processes to determine if there were gains in effectiveness and value.

2.1 KANBAN AS A STARTING PLACE

A kanban (signal card) approach is a form of on-demand scheduling that provides a visual means of managing the flow within a process. The signal cards are created to the agreed capacity of the process and one card is associated with each piece of work. In manufacturing, work can mean the creation of a part, the integration of a part into an assembly, the completion of a particular analysis process, or whatever bounded and completeable activity you wish to track through the process. Once all of the cards have been associated, no more work in that process can begin until some piece of work is completed and the card becomes available. A common example of a simple kanban is the use of a limited number of tickets for entry into the Japanese Imperial Gardens [8]. The fundamental idea is to use visual signals to synchronize the flow of work with process capacity, limit the waste of work interruption, minimize excess inventory or delay due to shortage, prevent unnecessary rework, and provide a means of tracking work progress.

In knowledge work, the components of production are ideas and information [10, 11]. In software and systems, kanban systems have evolved into a means of smoothing flow by

balancing work with resource capability. The concept was extended to include the limiting of work in progress according to capacity. Work cannot be started until there is an available appropriate resource. In that way, it is characterized as an on-demand or "pull" system, since the work is pulled into the activity as capacity is available rather than "pushed" via a schedule.

A kanban system is a visually monitored set of activities, where each activity has its own ready queue and set of resources to add value to work units that flow through it. The fact that queues are explicit in the system allows costs of delay and other usually invisible aspects of scheduling to be front and center in decision making. Queues also provide a vast body of experience and underlying science from the queuing theory discipline. Control of the kanban system is generally maintained through *batch size*, *Work in Progress (WIP)* limits and *Classes-of-Service* (COS) definitions that prioritize work with respect to risk.

The visual representation of work is critical to kanban success, because it provides immediate understanding of the state of flow through the set of activities. This transparency makes process anomalies (both common and special cause) or resource issues easily visible, enabling the team to recognize and react immediately to resolve the issue. Flow through the kanban system is measured and tracked through statistical methods that support tuning the control parameters to improve the system. Flow measures also provide a good handle for effectiveness comparison. Because the team and management interact with the kanban board and collectively solve problems, this aspect is important in achieving continuous improvement (kaizen).

WIP is partially-completed work, equivalent to the manufacturing concept of parts inventory waiting to be processed by a production step. WIP accumulates ahead of bottlenecks unless upstream production is curtailed or the bottleneck resolved [12]. WIP in knowledge work can be roughly associated to the number of work items that have been started and not delivered. *Limiting WIP* is a concept to control flow and enhance value by specifically limiting the amount of work to be assigned to a set of resources (a WIP Limit). WIP limits accomplish several goals: they lower the context-switching overhead that impacts individuals or teams attempting to handle several simultaneous work items; they accelerate useful value by completing work in progress before starting new work; and, they provide for reasonable and sustainable resource work loads.

Using *small batch sizes* is a supporting concept to WIP. Reducing batch size limits rework and provide flexibility in scheduling and response to unforeseen change. Smaller batch sizes help stabilize the process flow and allow downstream processes to consume the batches smoothly, rather than in a start-and-stop fashion that makes inefficient use of resources. The move from "one step to glory" system initiatives to iterative, deployable increments is an example of reducing batch size. Incremental builds and ongoing, continuous integration also approximate the effect of small batch sizes.

For a different approach to describing kanban, see Mike Burrows' *Kanban in a Nutshell* (http://positiveincline.com/index.php/2010/03/kanban-in-a-nutshell/)

In the remainder of the paper we will refer to the proposed approach as a *kanban-based scheduling system (KSS)*. While not a true kanban in the manufacturing sense, the characteristics are sufficiently similar to support the name.

2.2 PREDICTED BENEFITS OF THE PROPOSED APPROACH

A workshop was held January 27-28 2010 to discuss the development of a 3-year roadmap for transforming systems engineering. A number of issues identified and discussed in that meeting are addressed by the following benefits likely to accrue from the application of this research.

2.2.1 More effective integration and use of scarce systems engineering resources

Using a KSS and applying a model of SE based on continuous activities and individually requested services is a value-based way to prioritize the use of scarce SE resources across multiple projects. The value function within the next-work selection policies can be tailored to provide efficient and effective scheduling that maximizes the value provided by the resource based on multiple, system-wide parameters. Additionally, having service requests including time vs. value parameters can help determine if the delay of other service requests fulfillment is warranted by the current service request. This is addressed further under the value function discussion.

2.2.2 Flexibility and predictability

SE activities are generally designed for pre-specifiable, deterministic (complete and traceable) requirements and schedules. There is often an overdependence on unnecessary formal ceremony and fairly rigid schedules. Using cadence rather than schedule can provide efficient SE flow with flexibility by operating with shorter planning horizons and on-demand services. We believe that the CoS concept not only handles expedite and date-certain conditions, but also supports cross-kanban synchronization. Even though the planning is dynamic and the selection of the next piece of work to do asynchronous, we believe the use of a value-based selection function, a time-cognizant service request, customized Classes of Service, and a statistically controlled cadence provide a sufficient level of predictability where necessary.

2.2.3 Visibility and coordination across multiple projects

In highly concurrent engineering, the KSS provides a means of synchronizing activities across mutually dependent teams by coordinating their activities through changing value functions (work item priority) according to the degree of data completeness and maturity (risk of change). It also provides an excellent way to show where work items are and the status of work-in-progress and queued or blocked work. The ability of teams to have a common visualization of work item status also encourages a sense of collective responsibility.

In addition, the on-demand/limited planning horizon of the KSS actually reduces the impact of long latency dependency between work items by not beginning work on items that would then languish until another work item was complete.

2.2.4 Low governance overhead

Implementing a KSS doesn't require major changes in the way work is accomplished or imply specific organizational structures like other agile methods (e.g. Scrum). Such systems can be set up in individual projects and allowed to evolve into more effective governance over time as the project and the organization as a whole understand the best way to attain value from the practices. Even the systems engineering resource scheduling can be implemented with very little organizational impact. Practitioners make most decisions using parameters set by management (e.g. WIP limits) and their own understanding of the needs. Issues are usually identifiable from walking the visible representation of the flow status and so are made clear to all who take part in the scheduling, including management. Metrics are inherent to the system, clearly identify problems, and track improvements. Most problems tend to be self-correcting.

2.2.5 Increased project and system value delivered earlier

The core rationale of most lean and agile approaches is to provide value to the customer as quickly as possible. In rapid development environments this is particularly important. By limiting WIP, more closely integrating the SE and project engineering activities, and providing both specific project and system-wide work item value determination, the KSS provides an intentional approach to achieving early value. Nevertheless, through Classes of Service, the KSS still provides for intangible or long-term investment activities to flow through the system with minimal impact on urgent activities.

3 THE KANBAN-BASED SCHEDULING SYSTEM

3.1 DEFINITION OF A KSS

In Figures 1 and 2, and Table 1, we define our concept of a KSS. We intend that this model be recursive at many levels to allow for complex implementations. While we currently believe work items and their associated parameters coupled with the visual representation of flow are sufficient, we may introduce new concepts to enable better communications and synchronization between the various interacting systems.

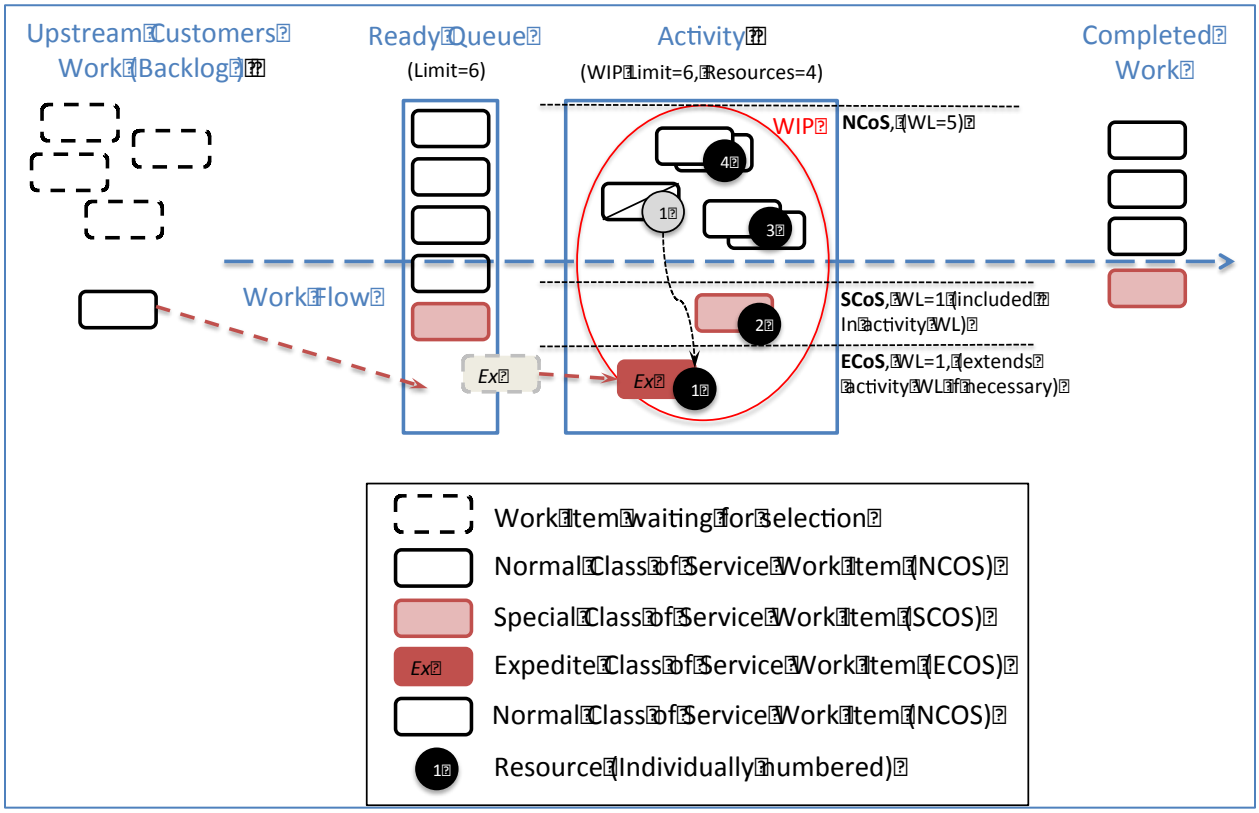


Figure 1. Kanban Scheduling System Model

Figure 1 shows the core concept of the KSS. This core concept can be thought of as a building block or even a recursive application of the fundamentals discussed in Section 2. In general, the upstream customer for the service provided is responsible for selecting the work that enters the KSS. This is usually done collaboratively with the KSS to make sure that significant dependencies, date certain events, and other special concerns are understood. As a resource becomes available, the highest value work item is executed until it is complete, and then added to the completed work. Depending on the delivery cadence, it may go directly to the downstream consumer or it may be held until the next delivery date.

A scheduling cadence provides regular meetings of the KSS team to assess the work flow and determine if resources should be moved between activities, WIP limits adjusted, or other actions taken. Often, this is a daily activity, but the actual planning horizon selected and the nature of the work items should be used to establish the most cost effective cadence. Planning horizon is based on the visibility into upcoming work and is dependent on the WIP and ready queue limits.

The illustration shows a work item with a CoS of *expedite* coming into the KSS. According to the policies established for this KSS, *expedite* is allowed to bump up the WIP limit for the activity, but the activity is itself limited to only one *expedite* CoS work item at a time. The entry of the expedited work item blocks the activity from pulling any additional work items, and causes resource #1 to suspend work on the their current work item, thus blocking it as well. In this case, the team felt that resource 1 was sufficient to accomplish the expedited work item, and that allowing the remaining resources to continue their current work items best served the KSS flow. If this turned out to be wrong, adjustments could be made immediately to resolve the imbalance.

In this illustration, the KSS consists of a single activity – and that is generally how the upstream customer would view it. However, it is easy enough to see that the activity and its associated ready queue could be subdivided into multiple linked instances. These could be linked sequentially or could represent different specializations for different types of services, each representing a full KSS. For example, there could be an initial activity that determines the relative value of a work item (its precedence given the current status of resources) and assigns it to the appropriate specialized service KSS.

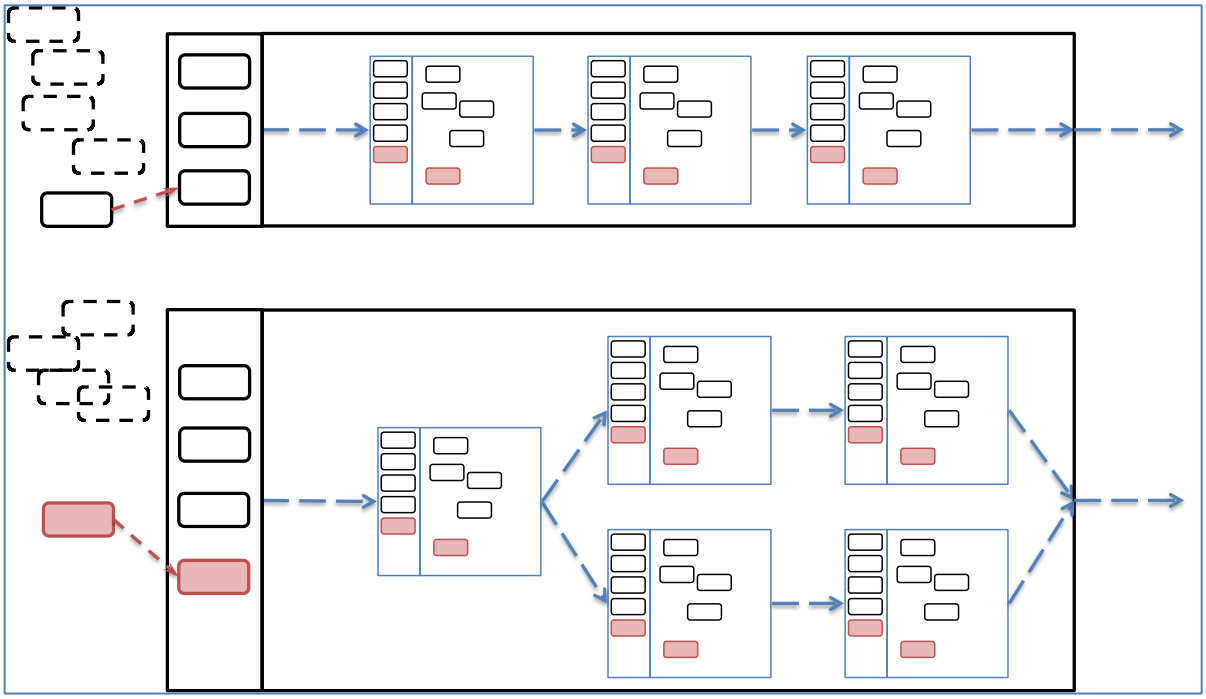


Figure 2. Kanban Scheduling System Hierarchy

Table 1. Kanban Scheduling System Definitions

Work Item	The item controlled in the kanban system. A work item has a definition, a Class of Service, and often a rough estimate of work effort required. The value of a work item is determined by a value function, and can vary over time (particularly for work items of special CoS such as expedite and time certain).
Effort Required	The approximate size of work in person-units of time. May be a negotiated function of desired quality.
Transit Time	The time measured from entrance of a particular work item into the KSS to its delivery to the customer
Backlog	A customer-prioritized queue containing upstream customer work items awaiting service by a kanban system.
Cadence (prioritization and delivery)	The rhythm of the production system. Prioritization cadence defines the planning horizon for the KSS. Delivery cadence allows bundling work items if desired by the downstream customer. Not necessarily an iteration. Kanban still allows for iterations but decouples prioritization and delivery to allow them to vary independently of cycle time according to customer desires, domain, and costs.
Activity	Value-adding work that can be determined as complete. Includes: ready queue, a set of resources, and a WIP Limit. Allows allocation of effort to complete a work item.
Ready Queue	A limited queue that holds work items awaiting processing by an activity. The items in the queue may be considered part of the Activity WIP or the queue may have a specific limit. The queue cannot be unbounded in order to maintain the kanban pull effect.
Resource	An agent for accomplishing work; may be generic or have specialized expertise. May include specific productivity. Usually associated with a specific activity, but may be shared across activities. Resources can swarm to alleviate bottlenecks or handle certain Classes of Service.
Work Item Selection Policies	Rules for selecting the next work item from the backlog or a ready queue when an activity has less work than its WIP limit; depends on both Class of Service and Value Function, and leads to specific flow behaviors.
Class of Service	Provides a variety of handling options for work items. May have a corresponding WIP limit for each activity to provide guaranteed access for work of that class of service. CoS WIP limit must be less than the activity's overall WIP limit. Examples are expedite, date-certain and normal. CoS may be disruptive (such as expedite) and is the only way to suspend work in progress.
Value Function	Estimates the current value of a work item within a CoS for use in the selection algorithm. Can be simple (null value function would produce FIFO) or a complex, multiple kanban-system, multi-factor method considering shared scarce resources and multiple cost/risk factors. The means of prioritizing work items. There may be multiple value functions that return independently established values for each hierarchical layer within the KSS For example, in SE, the overall systemic value of a work item may differ from the one that the project-level value function would return.
WIP Limit	Limit of work items allowed in progress at one time within an activity. Often initially set to twice the number of resources, but used to regulate and optimize flow and slack.
Visible Representation	A common, visual indication of work flow through the activities; Often a columnar display of activities and queues. May be manual or automated. Shows status of all work-in-progress, blocked work, WIP limits. It is a characteristic that provides transparency enabling better management. Difficult to model. Provides system wide understanding of status and value, and encourages collective responsibility for flow.
Flow Metrics	Includes cumulative flow charting and average transit time.

3.2 SYSTEMS ENGINEERING AS A SERVICE

Systems engineering has struggled with acceptance in rapid-response environments, partly because it tends to operate with a broader scope and with the assumption that a holistic view requires a deeper and fuller level of knowledge than is often available in the rapid response time frame. In rapid response environments, the time scale constrains the project scope, and detailed analysis up front is perceived as less achievable. Agile and lean assume holism comes from a learning process and is valuable even when incomplete.

The idea of using an on-demand scheduling system for systems engineering in the rapid development environment is an attempt to merge the SE flow and the software development project flow rather than simply lay SE functions on top of project activities without concern for the rapid-response constraints. Our initial model of such a system-wide KSS that includes both systems and software engineering is shown in Figure 3. We believe it will support better integration of SE into the rapid response software environment, better utilize scarce systems engineering resources, and improve the overall system-wide performance through a shared, more holistic resource allocation component.

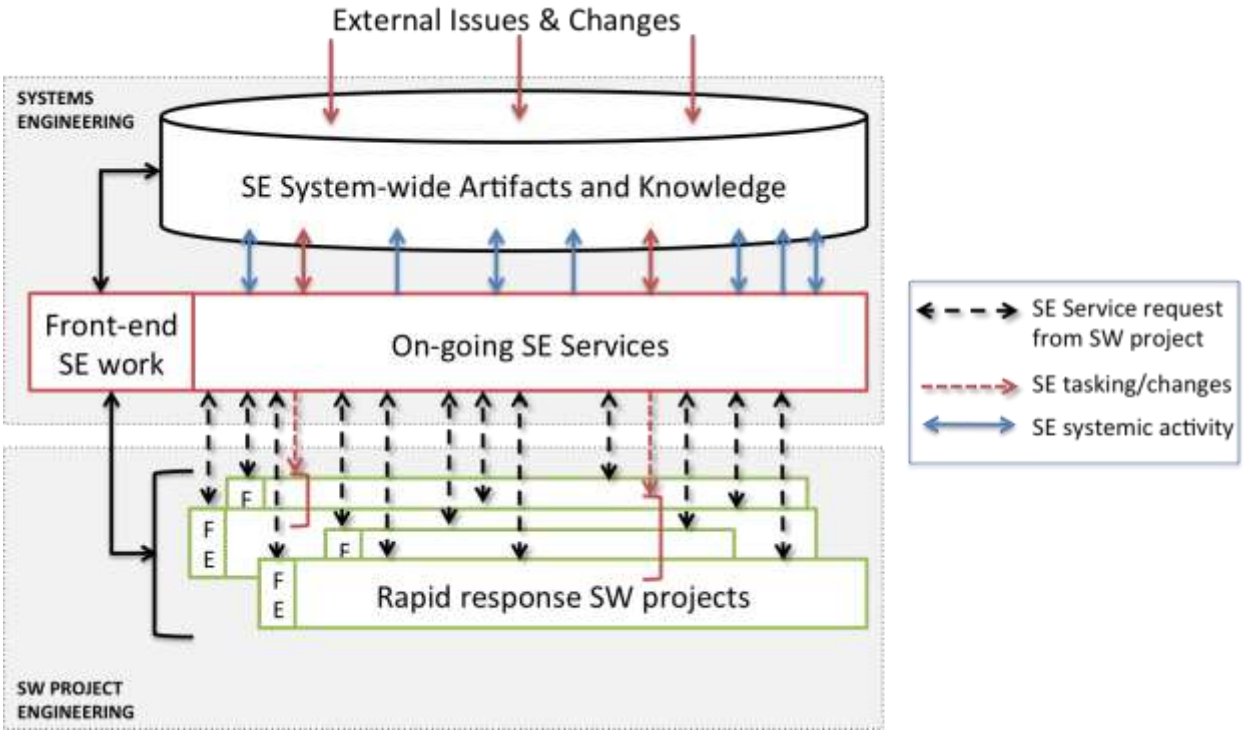


Figure 3. Overview of SE as a Service concept

In general, systems engineering is involved in three kinds of activities in rapid response environments: Up front, continuous, and requested. Up front activities are critical in greenfield projects, but are important in all systems and system of systems evolution. They include creating operational concepts, needs analysis, and architectural definitions. Continuous SE activities are ongoing, system–level activities (e.g. architecture, environmental risk management). These require not only substantial time, but also the maintenance and evolution of long-term, persistent artifacts that support development across multiple projects. Requested activities are generally specific to individual projects (e.g. trade studies, interface management), but will certainly draw on the persistent SE artifacts and knowledge.

By viewing the development and use of persistent artifacts as key components of services provided to various projects, SE can be opportunistic in applying its cross-project view and understanding of the larger environment to specific projects individually or in groups. It can also broker information between individual projects

where there may be contractual or access barriers. When a system-wide issue or external change occurs, SE can negotiate or unilaterally add or modify work items within affected projects to ensure that the broader issue is handled in an effective and compatible way. This is reminiscent of the agile management layer described in the iteration management approach in [13], and the approach envisioned can extend that concept throughout the rapid response lifecycle and across the multiple projects.

SE performs its services in parallel to those activities in the requesting project and then pushes the results to the requestor as soon as available. This is aimed at supporting the timeliness of projects, so that work can continue, even if at a higher risk of rework, unless waiting for the results is blocking all other work in the project (not a good thing).

SE services require persistent artifacts and knowledge for both requestor-specific and total system artifacts/understanding. The quality of a service could be pre-specified, specified as a parameter or input with service request, or could be negotiated as a function of typical value and time available to provide the service. In a KSS, SE services can be thought of as a single activity, although some activities, particularly those up front, are likely to be complex enough to have their own set of value adding activities and specialized resources.

The value function used to select the next request to be handled must be designed to identify the highest cost of delay among the ready work items in terms of the overall system value. This allows SE to be as effective as possible in providing its services across the enterprise. The function could be based on several parameters that are attributes of individual projects, individual requests, or system-wide activities. Possibilities include the maturity of the requesting project, lifecycle point of requesting project, criticality of the requesting project, and value/cost of delay/priority/class of service or other characteristics of the work impacted by the service requested. The details will be critical to achieve system wide benefits without impacting individual project timeliness. Only through modeling is the impact of various approaches to the value function determinable. In fact, modeling should be able to help identify the sweet spot of the amount and type of SE activity that produces the most value with the lowest impact to quality. Statistical and other measures will be needed to track the performance and improve the value function in vivo.

Table 2 is based on the US DoD Systems Engineering Guide for Systems of Systems [20]. It describes categories of services, specific characteristics, and provides initial estimates (high-medium-low) of the probability of an activity within the category occurring during the life cycle phases defined by the Rational Unified Process (RUP). A number of services can be defined in each category, and if needed, such definitions will be part of follow on research as the models are evolved. Sources for developing these services could include research into relevant standards (e.g. ISO/IEC-15288, CMMI), handbooks (e.g. NASA, OSD, INCOSE), and current investigations into Model-based Systems Engineering (MBSE) such as industry tools (e.g. RUP SE, Vitech MBSE), INCOSE and NDIA studies, and work by the Object Management Group (OMG).

It should be noted, however, that developing the concept of SE services is outside the scope of the first phase work documented in this report. The actual definitions of services will depend on the context of the projects and the development organizations. In our simulations, we have used the more general value of work effort rather than

detailing specific work item subject matter. In phase two, however, we intend to establish mechanics and templates for defining SE services and organizational constructs to support the KSS activities – particularly those that support collaborative engagement and collective responsibility for the system outcomes.

Table 2. Systems engineering service categories (adapted from [20])

Category	Description of activities within the category	Usage	Probability of service being required in phase			
			Inception	Elaboration	Construction	Transition
Translating Capability Objectives	Proxy for customer; translating needs into requirements and specifications; support for requirements management activities; enterprise and system level requirements allocation	Up front; Continuous; Requested	High	Medium	Low	Low
Understanding Systems and Relationships	View across multiple projects; Persistent memory across time and teams	Up front; Continuous; Requested	High	Medium	Medium	Medium
Assessing Performance Against Capability Objectives	Validation of TPMs or other performance requirements; typical V&V type activities; operational assessments	Up front; Continuous; Requested	Low	Medium	Medium	High
Developing and Evolving Architecture	Providing design guidance and supporting common architectural patterns across multiple projects; optimizing performance, throughput, maintenance through interoperable systems and system components	Continuous; Requested	Medium	High	Medium	Low
Monitoring and Assessing Changes	Supporting flexibility, resilience and agility; providing surveillance of the external environment and identifying issues and changes that might affect projects, the system or the enterprise (e.g. changes to COTS products, external interfaces, systems, operating environments)	Continuous; Requested	Low	Medium	High	Medium
Trade Studies And Decision Support	Supporting system-informed decision making by providing independent, competent analytical services	Up front; Requested	High	High	Medium	Low

4 MODELING AND SIMULATION OF THE KSS APPROACH

4.1 GOALS OF THE MODELS

The overall goal of the modeling component of this research task is to verify whether organizing projects as a set of cooperating kanbans (a kanban-based scheduling system, KSS) results in better project performance. Performance is measured through a value function, and better performance is defined as achieving value along one or more of the following scales, which seem most relevant to the rapid-response environment:

- Shortest-time to useful-value
- Highest-value for a given-time
- Lowest variation in transit time

The research question we seek to answer is: can value be improved through a KSS that controls the interaction of a resource-limited systems engineering team with one or more development teams via a service-oriented implementation. We hypothesize that if systems engineering produces a partial system definition (context, requirements, etc.) earlier, and releases that definition to development, the defects inherent in that less-complete definition can be resolved through coordinated KSS interactions between development and systems engineering. In this way, the total value realized by the project within its critical availability time limits is improved over the traditional up-front and separated parallel design process.

The web-based discrete-event and continuous hybrid simulation model is available at http://softwareprocessdynamics.org/models/se_kanban/. It is the source of the figures in Section 4.3. In Phase 2 this model may also incorporate the agent-based perspective.

4.2 MODELING STRATEGIES

Three approaches to modeling were considered for this research:

- System dynamics modeling
- Discrete-event modeling
- Agent-based modeling

As seen in Figure 4, each of these modeling approaches has advantages for the problem domain and level of abstraction.

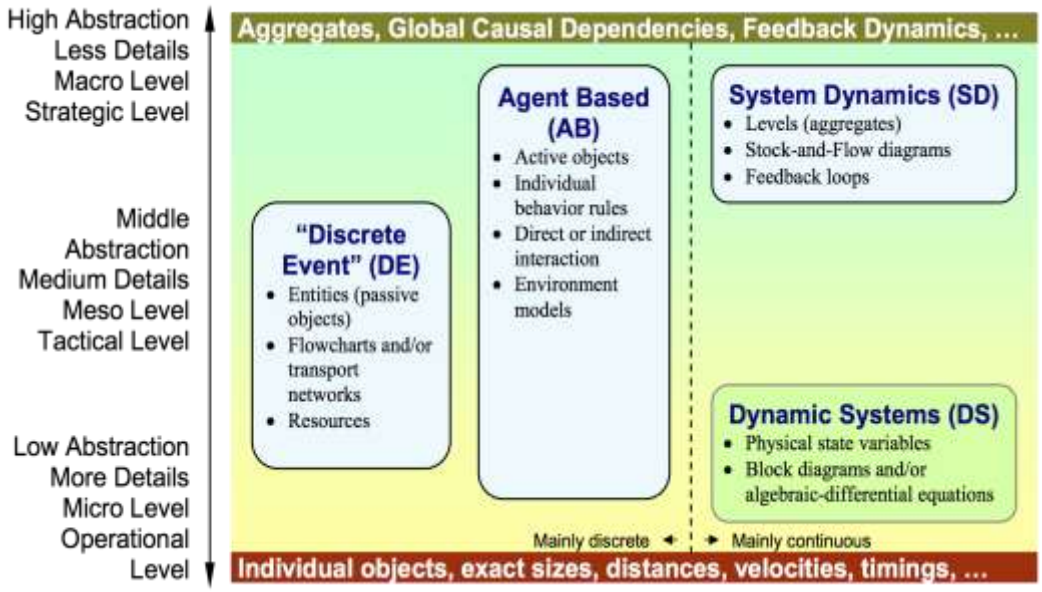


Figure 4. Modeling approach vs. abstraction level [17]

System dynamics models operate at a high-level of abstraction, and require the modeler to understand *a priori* the relationships among concepts, which are modeled as a set of interacting feedback loops [17]. They work by accumulating continuous flow quantities (representing a quantity of documents, work items, personnel, etc.) over time to create cumulative "levels" of those quantities. A given flow and its associated levels are homogeneous—that is, not divisible into discrete items—and modeling concepts of different types requires creating a separate flow for each type. In this research, the attributes of different work items—arrival time, duration, value-function, and desired quality-function—are expected to affect the overall performance of the system. The homogeneity of flows in systems dynamics models therefore seem less well-suited to simulate these types of interactions.

Discrete-event models operate at a low-level of abstraction, and consider the effect of events that occur at specific points in time by simulating the movement of discrete entities through blocks [17]. An entity (most likely representing an individual work item) is a passive construct, but can have individual characteristics that affect how the entity is processed in the simulation, for each block through which it passes. These per-entity characteristics, unlike the homogeneous flows of systems dynamics, seem better suited for modeling the attributes of the specific work items in this research. A discrete-event model is not well-suited to modify the emergent behavior of agents that act on these entities, however, and this behavior must instead be understood *a priori* and programmed into the model.

Agent-based models are similar to discrete-event models, but the entities modeled can be active objects, having attributes and performance, and active agents, having behaviors and executing work processes. While the behavior of the individual agents, and actions that can be taken by the objects, are pre-specified, system-level behavior may emerge from the interaction of agents with objects, and with other agents, that may be impossible to predict, and hence to model using the other modeling approaches. This

aspect of agent-based models seems well-suited to the research problem, since the intentional behavior of the human agents in projects is relatively simple and well-known, while the emergent systemic results of their interactions in a KSS are not. Agent-based models have the further capability of modeling beliefs and desires, which although not explored in this research, may be useful to construct more realistic behavior in the future.

Discrete event entities are needed because individual work item characteristics are critical in an actual Kanban management scheduling process. The different priorities of the work items are used for scheduling, and the WIP itself is managed as a discrete quantity. Individual performers are also mapped to work items and this aspect can be modeled with discrete attributes.

There are also important continuous parameters that drive agent behavior, including perception delays, feedback effects, schedule pressure and deadlines, motivation and other management pressures. A combined approach could provide a richer and more holistic perspective with interacting model compartments.

We recognize that in different applications any of the modeling paradigms may be more efficient. Agent-based modeling may be less efficient than system dynamics or discrete-event, harder to develop and not a good match for a given problem [17]. In this phase we have found that agent-based modeling in Brahms has been difficult requiring workarounds. For example, multiple resources working on the same work item necessitated extra logic and will probably not scale up with the modeling scenarios.

Combined hybrid modeling is often applicable [17], [19]. For example, in this phase we have used discrete events from the work item scheduling to drive continuous flows. Agent-based and/or discrete approaches could be used for event generation.

Therefore we are not limiting our modeling approach to a single view, because our needs dictate that all approaches are valuable and they can be connected. A comparison of approaches in Table 3 for systems and software processes supplementing [18]. It is updated for agent-based modeling.

Table 3. Comparison of Modeling Approaches (adapted from [18])

	System Dynamics	**Discrete Event**	**Agent-based**
Advantages	Accurately captures the effects of feedback Clear representation of the relationships between dynamic variables	CPU efficiency Attributes allow entities to vary Queues and interdependence capture resource constraints	Define bottom-up behavior of individuals Requires no knowledge of global interdependencies
Disadvantages	Sequential activities are more difficult to represent No ability to represent entities or attributes	Continuously changing variables are not modeled accurately No mechanism for representing states	Overhead for agents sharing work items

4.3 DISCRETE-EVENT AND CONTINUOUS MODELS AND TOOL

The discrete-event and continuous model is implemented in a web-based tool. It is parameterized for users to input the number of tasks[1], effort per task (deterministic or probabilistic), WIP limit, staffing level and value parameters for the tasks.

It models the Kanban WIP limit for a variable staff size with non-linear productivity. The non-linearity is due to context-switching losses when resources are split across multiple tasks. It contains two levels of value for the project and organization:

- Project Value – Value of the task towards fulfilling the project objectives (0-10).
- SE Value – Value of the task for the systems engineering enterprise at the organizational level. (0-10).

Continuous flows for tasks and value accumulation are driven by the discrete events. The corresponding rates are pulsed at the event times for task completion and value attainment. The aggregate accumulations are used for continuous quantities such as schedule pressure due to do progress gaps. The continuous parameters are in turn available to the agent-based model compartment for simulating individual agent behaviors (e.g. peoples' delayed perceptions of trends and their reactions).

Sample runs of the DE simulation are shown in Figures 5-8. In Figure 5-7 the graphs are Gantt charts with tasks down the page and time running horizontally. Visualizing a vertical line on the Gantt chart and then counting the intersected tasks can determine the current WIP size at any time. The number of tasks concurrently active cannot exceed the WIP limit.

For each run a normal distribution is used to generate task effort, and the duration is calculated using the available staff and WIP size. These figures represent a baseline case of a nominal 90 day project.

Figure 6 shows a case where 10% of the software tasks require rework. In subsequent models the percent of rework will be directly impacted by systems engineering.

A multiple project scenario at the enterprise level is shown in Figure 7. The top tasks are systems engineering service tasks that support the three software projects (red, blue and green) below it. The initial tasks numbered 1.0, 2.0 and 3.0 are project initiation tasks. The other tasks are in continuous support of software engineering who has "kicked back" some tasks requiring more work. These tasks are numbered the same in both swim lanes (e.g. Task 2.2, Task 3.2, etc.)

The diagram also displays project priorities among the projects competing for systems engineering support. In this scenario Project 1 in red has higher priority. This is why Task 3.3 in Project 3 has to wait for the Project 1 tasks in red to complete first.

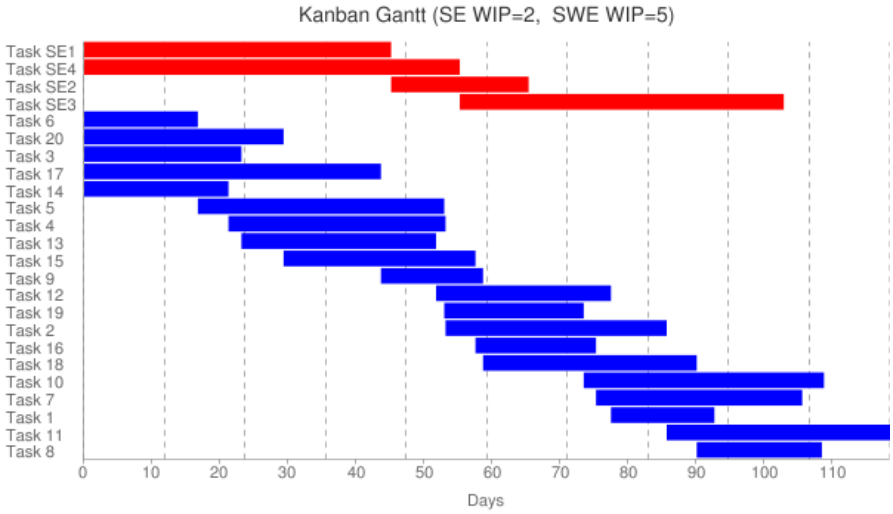
[1] In the discussion of discrete and continuous simulation, "tasks" represent KSS work items.

Systems Engineering Kanban Process Simulation

# SE Tasks (1-10)	4	# SE People (1-5)	1	SE WIP Limit (1-10)	2
# Software Tasks (5-40)	20	# Software People (1-10)	5	Software WIP Limit (1-10)	5
Monte Carlo Simulation	Off				

Run

Results
Effort = 4267.2 Person-Hours
Schedule = 118.4 Days
Value = 111



Task	Project Value	SE Value	Effort (Person-Hours)	Resource(s)	Duration	Start	Finish	Cumulative Value
6	10	9	133.5	1 SWE	16.7	0	16.7	10
14	7	1	169.6	1 SWE	21.2	0	21.2	17
3	8	1	184.9	1 SWE	23.1	0	23.1	25
20	9	2	234.1	1 SWE	29.3	0	29.3	34
17	7	3	348.9	1 SWE	43.6	0	43.6	41
13	7	8	228.8	1 SWE	28.6	23.1	51.7	48
5	7	5	289.3	1 SWE	36.2	16.7	52.9	55
4	7	6	255.1	1 SWE	31.9	21.2	53.1	62
15	6	10	225.5	1 SWE	28.2	29.3	57.5	68
9	6	3	119.8	1 SWE	15	43.6	58.6	74
19	5	7	163.9	1 SWE	20.5	52.9	73.4	79
16	5	4	141.6	1 SWE	17.7	57.5	75.2	84
12	5	2	205.7	1 SWE	25.7	51.7	77.4	89
2	5	10	260	1 SWE	32.5	53.1	85.6	94
18	4	5	250.8	1 SWE	31.4	58.6	90	98
1	3	1	121.6	1 SWE	15.2	77.4	92.6	101
7	3	5	242.1	1 SWE	30.3	75.2	105.5	104
8	2	6	147.1	1 SWE	18.4	90	108.4	106
10	3	2	282.5	1 SWE	35.3	73.4	108.7	109
11	2	10	262.4	1 SWE	32.8	85.6	118.4	111

Figure 5. Example Results

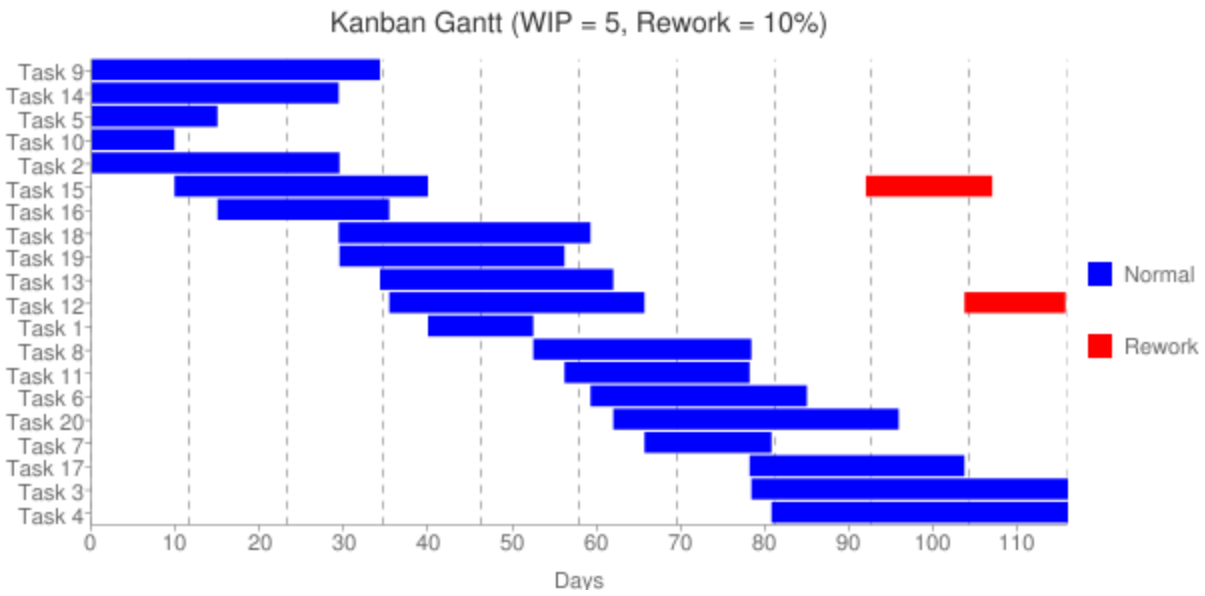


Figure 6. Example Project Gantt with Rework

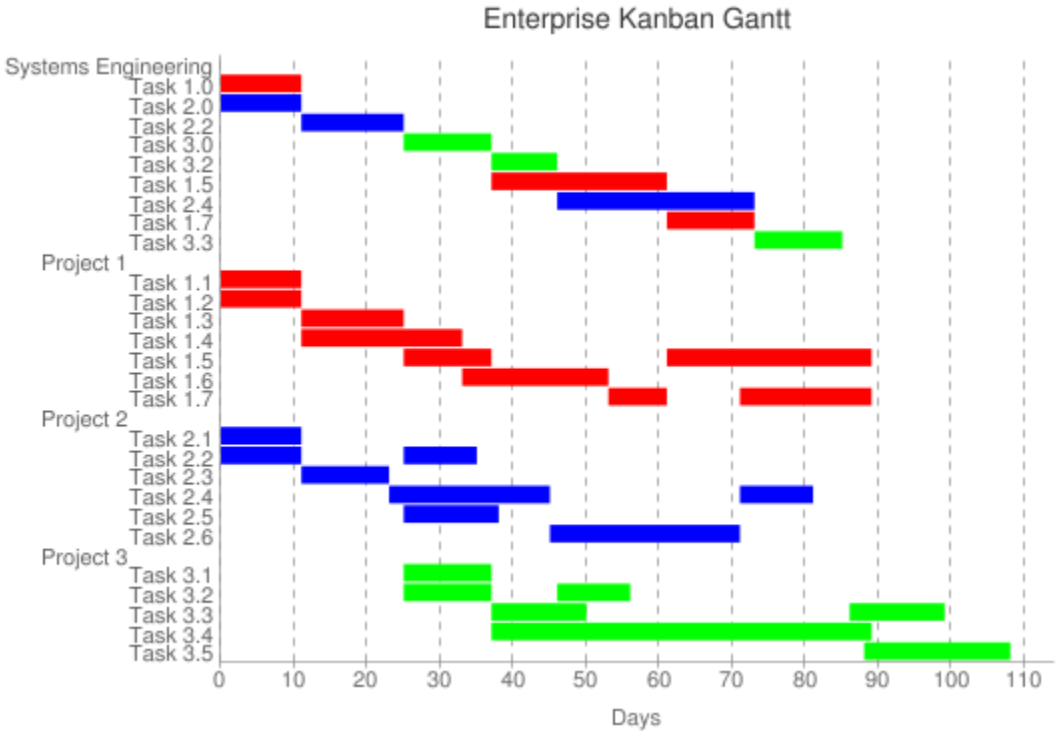


Figure 7. Example Enterprise Project Gantt with SE Services

Monte Carlo simulation can be invoked for multiple runs. Figure 8 shows sample results with output probability distributions for the four primary indicators. This capability will be enhanced in Phase 2 including normalizing the value outputs.

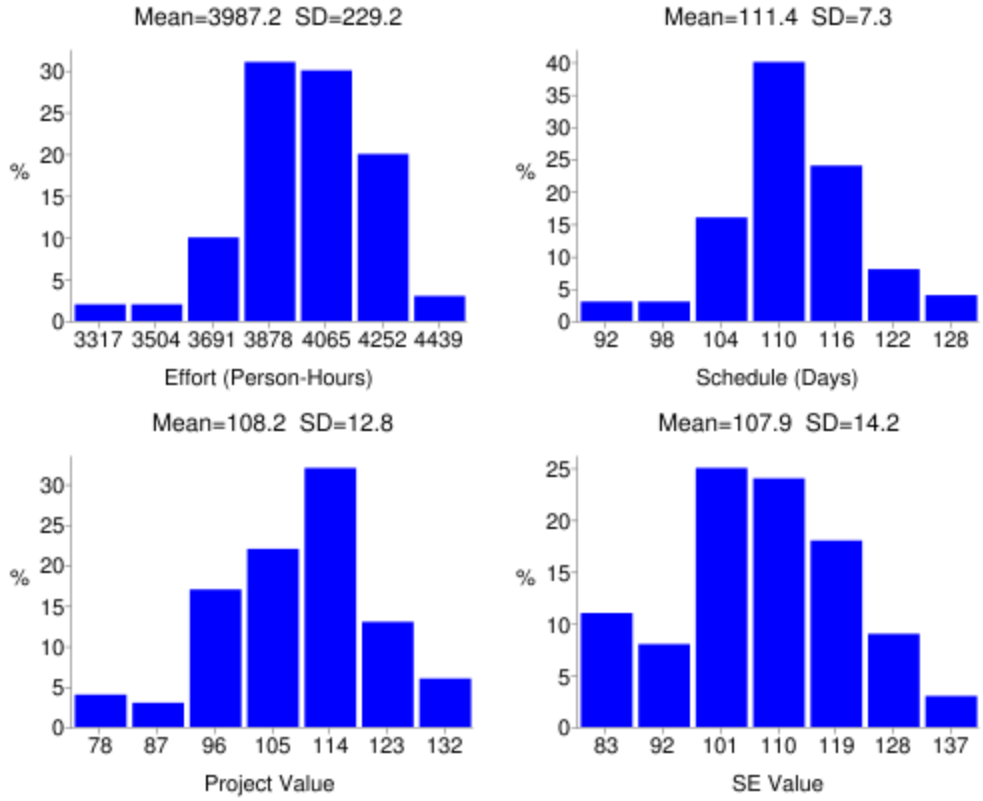


Figure 8. Example Monte Carlo Results

4.4 AGENT-BASED MODELS AND TOOLS

4.4.1 Tool selection

Two agent-based modeling tools were examined for use in this research: the Recursive Porous Agent Simulation Toolkit (Repast), originally developed by the University of Chicago; and Brahms, developed for NASA Ames Research Center. Repast is an open-source toolkit that researchers can use to develop agent-based models (ABMs) in Java, Python, and many other languages. Brahms is a Java-based proprietary tool, licensed at no charge for academic use, that provides an integrated framework within which ABMs are developed. Other tools considered early in the selection process include MASON and Swarm.

Both Repast and Brahms have been used for modeling social behavior networks of autonomous individuals. Repast appears to offer a more powerful modeling framework, but is a low-level toolkit that requires substantial programming to produce a functioning model. While Repast also has a graphical interface for constructing simple models, it did not appear suitable to construct the modeling elements necessary for this research. Brahms is oriented specifically toward modeling work processes, and is used within NASA for modeling spaceship crew tasking such as extra-vehicular activities (EVAs).

Brahms uses a belief-desire-intention (BDI) architecture and is programmed using a relatively simple, production-rule like syntax.

Both tools exhibit steep learning curves. Repast requires the modeler to understand the capabilities provided by an extensive application programming interface (API), and then to program the model directly in Java using this API. It does not appear to offer high-level facilities to communicate among agents, which must instead be explicitly programmed. Brahms operates at a higher level of abstraction, providing a direct means of representing world facts and agent beliefs, a rule-based technique for specifying the activities that arise from given fact/belief states, and built-in functions for communicating facts and beliefs among objects and agents. It also provides a facility, not used in the present research, to augment a model with capabilities programmed directly in Java.

Due to the short timeframe of this research task, the work process-oriented Brahms was considered the lower risk approach of the two. Once the model design is fully established and preliminary results are obtained, the additional analysis tools that Repast provides may make it worthwhile to convert the model in subsequent research.

4.4.2 Agent-based model design

Similar to the discrete-event simulation strategy described in (Anderson et al. 2011), the elements of the agent-based model include the concepts:

- Kanban Scheduling System (Kanban)
- Ready queue (The agent-based model uses the *ActivityQueue* object to hold *WorkItems* in a WIP-limited *Activity,* thus modeling the *Ready Queue*. A *ReleaseQueue* models the next activity's ReadyQueue or delivery downstream)
- Activities (In the agent-based model, an *Activity* object associates *Resources* with a particular project activity, and holds the *Queue* of in-process *WorkItems*.)
- Resources
- Work Items
- Customer
- Work-in-process (WIP) limits

4.4.2.1 Model workflow

Figure 9 diagrams the relationship of these concepts within the agent-based model for the KSS. The model is composed of one or more KSSs, each of which represents a project or, as will be seen, a pan-project team. Each KSS is composed of a ready queue and one or more serialized activities. Resources work within an activity, pulling completed work items from the next upstream activity (or incomplete items from the backlog, if the resource is in the first activity of a kanban), and taking some amount of time to complete each work item. The release queue pulls completed work items from the last activity of a kanban, at which point the work item is considered fully complete. The customer is the source of all work items that enter the system, which are pushed onto the backlog of one of the kanbans for processing.

Resources are the human agents whose actions take incomplete work items and transform them, with more or less fidelity and taking varying amounts of time, into

completed work items. The activity within which each resource works is constrained to a maximum work-in-process (WIP) limit, and at any point in time each activity contains no more than the WIP-limited number of work items, queued or in-process. Within each activity, some work items are queued awaiting the next available resource, and some are being processed. Work items are assigned an estimated duration and a value function at creation, and move through the system by being pulled from upstream activities into the next downstream activity, or the release queue.

This modeling approach offers additional flexibility over the model employed by [Anderson et al.]. The simplest system can be modeled with a single-activity kanban, with its backlog and release queues. More complex models can have multiple kanbans, each with multiple activities, where the release queue of the upstream kanban feeds the backlog queue of downstream kanbans. This flexibility allows the model to show how the interaction of multiple kanbans might affect project performance.

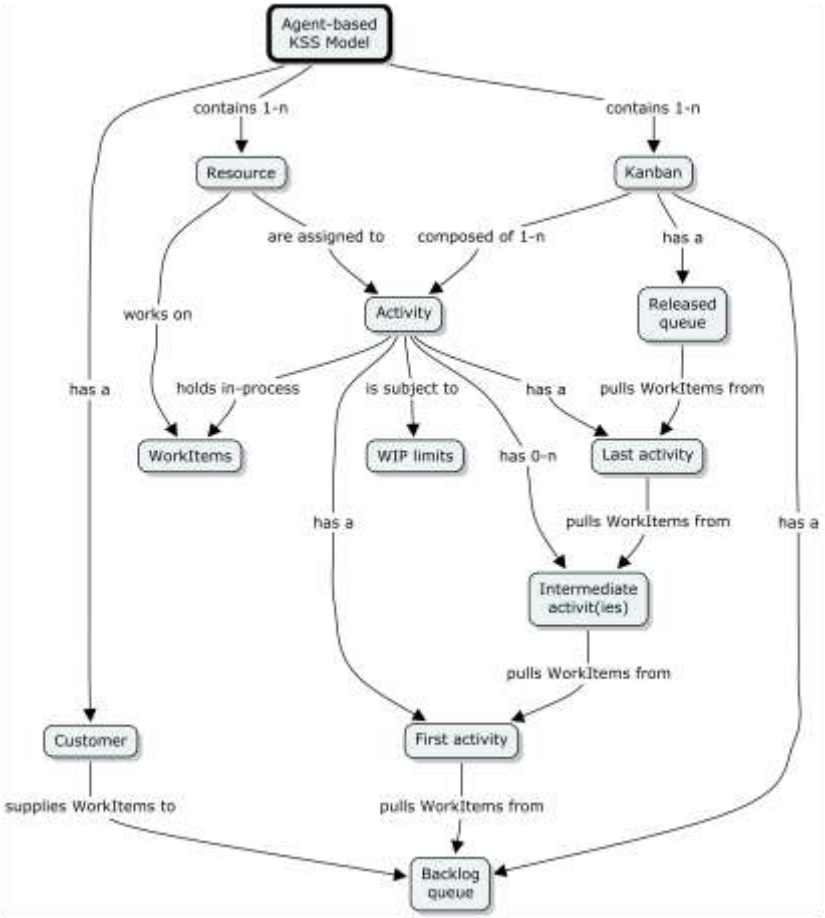


Figure 9. Agent-based model of kanban-based scheduling system

4.4.2.2 Development-SE feedback

The high-level flow of information through the KSS is presented in Figure 10. The customer is the source of high-level requirements inserted into the workflow by pushing

them to the backlog of the systems engineering kanban. Systems engineering elaborates each requirements into multiple lower-level work items, assigns a value function to each, and pushes the work items into the backlog of one or more development kanbans. The resources assigned to the development kanbans select the next work item based on its value function, and take some amount of time to complete it. Once complete, the work item is pulled by the next downstream activity, which is not described in this diagram.

We assume that, due to the time constraints of the rapid-response environment, systems engineering creates work items and releases them to development even though their design might be incomplete. This early release is necessary to avoid the large delay that would be inherent in performing a "big design up front" (BDUF), and enables development to proceed in parallel with systems engineering. We further assume that this partially-complete design leads to defects that might have been avoided or lessened in BDUF, and that these defects are detected later in the development (or some downstream) process. Such defects are then fed back as a service request, tagged with the time-criticality of the request, for systems engineering to resolve. The time-criticality informs systems engineering how quickly the request must be resolved.

Systems engineering resolves service requests with some defect rate that is proportional to the time criticality—that is, with some probability, requests that must be serviced in a shorter period will have more defects. Systems engineering completes the feedback loop by pushing a work item that results from processing the service request, with its potential additional defects, to the development kanban. The cycle may repeat if further defects are detected during development of the completed request.

The initial models have evolved by simulating more complex behavior of the resource agents, adding Brahms *thoughtframes* for the agent to "realize" it needs work, or has completed the work it has, and *workframes* for the agent to "examine" the *activity* queue and the *work items* it finds there. The queuing behavior of *activities* has been separated out as its own object and generalized, allowing its use in other contexts. A log-normal random distribution algorithm has been added, so that the actual performance time of a particular *work item* varies randomly around the estimated duration, favoring the performance taking longer than expected through the "long-tail" distribution.

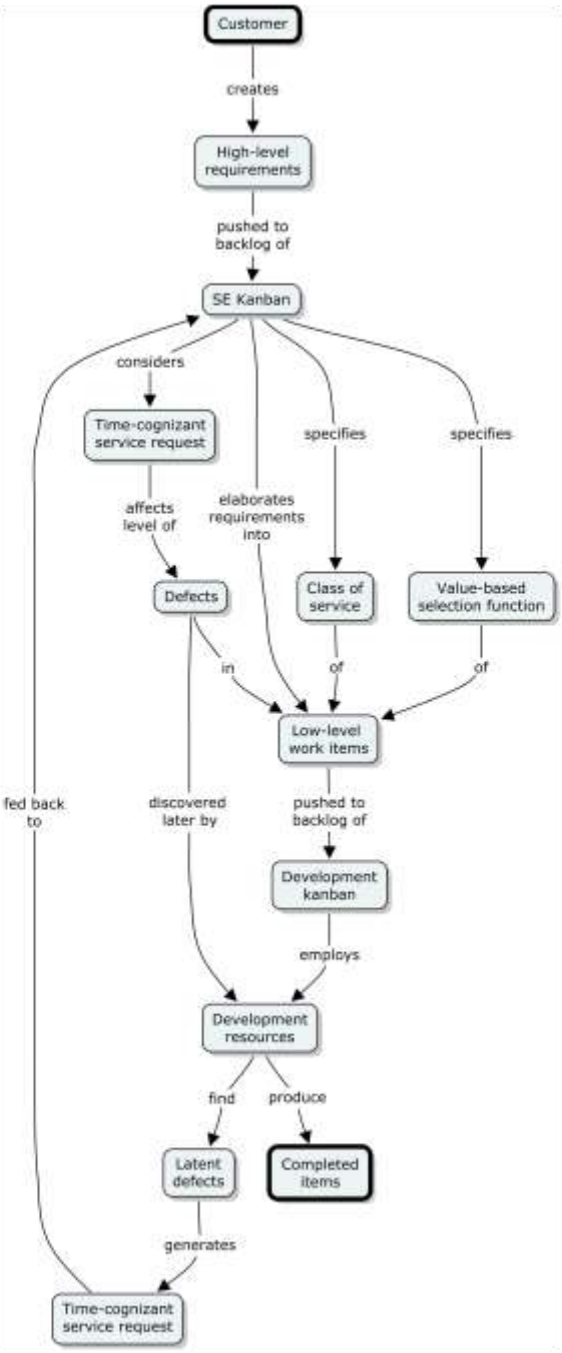


Figure 10. Information flow through KSS

Figure 11 illustrates an excerpt from the Brahms agent viewer screen, showing the results of a short example simulation run. In this example, ten WorkItems have been produced by the Customer, and pushed into the Backlog of the SensorDevelopmentProject kanban. The Development activity of that kanban detects that it is below its WIP limit, and pulls an item from the backlog into its waiting queue. Once there, a Developer resource detects the WorkItem is present, and operates on it for

a period of time. Once the WorkItem is complete, another slot opens in the activity, which again pulls the next WorkItem from the Backlog, which again the Developer detects and works on. This cycle repeats until the Backlog is empty and all WorkItems are complete.

Figure 11. Example from KSS model run

The Brahms source code for this model can be seen in Appendix B. This source code shows the attributes, relations, activities, and workframes of the inanimate objects and intentional agents comprising the model. In Brahms, objects detect the presence of facts in the world, and take action based on these facts. Agents have beliefs about these world-facts, which they in turn act upon. Both objects and agents specify their actions using a production rule-like syntax, which is similar to the syntax used in many rule-based expert systems.

4.5 MODELING SCENARIOS

The modeling scenarios chosen assume that there are multiple projects that are made up of sequential 90-day spins. Each spin produces an intrinsic value that is some fraction of that determined for the project at its inception. Significant defects, schedule slips, changes to the external environment, conflicts with other projects are factors that may reduce the value. Lower than expected defects, schedule advances, adaptation to other parts of the system, changes to the external environment are factors that could increase this value.

We are assuming development teams of 8-10 people and an SE team of similar size. The projects each have around 200 assignable tasks that average about 24 hours of effort to complete. In every case, we have assigned approximately 15% of the work to SE tasks,

with about 10% performed early in the spin (design/architectural/interface work) and 5% performed late (verification and validation work). This is in line with traditional rules of thumb for projects of a similar size.

Randomness has been added so that task values and expected durations are distributed normally and task actual durations distributed skewed to the plus side (generally tasks take longer than estimated). Additionally, extra tasks may be added that represent unplanned rework due to requirements, scope, political or technical changes within the environment. There have been some simplifications applied to the model to meet the time constraints of the project:

- All resources have the same skills and effectiveness/productivity
- A resource can only work on one task at a time
- Only one resource can work on a task at a time

It should be noted that these simplifications prevented us from modeling all the benefits of WIP limits and small batches. However, this will be addressed in the next phase of our research.

4.5.1 Scenario 1: Common

Reduced SE involvement up-front, some involvement through project execution as change traffic, heavy involvement in back-end.

4.5.1.1 Rationale

This is the approach used by projects where SE is not well integrated. Although SE may have developed (or understands an existing) system-level design, the development projects "go it alone," either following their own understanding of the design, or allowing a design to emerge as development proceeds. SE issues new requirements, or inserts change requests during project execution, as deviation from the desired system-level design is detected. As finished work items emerge from the project, their incidence of defects is higher due to lack of a coordinated design, which causes rework.

4.5.1.2 Description and modeling considerations

This scenario will be modeled using the existing model constructs, but using a non-WIP limited activity. The backlog will be pre-populated with a set of work items representing the requirements as understood, which will be processed by Development resources in random or FIFO order. If possible given model constraints, resources will switch among work items before completing them to simulate context-switching losses. SE will add new work items to the Development backlog throughout project execution, but at a low rate of insertion, to simulate change traffic. Completed work items will, with some probability, be considered defective, and be fed back to the backlog for re-processing, or their value will be decreased.

4.5.2 Scenario 2: Traditional SE

Traditional up-front/back-end SE, with Big Design Up Front (BDUF) delaying the start of development, which results in lower change traffic and defect incidence

4.5.2.1 Rationale

This is a traditional approach to SE, which takes time to perform trade studies and create a holistic design, but which also delays the start of development. The incidence of change traffic and defects should be lower. It is possible that in spite of the lower defect rate, however, lower total value may be delivered than Scenario 1, an effect that may be exacerbated in short-cycle development. It will be interesting to see how modifying the total project time, percent of time spent doing BDUF, and the change traffic and defect rate affect the project value realized by this scenario with respect to Scenario 1. (This scenario design resembles Architected Agile.)

4.5.2.2 Description and modeling considerations

This scenario will be modeled very similarly to Scenario 1, but shortening the amount of development time by the percent of SE work done up-front, and lowering the rates of new requirement- and defect-insertion. To model value-based SE, each work item may have a value assigned to it, which Development resources could use to prioritize work, perhaps with more or less fidelity. To model a KSS with this scenario, WIP limits could be set on activities.

4.5.3 Scenario 3: KSS

Incremental SE, with some design up-front and design continuing throughout development, interacting with projects using service-oriented model

4.5.3.1 Rationale

This is the approach envisioned in this KSS research: buying additional development time at the cost of less thorough up-front SE, and providing a mechanism for resolving design issues through a service interface between development and SE. In this scenario, SE will create new requirements at a relative high rate throughout project execution, simulating their emergence as design activities mature. A higher defect insertion rate than Scenario 2 will be used, but allowing an opportunity for SE to resolve defects, through requests submitted to SE by Development. It is hypothesized that this approach will yield both more timely performance and lower defect rates, resulting in higher total project value.

4.5.3.2 Description and modeling considerations

This scenario enlarges the previous scenarios by adding a separate KSS for SE, with its own *backlog*, *activities*, *resources*, and *work items*, to the KSS already established for Development. SE and Development interact through *work items* exchanged between their respective KSS's. SE undertakes both continuous activities—creating new *work*

items for Development; and requested activities—responding to *work item* requests inserted in its *backlog* by Development. Processing of a *work item* by Development will continue while SE is processing a request about the *work item*, with a higher defect rate resulting if SE does not respond quickly enough.

4.5.4 Scenario execution

The alternative scenarios as described are continuing to be created and run. To improve the comparability of the scenarios, as many variables as possible are kept constant between runs. The simulations allow seeding of the random number generator with a constant value, so that the sequence of numbers, while still pseudo-random, is repeatable in successive runs. This allows us to create work items with the same durations in each scenario, to reduce the likelihood of a different distribution of task estimated and actual durations affecting comparability.

The primary goal of the next phase is to evolve the simulations and run more carefully constructed experiments for the scenarios already defined. We will also begin to validate the simulation results against our hypotheses using carefully created and if possible, actual data.

5 RESEARCH OUTCOMES AND NEXT STEPS

5.1 SUMMARY

This research has developed the fundamental definitions and an initial set of simulation tools to evaluate a new approach to managing systems engineering where rapid response software development projects incrementally evolve capabilities of existing systems and/or systems of systems. It defines and models a kanban-based scheduling system and a services approach to systems engineering among software projects in such an environment.

Beyond the findings as presented, the following research accomplishments have been achieved:

- An international advisory working group has been established and has contributed to this work.
- Two peer-reviewed conference papers have been accepted; two others have been submitted
- One international conference workshop on the subject has been conducted and a second has been accepted for conduct this spring
- Two doctoral candidates are using this work in their dissertation approaches

5.2 NEXT STEPS FOR FURTHER RESEARCH

5.2.1 Phase 2: March-September 2012. Complete and validate simulation demonstration toolkit

Expand the existing work through adding team-related components and parametric representations for the key simulation variables. Establish a demonstration tool that can be used to show how the approach works, how the various parameters interact (e.g. values, WIP limits, work mix), and how existing teams may be modeled. This would be particularly valuable in showing executives, management and practitioners how workload, staffing, and priority decisions impact overall value produced and SE effectiveness. It also provides a basic toolkit to support ongoing experimentation and the impact on system-wide value of various alternatives. The work will complete the following deliverables/capabilities.

1. Expanded KSS Definition including:
    - Completed set of SE Services including value, quality and effort parameters/functions; templates and mechanisms to define SE services

- Specific representation of sponsor teams including multi-level SE authorities; variation of SE KSS in terms of authority levels, service needs and resource availability
- Begin to identify and incorporate initial behavioral (cognitive-affective) aspects of agents based on work by Te'eni, Weick&Roberts, Faraj&Sproull, Majchrzak and others

2. Expanded KSS Simulation including:
   - SE Services
   - Parametric Value functions
   - Parametric Quality functions
   - Parametric Effort functions
   - Defined scenarios for specific experiments
   - Include feedback from the sponsor on previous simulation
3. Initial demonstration/visualization capability
   - Demonstrates KSS concept and illustrates the key points
   - Uses data from simulation to graphically present results of various scenarios

5.2.1.1 Phase 3: October-September 2012. Apply the toolkit to multiple real environments

Complete the simulation by including behavioral and team-interaction components and the ability to run statistical experiments to identify the sensitivity of various parameters in determining outcomes. This is particularly important given the impact of team dynamics and interacting belief systems on SE process outcomes and for introducing change into organization scheduling and workflow processes. Refine and validate the tool and concept through working with real projects in environments within or similar to the sponsor's. The completed and validated simulation toolkit can lead to:

- Better understanding of the value of various SE services
- Better integration of SE and software engineering through the services concept
- Clarity in the value of SE as a knowledge broker and analysis service in brownfield evolution environments

The work will complete the following deliverables/capabilities.

1. Agent-based Simulation including:
   - Behavioral (cognitive-affective) aspects of teams
   - Bounded-rationality (team members do not always choose the best course of action).
   - Ability to run Monte Carlo-type experiments with random or parametric variation of scenarios

2. Demonstration/Analysis capability
   - Ability to define teams to match environment
   - Ability to select from various work streams
3. Validation of tool and concept through modeling actual environments. Some possible targets include:
   - Maxwell AFB LeMay Center
   - Defense industry projects (e.g. LMCO, NGC)

This page intentionally left blank

6 APPENDICES

6.1 APPENDIX A: REFERENCES

1. NDIA-National Defense Industrial Association (2010). Top Systems Engineering Issues In US Defense Industry. Systems Engineering Division Task Group Report. http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/Top%20SE%20Issues%202010%20Report%20v11%20FINAL.pdf. September, 2010.
2. Turner, Richard, Shull F., et al (2009a) "Evaluation of Systems Engineering Methods, Processes and Tools on Department of Defense and Intelligence Community Programs: Phase 1 Final Technical Report," Systems Engineering Research Center, SERC-2009-TR002, September 2009.
3. Turner, Richard, Shull F., et al (2009b) "Evaluation of Systems Engineering Methods, Processes and Tools on Department of Defense and Intelligence Community Programs: Phase 2 Final Technical Report," Systems Engineering Research Center, SERC-2010-TR004, December 2009.
4. Turner, Richard and Wade, J. (2011). "Lean Systems Engineering within System Design Activities," Proceedings of the 3rd Lean System and Software Conference, May 2-6, 2011, Los Angeles, CA.
5. Boehm, Barry and Turner, Richard (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison Wesley.
6. Larman C. and Vodde, B. (2009). *Scaling Lean & Agile Development*. Boston, MA: Addison Wesley.
7. Poppendiek, Mary. (2007). *Implementing Lean Software Development*. Boston, MA: Addison Wesley.
8. Anderson, David. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Sequim, WA: Blue Hole Press
9. Reinertsen, Donald G. (2010). *The Principles of Product Development Flow*. Redondo Beach, CA: Celeritas Publishing.
10. Poppendieck, Mary, and Tom Poppendieck. (2003). *Lean Software Development: An Agile Toolkit*. The Agile Software Development Series. Boston: Addison-Wesley.
11. Morgan, James M, and Jeffrey K Liker. (2006). *The Toyota Product Development System: Integrating People, Process, and Technology*. New York: Productivity Press.

12. Goldratt, Eliyahu M., and Jeff Cox. (2004). *The Goal: a Process of Ongoing Improvement*. Great Barrington, MA: North River..
13. Boehm, B. (2009). Applying the Incremental Commitment Model to Brownfield Systems Development, Proceedings, CSER 2009, April 2009.
14. Heath, B. et al. (2009). A survey of agent-based modeling practices (January 1998 to July 2008). Journal of Artificial Societies and Social Simulation. 12:4 2009.
15. Anderson et al. (2011). "Studying Lean-Kanban Approach Using Software Process Simulation." A. Sillitti et al. (Eds.): Agile Processes in Software Engineering and Extreme Programming, Part 1, Lecture Notes in Business Information Processing, Volume 77, Pages 12-26 2011.
16. Boehm, Barry, Ricardo Valerdi, and Eric Honour (2008). "The ROI of systems engineering: Some quantitative results for software- intensive systems." Systems Engineering 11 (3) (September 1): 221-234. doi:10.1002/sys.20096.
17. Borshchev, A., and A. Filippov (2004). "From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools." In Proceedings of the 22nd International Conference of the System Dynamics Society, 25–29.
18. M. Kellner, R. Madachy and D. Raffo (1999) *Software Process Simulation Modeling: Why? What? How?,* Journal of Systems and Software, Spring 1999.
19. R. Madachy (2008). *Software Process Dynamics*, Wiley-IEEE Press, Hoboken, NJ.
20. Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering (2008). *Systems Engineering Guide for Systems of Systems, Version 1.0*. Washington, DC: ODUSD(A&T)SSE, 2008.

APPENDIX B – SOFTWARE CODE

6.2 APPENDIX B – SOFTWARE CODE

6.2.1 Agent-based Simulation (Brahms)

6.2.1.1 Activity

```
class Activity extends BaseClass, Queue
{
  resource: true;

  attributes:
    // public boolean  hadWorkItemPulled;

  relations:
    public Kanban   isInKanban;
    public Activity hasUpstreamActivity;
    public Queue    hasDoneQueue;
    public Queue    hasBacklog;

  initial_beliefs:

  initial_facts:
    ( current.isFinished = false );

    // ( current isInKanban unknown );
    // ( current hasUpstreamActivity unknown );

  activities:
    /* -------------------------------------------------------------------
     *                                                  GET: pullWorkItem
     * ------------------------------------------------------------------- */

    get pullWorkItem( Activity activity, WorkItem workItem )
    {
      max_duration: 10;
      items: workItem;
      source: activity;
    }

    /* -------------------------------------------------------------------
     *                                        GET: pullBacklogWorkItem
     * ------------------------------------------------------------------- */

    get pullBacklogWorkItem( Queue backlog, WorkItem workItem )
    {
      max_duration: 10;
      items: workItem;
      source: backlog;
    }

  workframes:

    /* -------------------------------------------------------------------
     *                                              WFR: DoPullWorkItem
```

```
     * ---------------------------------------------------------------- */

/*
    workframe DoPullWorkItem
    {
      repeat: true;

      variables:
        forone ( Activity )   activity;
        forone ( WorkItem )   workItem;

      // This activity can accept another work item, and the upstream
      // activity has a work item that is complete.

      when ( knownval( current.canAcceptNewItem = true ) and
             knownval( current hasUpstreamActivity activity ) and
             knownval( activity contains workItem ) and
             knownval( workItem.isComplete = true ) )
      do
      {
        pullWorkItem( activity, workItem );

        conclude( workItem.isComplete = false, { fc:100, bc:100 } );
      }
    }
*/

    /* ----------------------------------------------------------------
     *                                          WFR: DoHadWorkItemPulled
     * ---------------------------------------------------------------- */

/*
    workframe DoHadWorkItemPulled
    {
      repeat: true;

      variables:
        forone ( WorkItem ) workItem;

      // Some activity has pulled a work item from this activity.

      when ( knownval( current.hadWorkItemPulled = true ) and
             knownval( current.nWorkItems > 0 )
           )
      do
      {
        conclude( current.hadWorkItemPulled = false, { fc:100, bc:0 } );
      }
    }
*/

    /* ----------------------------------------------------------------
     *                                  WFR: DoPullWorkItemFromBacklog
     * ---------------------------------------------------------------- */

    workframe DoPullWorkItemFromBacklog
    {
      repeat: true;

      variables:
        forone ( Kanban )   kanban;
        forone ( Backlog )  backlog;
        forone ( WorkItem ) workItem;
```

```
      when ( knownval( current.canAcceptNewItem = true ) and
             // knownval( current hasUpstreamActivity unknown ) and
             // knownval( current isInKanban kanban ) and
             // knownval( kanban hasBacklog backlog ) and
             knownval( current hasBacklog backlog ) and
             knownval( backlog contains workItem )
           )
      do
      {
        // TODO:  SensorProjectBacklog shouldn't be hard-wired

        pullBacklogWorkItem( backlog, workItem );

        conclude( workItem.isComplete = false, { bc:0, fc:100 } );
        conclude( current.canAcceptNewItem = false, { bc:0, fc:100 } );
      }
    }

    /* ------------------------------------------------------------------
     *                                          WFR: DoCheckIsFinished
     * ------------------------------------------------------------------ */

    workframe DoCheckIsFinished
    {
      repeat: true;

      variables:
        forone ( Kanban )   kanban;
        forone ( Backlog )  backlog;
        forone ( WorkItem ) workItem;

      when ( knownval( current hasUpstreamActivity unknown ) and
             knownval( current isInKanban kanban ) and
             knownval( kanban hasBacklog backlog ) and
             knownval( backlog.isFinished = true ) and
             knownval( current.nWorkItems = 0 ) and
             knownval( current.isFinished = false ) // and
             // not( current contains workItem )
           )
      do
      {
        conclude( current.isFinished = true, { fc:100, bc:100 } );
      }
    }
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.2 Backlog

```
class Backlog extends BaseClass, Queue
{
  resource: true;

/*
  attributes:
    public int      nWorkItems;
    public boolean  isFinished;

  relations:
```

```
    public WorkItem hasAccepted;

  initial_facts:
    ( current.nWorkItems = 0 );
    ( current.isFinished = false );
*/

  activities:

  workframes:

    /* ---------------------------------------------------------------
     *                                          WFR: DoAcceptWorkItem
     * --------------------------------------------------------------- */

/*
    workframe DoAcceptWorkItem
    {
      // repeat: true;

      variables:
        foreach ( WorkItem )   workItem;

      // Something put a work item in this backlog.

      when ( knownval( current contains workItem ) and
             not( current hasAccepted workItem )
           )
      do
      {
        conclude( current.nWorkItems = current.nWorkItems + 1, { bc:0, fc:100 } );
        conclude( current hasAccepted workItem, { bc:0, fc:100 } );
      }
    }
*/

    /* ---------------------------------------------------------------
     *                                        WFR: DoProvideWorkItem
     * --------------------------------------------------------------- */

/*
    workframe DoProvideWorkItem
    {
      // repeat: true;

      variables:
        foreach ( WorkItem )   workItem;

      // Something pulled a work item from this backlog.

      when ( knownval( current hasAccepted workItem ) and
             not( current contains workItem )
           )
      do
      {
        conclude( current.nWorkItems = current.nWorkItems - 1, { bc:0, fc:100 } );
        retractFactValue( current, hasAccepted, workItem );
      }
    }
*/

    /* ---------------------------------------------------------------
     *                                        WFR: DoCheckIsFinished
```

```
     * ------------------------------------------------------------- */

/*
    workframe DoCheckIsFinished
    {
      repeat: true;

      variables:
        forone ( WorkItem )   workItem;

      when ( knownval( current contains workItem is false ) and
             knownval( current.isFinished = false ) )
      do
      {
        conclude( current.isFinished = true, { fc:100, bc:0 } );
      }
    }
*/
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.3 Customer

```
jimport java.util.Random;

agent Customer memberof SystemGroup
{
  attributes:
    public int      nWorkItemsToCreate;
    public int      nWorkItemsCreated;
    public boolean  isInitialized;

    public java(Random) generator;

  initial_beliefs:
    ( current.nWorkItemsToCreate = 10 );
    ( current.nWorkItemsCreated = 0 );
    ( current.isInitialized = false );

  initial_facts:

  activities:

    /* ------------------------------------------------------------------
     *                                                 COA: createWorkItem
     * ------------------------------------------------------------------ */

    create_object createWorkItem( WorkItem workItem )
    {
      display: "Work Item";
      max_duration: 6;
      action: new;
      source: WorkItem;
      destination: workItem;
      when: end;
    }

    /* ------------------------------------------------------------------
     *                                                   PUT: sendWorkItem
     * ------------------------------------------------------------------ */
```

```
  get getWorkItem( WorkItem workItem )
  {
    max_duration: 0;
    items: workItem;
  }

  /* ------------------------------------------------------------------
   *                                          COM: tellWorkItemDuration
   * ------------------------------------------------------------------ */

  communicate tellWorkItemDuration( WorkItem workItem, int duration )
  {
    max_duration: 0;
    with: workItem;

    about:
      send( workItem.estimatedDuration = duration );
  }

  /* ------------------------------------------------------------------
   *                                                 PUT: sendWorkItem
   * ------------------------------------------------------------------ */

  put sendWorkItem( Backlog backlog, WorkItem workItem )
  {
    max_duration: 0;
    items: workItem;
    destination: backlog;
  }

workframes:

  /* ------------------------------------------------------------------
   *                                            WFR: DoInitialization
   * ------------------------------------------------------------------ */

  workframe DoInitialization
  {
    when ( knownval( current.isInitialized = false ) )
      do
      {
        java ( Random )   gen = new Random();

        conclude( current.generator = gen );
        conclude( current.isInitialized = true, { fc:0, bc:100 } );
      }
  }

  /* ------------------------------------------------------------------
   *                                            WFR: DoCreateWorkItem
   * ------------------------------------------------------------------ */

  workframe DoCreateWorkItem
  {
    repeat: true;

    variables:
      forone ( java(Random) ) gen;

    when (
      knownval( current.nWorkItemsCreated < current.nWorkItemsToCreate ) and
      knownval( gen = current.generator )
         )
```

```
      do
      {
        WorkItem  workItem;

        int       duration;
        double    mu = 24 * 1440;   // mean 24 hours, in seconds
        double    sigma = 6 * 1440; // sd 6 hours, in seconds
        double    dist = gen.nextGaussian() * sigma;
        double    dur = dist + mu;

        doubleToInt( dur, duration );

        println_n( "Estimated duration = %1", duration );

        createWorkItem( workItem );
        getWorkItem( workItem );

        conclude( workItem.estimatedDuration = duration, { fc:100, bc:0 } );
        conclude( current.nWorkItemsCreated = current.nWorkItemsCreated + 1, { fc:0,
bc:100 } );
      }
    }

    /* ------------------------------------------------------------------
     *                                          WFR: DoTransferWorkItem
     * ------------------------------------------------------------------ */

    workframe DoTransferWorkItem
    {
      repeat: true;

      variables:
        forone ( WorkItem )   workItem;

      when ( knownval( current contains workItem ) )
      do
      {
        sendWorkItem( SensorProjectBacklog, workItem );
      }
    }

    /* ------------------------------------------------------------------
     *                                          WFR: DoNotifyNoMoreItems
     * ------------------------------------------------------------------ */

    workframe DoNotifyNoMoreItems
    {
      repeat: true;

      when (
        knownval( current.nWorkItemsCreated >= current.nWorkItemsToCreate ) and
        knownval( SensorProjectBacklog.isFinished = false )
      )
      do
      {
        conclude( SensorProjectBacklog.isFinished = true,
                  { fc:100, bc:0 } );
      }
    }
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.4 Develop

```
object Develop instanceof Activity
{
  initial_beliefs:

  initial_facts:
    ( current.wipLimit = 5 );
    ( current isInKanban SensorProject );
    ( current hasUpstreamActivity unknown );
    ( current hasBacklog SensorProjectBacklog );
    ( current hasDoneQueue SensorProjectDoneQueue );
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.5 Developer_1

```
agent Developer_1 memberof Resource
{
  initial_beliefs:
    ( current.seniority = 1 );
    ( current isWorkingIn Develop );
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.6 Developer_2

```
agent Developer_2 memberof Resource
{
  initial_beliefs:
    ( current.seniority = 2 );
    ( current isWorkingIn Develop );
    // ( current isWorkingIn Test );
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.7 DoneQueue

```
class DoneQueue extends BaseClass, Queue
{
  resource: true;

  attributes:

  relations:
    public Activity hasUpstreamActivity;

  initial_facts:
    ( current hasUpstreamActivity unknown );
    ( current.wipLimit = 0 );       // infinite queue
    ( current.nWorkItems = 0 );

  activities:

    get pullWorkItem( Activity activity, WorkItem workItem )
    {
      max_duration: 10;
      items: workItem;
```

```
      source: activity;
    }

  workframes:

    workframe DoCheckUpstreamActivity
    {
      repeat: true;

      variables:
        forone ( Activity )   activity;
        forone ( WorkItem )   workItem;

      when ( knownval( current hasUpstreamActivity activity ) and
             knownval( activity contains workItem ) and
             knownval( workItem.isComplete = true ) and
             knownval( current.canAcceptNewItem = true )
           )
        do
        {
          pullWorkItem( activity, workItem );

          // conclude( activity.nWorkItems = activity.nWorkItems - 1, { fc:100, bc:0 }
);
          // conclude( current.nWorkItems = current.nWorkItems + 1, { fc:100, bc:0 }
);
        }
    }
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.8 Kanban

```
class Kanban extends BaseClass
{
  resource: true;

  relations:
    public Queue      hasBacklog;
    public DoneQueue  hasDoneQueue;

  initial_facts:
    ( current hasBacklog unknown );
    ( current hasDoneQueue unknown );
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.9 Queue

```
class Queue extends BaseClass
{
  resource: true;

  attributes:
    public int      wipLimit;
    public int      nWorkItems;
    public boolean  canAcceptNewItem;
    public boolean  hasModifiedQueue;
    public boolean  isFinished;
```

```
relations:
  public WorkItem hasAccepted;

initial_facts:
  ( current.nWorkItems = 0 );
  ( current.canAcceptNewItem = true );
  ( current.hasModifiedQueue = false );
  ( current.isFinished = false );

workframes:

  /* ------------------------------------------------------------------
   *                                          WFR: DoNoteInfiniteDepth
   * ------------------------------------------------------------------ */

  workframe DoNoteInfiniteDepth
  {
    when ( knownval( current.wipLimit = 0 ) )
      do
      {
        conclude( current.canAcceptNewItem = true, { fc:100, bc:0 } );
      }
  }

  /* ------------------------------------------------------------------
   *                                              WFR: DoCheckWipLimit
   * ------------------------------------------------------------------ */

  workframe DoCheckWipLimit
  {
    repeat: true;

    when ( knownval( current.hasModifiedQueue = true ) and
           knownval( current.wipLimit > 0 ) and
           knownval( current.nWorkItems < current.wipLimit ) )
      do
      {
        conclude( current.canAcceptNewItem = true, { fc:100, bc:0 } );
        conclude( current.hasModifiedQueue = false, { bc:0, fc:100 } );
      }
  }

  /* ------------------------------------------------------------------
   *                                              WFR: DoNoteQueueFull
   * ------------------------------------------------------------------ */

  workframe DoNoteQueueFull
  {
    repeat: true;

    when ( knownval( current.hasModifiedQueue = true ) and
           knownval( current.wipLimit > 0 ) and
           knownval( current.nWorkItems >= current.wipLimit ) )
      do
      {
        conclude( current.canAcceptNewItem = false, { fc:100, bc:0 } );
        conclude( current.hasModifiedQueue = false, { bc:0, fc:100 } );
      }
  }

  /* ------------------------------------------------------------------
   *                                            WFR: DoAcceptWorkItem
```

```
     * ---------------------------------------------------------------- */

    workframe DoAcceptWorkItem
    {
      variables:
        foreach ( WorkItem )   workItem;

      // Something put a work item in this backlog.

      when ( knownval( current contains workItem ) and
             not( current hasAccepted workItem )
           )
      do
      {
        conclude( current.nWorkItems = current.nWorkItems + 1, { bc:0, fc:100 } );
        conclude( current.hasModifiedQueue = true, { bc:0, fc:100 } );
        conclude( current hasAccepted workItem, { bc:0, fc:100 } );
      }
    }

    /* ----------------------------------------------------------------
     *                                          WFR: DoProvideWorkItem
     * ---------------------------------------------------------------- */

    workframe DoProvideWorkItem
    {
      variables:
        foreach ( WorkItem )   workItem;

      // Something pulled a work item from this backlog.

      when ( knownval( current hasAccepted workItem ) and
             not( current contains workItem )
           )
      do
      {
        conclude( current.nWorkItems = current.nWorkItems - 1, { bc:0, fc:100 } );
        conclude( current.hasModifiedQueue = true, { bc:0, fc:100 } );
        retractFactValue( current, hasAccepted, workItem );
      }
    }

    /* ----------------------------------------------------------------
     *                                          WFR: DoCheckIsFinished
     * ---------------------------------------------------------------- */

    workframe DoCheckIsFinished
    {
      repeat: true;

      variables:
        forone ( WorkItem )   workItem;

      when ( knownval( current contains workItem is false ) and
             knownval( current.isFinished = false ) )
      do
      {
        conclude( current.isFinished = true, { fc:100, bc:0 } );
      }
    }
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.10 Resource

```
jimport java.util.Random;
jimport java.io.PrintStream;

group Resource memberof SystemGroup
{
  attributes:
    public boolean      isFinished;
    public boolean      activityHasWorkItem;
    public boolean      isInitialized;
    public boolean      isLookingForWork;
    public boolean      hasCheckedQueue;
    public boolean      hasNotifiedTeammates;
    public boolean      hasCheckedConflicts;
    public boolean      hasConflict;
    public boolean      hasWorkItem;
    public int          seniority;

    public java(Random) generator;

  relations:
    public Activity     isWorkingIn;
    public WorkItem     isWorkingOn;
    public WorkItem     hasSelected;
    public WorkItem     ignoreConflicting;

  initial_beliefs:
    ( current.isFinished = false );
    ( current.activityHasWorkItem = false );
    ( current.isInitialized = false );
    ( current.isLookingForWork = false );
    ( current.hasCheckedQueue = false );
    ( current.hasNotifiedTeammates = false );
    ( current.hasCheckedConflicts = false );
    ( current.hasConflict = false );
    ( current.hasWorkItem = false );

    ( current isWorkingIn unknown );
    ( current isWorkingOn unknown );
    ( current hasSelected unknown );

  initial_facts:

  activities:

    /* ------------------------------------------------------------------
     *                                  BCT: announceProjectMembership
     * ------------------------------------------------------------------ */

    broadcast announceProjectMembership()
    {
      about: send( current isWorkingIn );
    }

    /* ------------------------------------------------------------------
     *                                          BCT: announceSeniority
     * ------------------------------------------------------------------ */

    broadcast announceSeniority()
    {
```

```
  about: send( current.seniority );
}

/* ------------------------------------------------------------------
 *                                                  CAC: checkForWork
 * ------------------------------------------------------------------ */

composite_activity checkForWork( Activity activity )
{
  detectables:

    /* ------------------------------------------------------------
     *                                       DET: activityIsFinished
     * ------------------------------------------------------------ */

    detectable activityIsFinished
    {
      when ( whenever )
        detect ( ( activity.isFinished = true ) )
        then end_activity;
    }

  activities:

    /* ------------------------------------------------------------
     *                                          PAC: lookForWorkItem
     * ------------------------------------------------------------ */

    primitive_activity lookForWorkItem()
    {
      min_duration: 6;
      max_duration: 300;
    }

    /* ------------------------------------------------------------
     *                                           PAC: waitForAwhile
     * ------------------------------------------------------------ */

    primitive_activity waitForAwhile( int awhile )
    {
      max_duration: awhile;
    }

    /* ------------------------------------------------------------
     *                                             GET: getWorkItem
     * ------------------------------------------------------------ */

    get getWorkItem( Activity activity, WorkItem workItem )
    {
      items:  workItem;
      source: activity;
      when:   start;
    }

    /* ------------------------------------------------------------
     *                                  COM: notifySelectedWorkItem
     * ------------------------------------------------------------ */

    communicate notifySelectedWorkItem( Resource teammate )
    {
      with: teammate;
      about: send( current hasSelected );
    }
```

```
workframes:

  /* ------------------------------------------------------------
   *                                   WFR: DoCheckActivityQueue
   * ------------------------------------------------------------ */

  workframe DoCheckActivityQueue
  {
    detectables:
      detectable activityHasWorkItems
      {
        when ( whenever )
          detect ( ( activity contains ? ) )
          then complete;
      }

    when ( unknown( current contains ) and
           unknown( current hasSelected ) and
           knownval( current.isLookingForWork = true ) and
           knownval( current.hasCheckedQueue = false )
         )
      do
      {
        lookForWorkItem();

        conclude( current.hasCheckedQueue = true, { bc:100, fc:0 } );
      }
  }

  /* ------------------------------------------------------------
   *                                       WFR: DoSelectWorkItem
   * ------------------------------------------------------------ */

  workframe DoSelectWorkItem
  {
    variables:
      forone ( WorkItem ) workItem;

    when ( knownval( current.hasCheckedQueue = true ) and
           unknown( current hasSelected ) and
           unknown( current contains ) and
           knownval( activity contains workItem ) and
           not( current ignoreConflicting workItem )
         )
      do
      {
        conclude( current.hasNotifiedTeammates = false, { bc:100, fc:0 } );
        conclude( current.hasCheckedConflicts = false, { bc:100, fc:0 } );
        conclude( current hasSelected workItem, { bc:100, fc:0 } );
        retractBelief( current, ignoreConflicting );
      }
  }

  // NOTE:  This whole issue of two Resources selecting the same
  // WorkItem could have been avoided by making the Activity
  // Queue active.  That is, if Resources asked the Queue for an
  // available item, the Queue could actively resolve any
  // conflict.  This would be the model for a computer-based
  // Kanban board, rather than a paper-based one.

  /* ------------------------------------------------------------
   *                                     WFR: DoNotifyTeammates
```

```
 * ------------------------------------------------------------ */

workframe DoNotifyTeammates
{
  variables:
    forone ( WorkItem )     workItem;
    collectall ( Resource ) teammate;

  when ( knownval( current.hasNotifiedTeammates = false ) and
         knownval( current hasSelected workItem ) and
         knownval( teammate isWorkingIn activity ) and
         knownval( teammate != current )
       )
    do
    {
      notifySelectedWorkItem( teammate );

      conclude( current.hasNotifiedTeammates = true, { bc:100, fc:0 } );
      conclude( current.hasCheckedConflicts = false, { bc:100, fc:0 } );
    }

    // TODO: It's unnecessary to notify all the teammates,
    // because only those presently looking for a WorkItem
    // would care.
}

/* ------------------------------------------------------------
 *                                      WFR: DoWaitForTeammates
 * ------------------------------------------------------------ */

workframe DoWaitForTeammates
{
  variables:
    forone ( WorkItem )   workItem;

  when ( knownval( current.hasNotifiedTeammates = true ) and
         knownval( current.hasCheckedConflicts = false )
       )
    do
    {
      // wait to see if any teammate also hasSelected
      waitForAwhile( 1 );

      conclude( current.hasCheckedConflicts = true, { bc:100, fc:0 } );
    }
}

/* ------------------------------------------------------------
 *                                    WFR: DoDetectNoConflicts
 * ------------------------------------------------------------ */

workframe DoDetectNoConflicts
{
  variables:
    forone ( WorkItem )     workItem;
    collectall ( Resource ) teammate;

  when ( knownval( current.hasConflict = true ) and
         knownval( current.hasCheckedConflicts = true ) and
         knownval( current hasSelected workItem ) and
         knownval( teammate isWorkingIn activity ) and
         knownval( teammate != current ) and
         not( teammate hasSelected workItem )
```

```
        )
    do
    {
      conclude( current.hasConflict = false, { bc:100, fc:0 } );
    }
}

/* ------------------------------------------------------------
 *                                    WFR: DoDetectAnyConflicts
 * ------------------------------------------------------------ */

workframe DoDetectAnyConflicts
{
  variables:
    forone ( WorkItem )     workItem;
    forone ( Resource )     teammate;

  when ( knownval( current.hasConflict = false ) and
         knownval( current.hasCheckedConflicts = true ) and
         knownval( current hasSelected workItem ) and
         knownval( teammate isWorkingIn activity ) and
         knownval( teammate != current ) and
         knownval( teammate hasSelected workItem )
       )
    do
    {
      conclude( current.hasConflict = true, { bc:100, fc:0 } );
    }
}

/* ------------------------------------------------------------
 *                                WFR: DoResolveConflictAsLoser
 * ------------------------------------------------------------ */

workframe DoResolveConflictAsLoser
{
  variables:
    forone ( WorkItem ) workItem;
    forone ( Resource ) teammate;

  when ( knownval( current.hasConflict = true ) and
         knownval( current hasSelected workItem ) and
         knownval( teammate isWorkingIn activity ) and
         knownval( teammate != current ) and
         knownval( teammate hasSelected workItem ) and
         knownval( current.seniority > teammate.seniority )
       )
    do
    {
      conclude( current ignoreConflicting workItem, { bc:100, fc: 0 } );
      retractBeliefValue( current, hasSelected, workItem );
    }
}

/* ------------------------------------------------------------
 *                               WFR: DoResolveConflictAsWinner
 * ------------------------------------------------------------ */

workframe DoResolveConflictAsWinner
{
  variables:
    forone ( WorkItem )   workItem;
    foreach ( Resource )  teammate;
```

```
  when ( knownval( current.hasConflict = true ) and
         knownval( current hasSelected workItem ) and
         knownval( teammate isWorkingIn activity ) and
         knownval( teammate != current ) and
         knownval( teammate hasSelected workItem ) and
         ( current.seniority < teammate.seniority )
       )
    do
    {
      // forget that teammate selected this work item
      retractBeliefValue( teammate, hasSelected, workItem );

      // When this workframe ends, we believe that we alone
      // have selected this work item.
    }
}

/* ------------------------------------------------------------
 *                              WFR: DoForgetTeammateSelections
 * ------------------------------------------------------------ */

workframe DoForgetTeammateSelections
{
  variables:
    foreach ( Resource )  teammate;
    forone ( WorkItem )   workItem;

  // If we have nothing selected, we don't care what the other
  // teammates have selected.  This cleans up after the
  // decision reached in DoResolveConflictAsLoser.

  when ( knownval( teammate hasSelected workItem ) and
         knownval( current != teammate ) and
         not( current hasSelected workItem )
       )
    do
    {
      retractBeliefValue( teammate, hasSelected, workItem );
    }
}

/* ------------------------------------------------------------
 *                                          WFR: DoGetWorkItem
 * ------------------------------------------------------------ */

workframe DoGetWorkItem
{
  variables:
    forone ( WorkItem )     workItem;
    collectall ( Resource ) teammate;

    when ( knownval( current.hasCheckedConflicts = true ) and
           knownval( current.hasConflict = false ) and
           knownval( current hasSelected workItem ) and
           not( current contains workItem ) and
           knownval( teammate isWorkingIn activity ) and
           knownval( teammate != current ) and
           not( teammate hasSelected workItem )
         )
    do
    {
      getWorkItem( activity, workItem );
```

```
          conclude( current isWorkingOn workItem, { bc:100, fc:0 } );
          conclude( current.isLookingForWork = false, { bc:100, fc:0 } );
          retractBelief( current, hasSelected );
        }
    }

  thoughtframes:
}

/* ------------------------------------------------------------
 *                                          CAC: processWorkItem
 * ------------------------------------------------------------ */

composite_activity processWorkItem( Activity activity, WorkItem workItem )
{
  detectables:

    /* ------------------------------------------------------------
     *                                    DET: workItemIsComplete
     * ------------------------------------------------------------ */

    detectable workItemIsComplete
    {
      when ( whenever )
        detect ( ( workItem.isComplete ) )
        then continue;
    }

    /* ------------------------------------------------------------
     *                                    DET: hasDoneQueue
     * ------------------------------------------------------------ */

    detectable hasDoneQueue
    {
      when ( whenever )
        detect ( ( activity hasDoneQueue ) )
        then continue;
    }


  activities:

    primitive_activity checkWorkItemDuration()
    {
      max_duration: 6;
    }

    primitive_activity processWorkItem( int duration )
    {
      max_duration: duration;
    }

    primitive_activity waitForDownstreamToTakeItem()
    {
      max_duration: 60;
    }

  workframes:

    workframe CheckWorkItemDuration
    {
      detectables:
```

```
      detectable findWorkItemDuration
      {
        when ( whenever )
          detect ( ( workItem.estimatedDuration = ? ) )
          then complete;
      }

    when ( unknown( workItem.estimatedDuration ) )
      do
      {
        checkWorkItemDuration();
      }
  }

  workframe HasWorkItemNeedingWork
  {
    variables:
      forone ( double )         duration;
      forone ( java( Random ) ) gen;

    when ( knownval( workItem.isComplete = false ) and
           knownval( workItem.estimatedDuration > 0 ) and
           knownval( duration = workItem.estimatedDuration ) and
           knownval( gen = current.generator )
         )
      do
      {
        int             actualDuration;
        double          sigma = 0.25;
        double          sigma2 = Math.pow( sigma, 2.0 );

        // duration value for mode (for median, drop sigma2 term)
        double          mu = Math.log( duration ) + sigma2;
        double          sigmaN = sigma * gen.nextGaussian();
        double          dur = Math.exp( mu + sigmaN );

        doubleToInt( duration /*dur*/, actualDuration );

        println_n( "Duration = %1", actualDuration );

        processWorkItem( actualDuration );

        conclude( workItem.isComplete = true, { bc:100, fc:100 } );
      }
  }

  workframe HasCompletedWorkItem
  {
    variables:

/*
    detectables:
      detectable workItemTaken
      {
        when ( whenever )
          detect ( ( activity contains workItem is false ) )
          then complete;
      }
*/

    when ( knownval( workItem.isComplete = true )
           // knownval( activity hasDoneQueue doneQueue )
         )
```

```
          do
          {
            waitForDownstreamToTakeItem();
          }
      }

      workframe HasReleasedWorkItem
      {
        when ( knownval( current isWorkingOn workItem ) and
               knownval( activity contains workItem is false )
             )
          do
          {
            conclude( current isWorkingOn unknown, { bc:100, fc:0 } );
            conclude( current.activityHasWorkItem = false, { bc:100, fc:0 } );
          }
      }
  }


workframes:

  /* ------------------------------------------------------------------
   *                                              WFR: DoInitialization
   * ------------------------------------------------------------------ */

  workframe DoInitialization
  {
    when ( knownval( current.isInitialized = false ) )
      do
      {
        java ( Random )   gen = new Random( 19560516L );

        conclude( current.generator = gen );
        conclude( current.isInitialized = true, { bc:100, fc:0 } );

        announceProjectMembership();
        announceSeniority();
      }
  }

  /* ------------------------------------------------------------------
   *                                                WFR: DoCheckForWork
   * ------------------------------------------------------------------ */

  workframe DoCheckForWork
  {
    repeat: true;

    variables:
      forone ( Activity ) activity;
      forone ( Resource ) another;

    detectables:

    when ( knownval( current.isLookingForWork = true ) and
           knownval( current isWorkingIn activity ) and
           unknown( current contains )
         )
      do
      {
        conclude( current.hasCheckedQueue = false, { bc:100, fc:0 } );
        checkForWork( activity );
```

```
      }
  }

  /* ------------------------------------------------------------------
   *                                              WFR: DoProcessWorkItem
   * ------------------------------------------------------------------ */

  workframe DoProcessWorkItem
  {
    repeat: true;

    variables:
      forone ( Activity ) activity;
      forone ( WorkItem ) workItem;

    // The activity this resource is working in has an incomplete
    // work item.

    when ( knownval( current contains workItem ) and
           knownval( current isWorkingOn workItem ) and
           knownval( workItem.isComplete = false ) and
           knownval( current isWorkingIn activity )
         )
    do
    {
      processWorkItem( activity, workItem );
    }
  }

thoughtframes:

  /* ------------------------------------------------------------------
   *                                        TFR: DoCheckActivtyStatus
   * ------------------------------------------------------------------ */

  thoughtframe DoCheckActivityStatus
  {
    repeat: true;

    variables:
      forone ( Activity )   activity;

    when ( knownval( current isWorkingIn activity ) and
           knownval( activity.isFinished = true ) and
           knownval( current.isFinished = false )
         )
      do
      {
        conclude( current.isFinished = true, { bc:100, fc:0 } );
        conclude( current.isLookingForWork = false, { bc:100, fc:0 } );
      }
  }

  /* ------------------------------------------------------------------
   *                                        TFR: DoConsiderWorkStatus
   * ------------------------------------------------------------------ */

  thoughtframe DoConsiderWorkStatus
  {
    repeat: true;

    variables:
      forone ( WorkItem )   workItem;
```

```
      when ( knownval( current.isFinished = false ) and
             knownval( current.isLookingForWork = false ) and
             unknown( current contains )
           )
        do
        {
          conclude( current.isLookingForWork = true, { bc:100, fc:0 } );
        }
    }
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.11 SensorProject

```
object SensorProject instanceof Kanban
{
  initial_facts:
    ( current hasBacklog    SensorProjectBacklog );
    // ( current hasDoneQueue  SensorProjectDoneQueue );
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.12 SensorProjectDoneQueue

```
object SensorProjectDoneQueue instanceof Queue
{
  initial_facts:
    ( current.wipLimit = 0 );
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.13 Workitem

```
class WorkItem extends BaseClass
{
  resource: false;

  attributes:
    public int      estimatedDuration;
    public boolean  isComplete;
    // public boolean  isQueued;
    public map      needsSpecializations;

  relations:
    public Resource isBeingWorkedBy;

  initial_beliefs:
    // ( current.estimatedDuration = unknown );
    // ( current.isComplete = false );

  initial_facts:
    ( current.estimatedDuration = unknown );
    ( current.isComplete = false );
    // ( current.isQueued = false );

    ( current isBeingWorkedBy unknown );
```

```
  activities:

  workframes:
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.14 SensorProjectBacklog

```
object SensorProjectBacklog instanceof Backlog
{
}

// vim:ts=2:sw=2:sts=2:et
```

6.2.1.15 LeanSE

```
import *;
```