An Event-driven, Value-based, Pull Systems Engineering Scheduling Approach

Richard Turner
School of Systems and Enterprises
Stevens Institute
Hoboken, NJ, USA
rturner@stevens.edu

Jo Ann Lane and Dan Ingold
Center for Systems and Software Engineering
University of Southern California
Los Angeles, CA, USA
{jolane, dingold} at usc.edu

Raymond Madachy
Department of Systems Engineering
Naval Postgraduate School
Monterrey, CA, USA
rjmadach@nps.edu

David Anderson
David J. Anderson Associates
Seattle, WA, USA
dja@djaa.com

***Abstract*— Effective application of systems engineering in rapid response environments has been difficult, particularly those where large, complex brownfield systems or systems of systems exist and are constantly being updated with both short and long term software enhancements. This paper proposes a general case for solving this problem by combining a services approach to systems engineering with a kanban-based scheduling system. It provides the basis for validating the approach with agent-based simulations.**

Keywords-systems engineering; systems engineering process; lean; kanban; process simulation

I. INTRODUCTION AND BACKGROUND

Traditional systems engineering developed half a century ago, primarily driven by the challenges faced in the aerospace and defense industries. In rapid or continuous deployment environments, where requirements are not precise and can change or emerge quickly, traditional systems engineering has often failed to perform its tasks within the available schedule and resource bounds [1], [2]. Clearly, new and flexible methods, processes and tools are required for effective systems engineering in these environments.

Engineering principles involving agility and leanness have been adopted to address non-determinism in software systems. However, integrating these agility and leanness concepts into the systems engineering workflow has proven difficult. Leveraging work done in earlier research [3], [4], agile and lean practice research [5–7], and including new experience with lean approaches [8], [9], we are investigating the use of flow-based pull scheduling techniques (kanban systems) in a rapid response development environment.

A kanban scheduling approach provides a visual means of managing the flow within a process. The fundamental idea is 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 [8]. 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 a "pull" system, since the work is pulled into the process rather than "pushed" via a schedule.

A software kanban system is usually implemented as a set of process steps, each step with its own queue and set of resources, that add value to development work units that flow through them. The fact that queues are included in the system allows costs of delay and other usually invisible aspects of scheduling to be front and center in decision making. The visual representation of the work, usually via a kanban board (Fig. 1), is critical to kanban success, because it provides immediate understanding of the state of flow through the set of process activities.

This transparency makes apparent process delays or resource issues and enables the team to recognize and react immediately to resolve the cause. The process is managed through *Work in Progress (WIP)* limits, *small batch siz*es, and *Classes-of-Service* (COS) definitions that prioritize work with respect to value and risk. Flow is measured and tracked through statistical methods that provide insight to tune and improve the system.

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 tasks that have been started and not completed. *Limiting WIP* is a concept to control flow

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.

978-1-4673-0750-5/12/$31.00 ©2012 IEEE

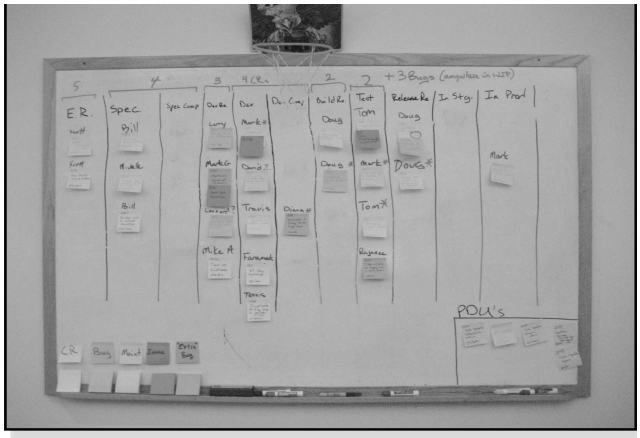

Figure 1. Example Kanban Board [8]

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 can lower the context-switching overhead that impacts individuals or teams attempting to handle several simultaneous tasks; they can accelerate value by completing higher value work before starting lower value work; and, they can provide for reasonable resource work loads over time.

Using *small batch sizes* is a supporting concept to WIP to further limit rework and provide flexibility in scheduling and response to unforeseen change. Smaller batch sizes even out 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.

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.

II. Predicted Benefits of the Proposed Approach

A. More Effective Integration and Use of Scarce Systems Engineering Resources

Using a KSS and applying a model of SE based on continuous activities and taskable 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 process 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.

B. 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 minimal planning. 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.

C. Visibility and Coordination Across Multiple Projects

In highly concurrent engineering tasks, the KSS provides a means of synchronizing activities across mutually dependent teams by coordinating their activities through changing value functions (task priority) according to the degree of data completeness and maturity (risk of change). The visible monitoring of a kanban board also provides an excellent way to show where tasks are and the status of work-in-progress and queued or blocked work.

D. 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 on the visible representation of the flow status and so are clear to all who take part in scheduling, including management. Measurements clearly identify problems and track improvements.

III. Defining the Approach

In Fig. 2, Fig. 3, and Table I, 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 tasks 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.

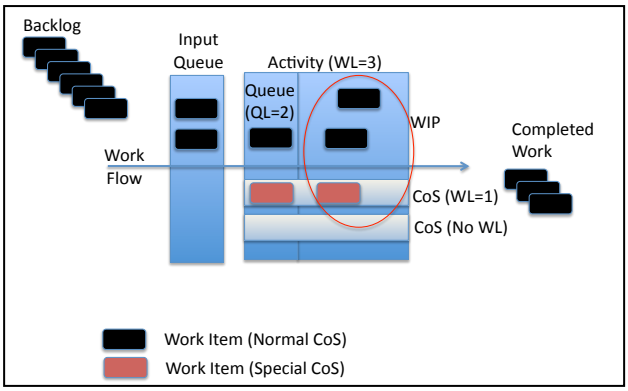

Figure 2. Kanban Scheduling System Model

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 often narrows the scope, and detailed up-front analysis 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 a pull system for systems engineering is an attempt to merge the breadth of SE into the rapid development rather than lay it on top of the activities. Our idea of a KSS for systems engineering is shown in Fig. 4. 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.

A. Systems Engineering as a Service

In general, systems engineering is involved in three kinds of activities in rapid response environments: Up front, continuous, and taskable.

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.

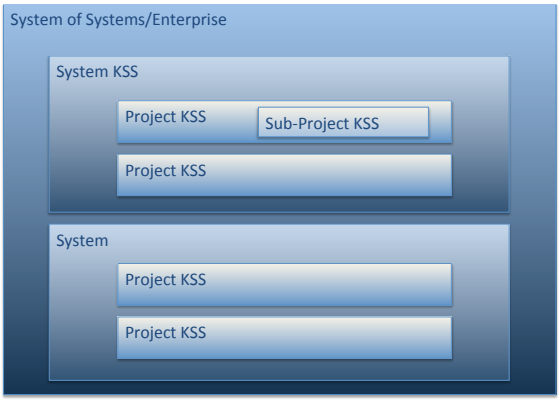

Figure 3. Kanban Scheduling System Hierarchy

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.

Taskable activities are generally specific to individual

TABLE I. KANBAN SCHEDULING SYSTEM DEFINITIONS

Term	Definition
Work Item	The item controlled in the kanban system; essentially, the kanban carrier
Effort Required	Determines the approximate size of work in person-units of time. May be a negotiated function of desired quality.
Backlog	A non-WIP-limited queue containing work items items awaiting service by the initial activity in a kanban system.
Cadence	The rhythm of the production system. Not necessarily an iteration. Kanban still allows for iterations but decouples prioritization, delivery and cycle time to vary naturally according to the domain and its intrinsic costs. The average transit time of a work item through a kanban system.
Activity	Value-adding work that can be determined as complete. Includes: activity queue, a set of resources, and a WIP Limit. Represents an allocation of the effort required to complete a work item.
Resource	An agent for accomplishing work; may be generic or have specialized expertise. Includes: expertise-productivity pair(s), where productivity is in effort per unit time. Usually associated with a specific activity, but may be shared across activities.
Procedure for Selecting Next Work Item	Rule for selecting the next work item from a 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.
Activity Queue	Holds work items within an Activity that are awaiting processing. The sum of items in process and items in activity queue must be within the WIP limit for each CoS.
WIP Limit	Limit of work items allowed at one time within an activity.
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.
Flow Metrics	Includes cumulative flow charting and average transit (lead) time.

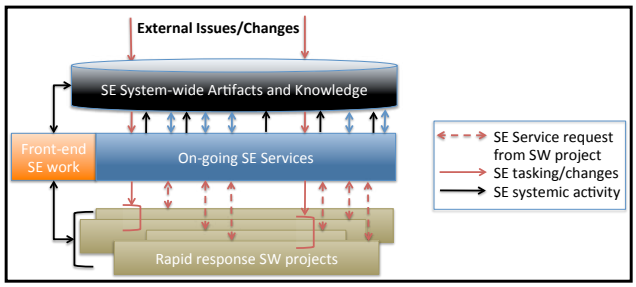

Figure 4. Overview of SE as a Service concept

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 tasks 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 requested 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. The value function used to select the next request to be handled must be designed to identify the highest cost of delay among the queued requests in terms of the overall system value. This allows SE to be a 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 II describes categories of services, specific characteristics.

IV. Modeling the Approach

A. Goals of the Model

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 initial-value
- Highest-value in the quickest-time
- Highest-value for a given-time

B. Modeling Strategies

Three approaches to modeling were considered for this research:

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

As seen in Fig. 5, each of these modeling approaches has advantages for the problem domain and level of abstraction.

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 [14]. They work by accumulating continuous flow quantities (representing a quantity of documents, tasks, personnel, etc.) over time to create cumulative "levels" of those quantities. A given flow and its

TABLE II. Systems Engineering Service Categories

Category	Description	Usage
Translating Capability Objectives	Proxy for customer; support for requirements management activities	Continuous; Taskable
Understanding Systems and Relationships	View across multiple projects; Persistent memory across time and teams	Continuous; Taskable
Assessing Performance Against Capability Objectives	Validation of TPMs or other performance requirements; typical V&V type activities	Continuous; Taskable
Developing and Evolving Architecture	Providing design guidance and supporting common architectural patterns across multiple projects	Continuous; Taskable
Monitoring and Assessing Changes	Supporting flexibility and agility by providing surveillance of the external environment and identifying issues and changes that might affect projects	Continuous; Taskable
Trade Studies and Decision Support	Supporting system-informed decision making by providing independent, competent analytical services to the projects	Taskable

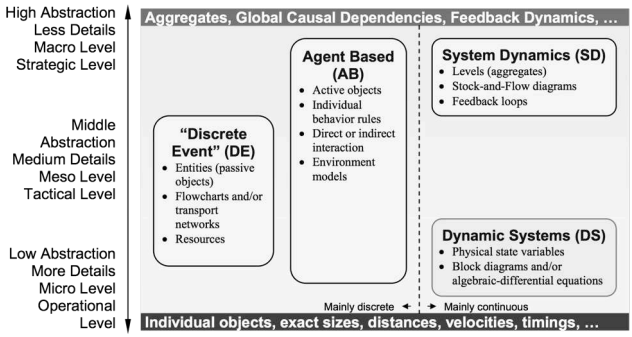

Figure 5. Modeling approach vs. abstraction level [14]

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 seems 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 [14]. 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 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.

C. 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 that provides an integrated framework within which ABMs are developed. Other tools considered include MASON and Swarm.

Both tools exhibit steep learning curves. 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, however, the additional analysis tools that Repast provides may make it worthwhile to convert the model in subsequent research.

D. Agent-based model design

Similar to the discrete-event strategy described in [15], the elements of the agent-based model include the concepts: Kanban, Backlog queue, Activities, Resources, Work Items, Release queue, Customer and WIP limits

E. Model workflow

Figure 6 diagrams the relationship of these concepts within the agent-based model for the KSS. The model is composed of one or more kanbans, each of which represents a project or, as will be seen, a pan-project team. Each kanban is composed of a backlog queue, one or more serialized activities, and a release queue. 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

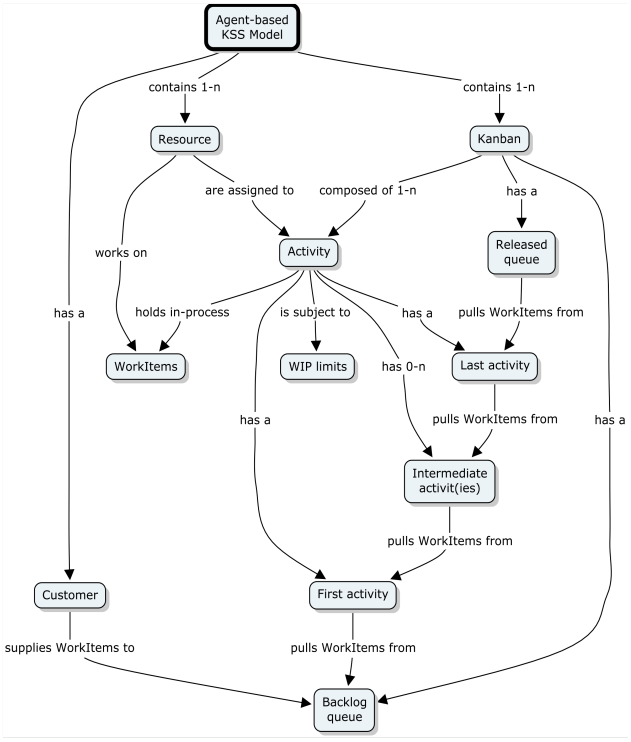

Figure 6. Agent-based model of kanban-based scheduling system

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 by those resources. 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 [15]. 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 modeling of more realistic projects, to see how the interaction of multiple kanbans might affect project performance.

F. Development-SE feedback

The high-level flow of information through the KSS is presented in Figure 7. 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 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.

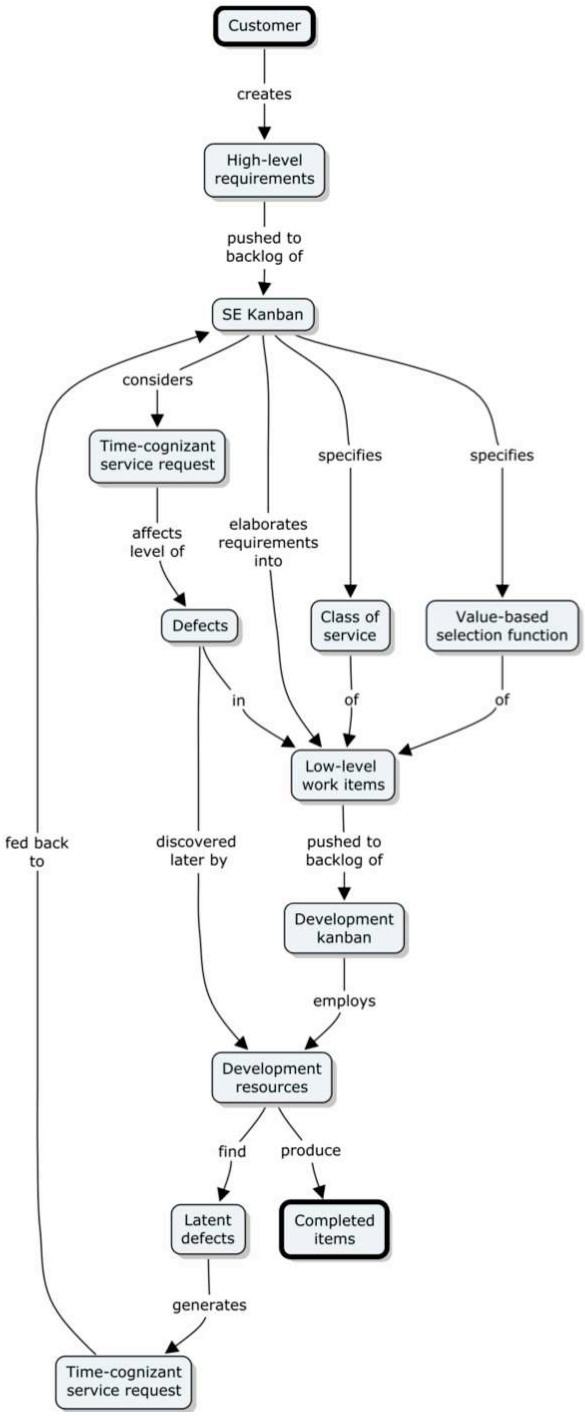

Figure 7. Information flow through KSS

V. CONCLUSIONS AND FUTURE WORK

The work so far has been in progressive elaboration of the concept, model, and simulations. Initial results have been

promising, but more work is needed to better understand how SE services are defined and requested, how the various levels of systems engineering (enterprise, system and project) establish value, how the various value functions are actually applied, and whether the social aspects of such a model impact its viability.

A number of industry and government entities have expressed interest in this approach, and some support for pilot opportunities. To support that interest, the next phase of the research will develop a demonstration and research platform for introducing the kanban-based, services approach to systems engineering in rapid response environments. The approach will be refined to specifically address a three-level SE hierarchy including project/program, portfolio, and enterprise levels of systems engineering activity. Infrastructure for simulating social aspects will be included, but no specific research in this area will occur in this phase.

The platform will consist of an integrated set of simulations and a user interface. It will include reference baselines against which new simulations runs may be compared, as well as a means of storing information gathered in non-attributive fashion for benchmarking.

Testing of the platform will include comparison to the reference baselines for a typical application of the approach. An experimental validation of the kanban/SE as service concept will be conducted using an actual historical project with inter-task dependencies, relative stakeholder values for the task, pre-effort planning information, and post effort actuals. The experiment will use the task sizes, values and dependencies to compare how well the IMS and KSS approaches achieve one or more of the value goals (e.g. highest value earliest, most total value).

REFERENCES

[1] NDIA - National Defense Industries Association, "Top Systems Engineering Issues In US Defense Industry." Systems Engineering Division Task Group Report, Sep-2010.

[2] R. Turner and F. Shull, "Evaluation of Systems Engineering Methods, Processes and Tools on Department of Defense and Intelligence Community Programs: Phase I Final Report," Systems Engineering Research Center, SERC-2009-TR-002, Sep. 2009.

[3] R. Turner and F. Shull, "Evaluation of Systems Engineering Methods, Processes and Tools on Department of Defense and Intelligence Community Programs: Phase 2 Final Report," Systems Engineering Research Center, SERC-2009-TR-004, Dec. 2009.

[4] R. Turner and J. Wade, "Lean Systems Engineering within System Design Activities," in *Proceedings of the 3rd Lean System and Software Conference*, Los Angeles, CA, 2011.

[5] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Addison-Wesley, 2004.

[6] C. Larman and B. Vodde, *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Pearson Education, 2008.

[7] M. Poppendieck and T. Poppendieck, *Implementing Lean Software Development: From Concept to Cash*, 1st ed. Addison-Wesley Professional, 2006.

[8] D. J. Anderson and D. G. Reinertsen, *Kanban: Successful Evolutionary Change for Your Technology Business*. Sequim, Washington: Blue Hole Press, 2010.

[9] D. G. Reinertsen, *The Principles of Product Development Flow: Second Generation Lean Product Development*, 1st ed. Celeritas Publishing, 2009.

[10] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley, 2003.

[11] J. M. Morgan and J. K. Liker, *The Toyota Product Development System: Integrating People, Process And Technology*. Productivity Press, 2006.

[12] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*, 3rd Revised. North River Pr, 2004.

[13] B. Boehm, "Applying the Incremental Commitment Model to Brownfield System Development," *Proceedings, CSER*, 2009.

[14] A. Borshchev and A. Filippov, "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*, 2004, pp. 25–29.

[15] D. Anderson, G. Concas, M. I. Lunesu, and M. Marchesi, "Studying Lean-Kanban Approach Using Software Process Simulation," in *Agile Processes in Software Engineering and Extreme Programming*, vol. 77, A. Sillitti, O. Hazzan, E. Bache, and X. Albaladejo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 12–26.