Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2014) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

Conference on Systems Engineering Research (CSER 2014)

Eds.: Azad M. Madni, University of Southern California; Barry Boehm, University of Southern California;
Michael Sievers, Jet Propulsion Laboratory; Marilee Wheaton, The Aerospace Corporation
Redondo Beach, CA, March 21-22, 2014

A Hybrid System Engineering Approach for Engineered Resilient Systems: Combining Traditional and Agile Techniques to Support Future System Growth

Shamsnaz Virani[a]* and Lauren Stolzar[b]

[a]*Worcester Polytechnic Institute, 100 Instiute Rd, Worcester, MA 01609*
[b]*BAE Systems, 6New England Exceutive Park, Burlington, MA 01803*

Abstract

Accommodating new requirements and user driven features is the major challenge of engineered resilient systems. An enterprise-centric system-engineering paradigm provides a holistic framework for guiding future growth of systems. This paradigm requires new processes to support such growth. Traditional system engineering approaches, while helpful in developing system structure and framework, are not adaptive to future growth. New user desired features or requirements can be added to legacy systems, but typically entail unfortunate tradeoffs. To meet new requirements, engineered resilient systems have adopted a hybrid approach of using traditional and agile system engineering. Traditional system engineering approaches handle original concept development and design requirements, while the specific implementation is agile. The hybrid approach presents some application issues such as abstraction (how much of the systems should be designed before agile implementation). Others include how to facilitate user features requiring major design changes; and what are acceptable tradeoffs. These issues are particularly problematic during verification and validation phases when a system fails to meet a particular requirement, which necessitates a design change. In this paper, we present a real world case study of a large-scale data processing and analysis system. Design and implementation of this system took place over two years using a mix of traditional and agile system engineering approaches. This paper presents the

* Corresponding author. Tel.: +1-508-831-4848; fax: +1-508-831-5491.
E-mail address: ssvirani@wpi.edu

Selection and peer-review under responsibility of the University of Southern California.
Approved for Public Release; Distribution Unlimited. Cleared for Open Publication on 1/10/2014.

current best practices, customization, and the issues of abstraction, design agility, user accommodation, and tradeoffs while using hybrid system engineering approaches. We include lessons learned from the case study and suggest future guidelines.

Selection and peer-review under responsibility of the University of Southern California.

*Keywords:*Agile Systems Engineering, Lessons Learned, Case Study, Hybrid Approaches, Engineering Resilent Systems, Software Systems Engineering

1. Introduction

Designing software-dominated systems to be adaptable to change has become increasingly critical when building user driven systems. While it is possible to define an initial set of requirements up front, adapting to user needs is necessary and often requires significant scope change. Incorporating multiple design paradigms into the system development process enables requirements to continue to change throughout the life of a system while allowing users to start taking advantage of existing functionality. A key insight is the benefit of applying the right paradigm given the stage of system development. Switching between paradigms during system development constitutes a novel hybrid approach to system engineering. This paper describes a case study where a hybrid approach clearly contributed to the achievement of challenging system development goals.

2. Literature review

Software dominates most new system development (Bohem, 2006). Software is critical for systems to remain competitive, but software dominated systems present a new set of system engineering challenges. Most recently, the debacle with healthcare.gov exemplifies the magnitude of the issues faced by most systems engineers of software systems. One major issue is the introduction of new requirements – a.k.a. requirements creep. However, they are a fact of system life. One way software system engineering deals this inevitability is spiral development, however the spiral methodology often leads to undocumented systems (Bijan *et al.*, 2013). The use of traditional methods results in systems that are well structured but don't allow room to grow. Conversely, the use of agile methods results in systems with room to grow but no well-defined path or end point.

Bijan *et al.* (2013) reviewed 15 methods of requirements gathering and design development. Among them was Agile. The Agile manifesto focuses on individuals and interactions, working software, customer collaboration, and responding to change. Lack of documentation during requirements gathering presents issues for future growth of the system (Bijan *et al, 2013.*). Software developers often jump into code implementation prior to having completed adequate design. The resulting development can provide products for customer use, but the software embodies no planning to accommodate future functionality and growth. Another common issue with software intensive systems is systems integration. System success depends on effective systems integration, whereby all components come together – with the subsystems building the system. Madni and Sievers (2013) state "in some of the so-called 'agile' design methodologies, delivery speed trumps stability, functionality and robustness". They suggest development of the mission model and mission verification and validation to test the systems integration before the actual system development and propagation of issues that might originate at the component lever or user level. The fact of the matter is that software intensive systems are here to stay. Major problems can ensue during both requirements gathering and integration phases without the adoption of software-specific system engineering processes. For both phases, traditional and agile methodologies exist for addressing such problems, but our thesis is that a better solution comprises a hybrid approach.

In fact, Bohem (2006) states that "a hybrid plan-driven/agile process architecture for developing Software Intensive System Of Systems (SISOS) are emerging. In order to keep SISOS developments from becoming destabilized from large amounts of change traffic, it is important to organize development into plan-driven increments." The Hybrid approach tends to help navigate requirements creep and systems integration issues not only by providing a flexibility

to accommodate change, but also by incorporating structure and plans for future increments. In this paper, we present an application of the hybrid approach to a large-scale data processing and analysis system. Design and implementation of this system took place over two years using a mix of traditional and agile system engineering approaches.

3. Large scale data processing and analysis system case study

This case study examines the development of a system that started with the high-level goal of taking in many disparate types of data, processing and analyzing them, and providing them to users for further analysis. While the high-level goals of the system were set before work on the system started, the plan was for definition of detailed goals and requirements was to occur iteratively as the system developed. The team developing the system performed most of this work. System development took two years. Three major milestones provided the team opportunities to work directly with end users with significant amounts of data available.

Throughout the two years of system development, the team applied lessons learned, goals, and identified user needs to determine the best system engineering approaches to use given the circumstances. The willingness of the team to adjust to changing circumstances allowed them to shift between different software system engineering approaches as appropriate for the phase of the program. The following sections describe the evolution of the system engineering approaches as the team adjusted to a series of lessons learned and the system matured. Table 1 defines terms used to describe this case study.

Table 1 – Definition of Terms.

Term	Definition
Team	The major participants in definition, development, and test of the system being described
Milestone	A major system test event designed to test system performance against real world examples
Spiral	A 1-2 month requirements, development, and test time period
Sprint	A 1-2 week requirements, development, and test time period
Component Team	The major participants in definition, development, and test of a specific component
Group	A set of people consisting of a subset of the team formed to execute a specific task

3.1. Milestone 1 Approach

The approach leading up to the first major system milestone was a waterfall-based approach (see Figure 1). Existing component capabilities and their potential applications drove the requirement development cycle for this milestone period. Once the team developed the requirements, there were several mini-releases prior to integration: a stub release, an initial feature release, and the final feature-complete milestone release. The first time any integration occurred was after the initial feature release.

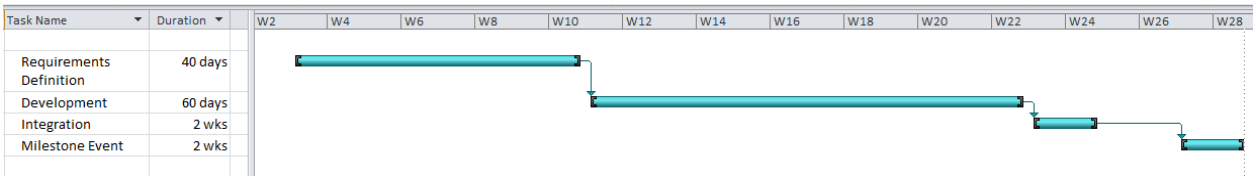

Figure 1 – Overall schedule leading up to Milestone 1.

When the team met for the first time at the project kickoff meeting (Figure 1), they divided into two groups to start the development of requirements immediately. Each group focused on a specific requirement types to allow the team

to begin specification of the system without a complete picture of the end state in mind. The first main requirement type focused on how to apply the system to improve users' ability to perform their current jobs. The second main requirement type focused on data flows through the system and component-to-component contracts.

As requirements were developed and the vision for the first milestone came together, component teams began design and then development of the system (Figure 1). Each component team made between one and three releases prior to commencement of formal integration. System infrastructure development occurred concurrently with component development – there were two releases made prior to commencement of integration. To enable testing during the development cycle, the team developed a surrogate infrastructure. This allowed the team to test components against a simple version of the infrastructure while the formal infrastructure was under development in parallel to the components. This enabled at least some component testing against some type of infrastructure prior to the formal infrastructure being available for integration.

Once integration and test started (Figure 1), the focus was on system verification. Integration took place over a few weeks, with representatives from each component working to get data flowing along major data paths. At the end of the integration period, the team demonstrated current functionality to a group of customer stakeholders. This demonstration also supported planning of the path to finish remaining work needed for successful completion of the milestone event.

At the milestone event (Figure 1), as well as the weeks leading up to it, the team continued to work on integrating and testing the planned functionality, ensuring that there was an assessment of what features were expected to work fully, what features worked partially, and what features were not yet available. During this time, the team trained a group of users. Those users began using the system for the milestone event itself. This was the first time any validation of the system occurred, however it was unstructured and often interrupted by underlying system issues.

3.2. Milestone 2 Approach

The approach leading up to the second major system milestone was an iterative waterfall-based approach (see Figure 2). There was an initial period of requirements development based on results from the first milestone, which developed overall plans for the system and individual component-level capabilities. After the initial requirements were set, there were three spirals, each with a set integration period at the end. During the integration for the second spiral, it became clear that there were too many issues with the system, such that completion of integration would not occur within the allotted time. Thus, requirements were reprioritized and postponed, and the integration process proceeded almost continuously until the milestone event.

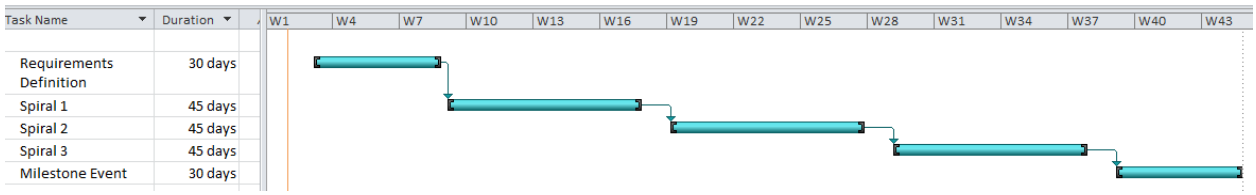

Figure 2 – Spiral schedule leading up to Milestone 2. Requirements were updated for each spiral.

Shortly after the execution of the first milestone, the team met to define the path to the second milestone (Figure 2). This exercise was based on a combination of overall system goals, reports from the first milestone detailing what worked, and feedback from the users at the first milestone outlining capabilities that needed to be updated, added, or removed for the system to be useful. Based on this meeting, the team began fixing known issues and developing requirements for new and updated capabilities.

For the requirements process, the team divided this phase of the program into three spirals (Figure 2). Development of the requirements for all spirals proceeded with varying degrees of specificity, with review and refinement of

requirements to take place at the completion of each spiral completed when additional information became available. As with the first milestone, the team performed concurrent work on all components as well as the system infrastructure, especially in the earlier spirals. The team continued to mitigate this risk by using a test infrastructure.

For integration during this time, the major difference was between the first spiral and the final two spirals. During the first spiral, there was a planned two-week integration period. When this integration period ended, the team resumed work on the second spiral with a list of remaining issues and untested features. During the second spiral, there was the same plan for a two-week integration period; however, the team made a decision to continue integration in parallel with the component development. This approach continued through the third spiral and until the second milestone event. While there was still more focus on verification than validation, the extended time period for integration allowed more people to use the system with a longer period of time to provide feedback to the system developers. This allowed the team to make small improvements based on observations received from integration.

At the milestone event, and immediately prior to it, we trained a set of users (different from those involved at the Milestone 1 event) on the use of the system. The team continued testing and verifying capabilities, but with a stricter gate for allowing system updates than in the prior milestone. This milestone provided another critical validation data point and the feedback received indicated a need for a paradigm shift for the team in order to ensure that user needs were met.

3.3. Milestone 3 Approach

After execution of Milestone 2, it became clear that additional interaction with users would be required for successful Milestone 3 execution. This led to an agile-based approach along with an additional focus on system validation earlier and more often than for prior milestones (see Figure 3). At this point in system development, the infrastructure was largely developed and there were no plans for major additional functionality. The team met at the beginning of this phase of the program to determine what changes to make based on feedback from users and other known issues from the second milestone event. The team laid those changes out across a schedule of five, two-week sprints. In addition to the original set of two-week sprints, the team added a series of one-week sprints after the original feature freeze when the originally planned formal milestone test changed format in a way that allowed the team to continue making system changes until closer to the event itself.

In addition to the change in scheduling, the team made always having a system available for use a high priority. During prior milestone periods, the system had not reached a sufficient level of stability that warranted dedication of hardware for validation – at the expense of hardware available for integration and test. By making such a change for this milestone, we allowed validation to occur on a release right after it went through integration and test.

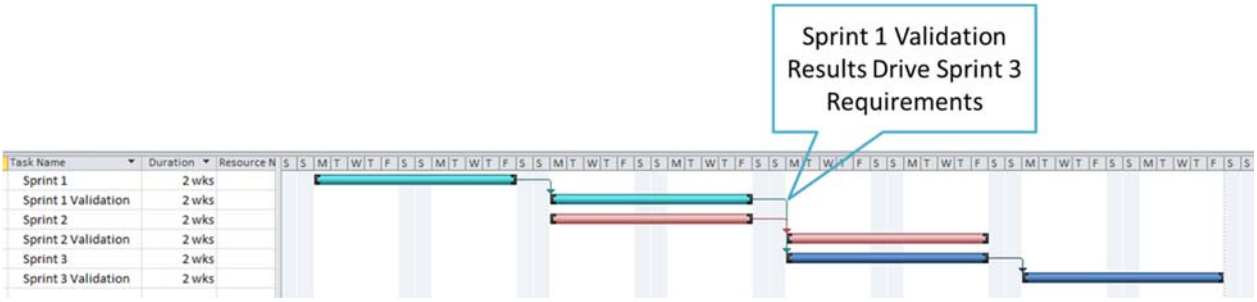

Figure 3 – A subset (three sprints) of the Milestone 3 schedule, including validation periods and how they overlap. The validation results from Sprint 1 feed directly into the Sprint 3 process, enabling rapid adjustment in response to user feedback.

During the two-week sprints, the first week was devoted to design and implementation of new features, with the second week focusing on component verification, integration, and system test. At the end of each week, we cut and transferred a release to our baseline system in support of a series of tests to verify correct configuration and the system

readiness for use. When we switched to one-week sprints, the schedule mirrored the 2nd week schedule from the two-week sprints. During that time period, there were always new features being rolled into the system, components just needed to schedule which sprint they wanted a feature to be part of. This rhythm was similar in many ways to the constant integration effort of Spiral 2 and 3 for the second milestone. The key differences were the more structured format for regular releases (leading up to the second milestone, intermediate releases addressed any known issues) along with the introduction of a workflow-centric testing methodology that focused more on testing the system as it would be used instead of verifying workflows only.

As soon as dedicated users began using the system for realistic tasks, there were additional requirements identified ranging from front-end focused to more fundamental changes. Upon identification, the team ranked requirements in terms of impact to the user, schedule feasibility, and technical risk. Based on this information, the team substituted the newly identified requirements for originally planned work based on relative priority. Because there was relatively little time to make changes and the infrastructure was already in place, there were a number of changes that were totally incompatible with the system design, or that required workarounds or partial solutions that helped in the short turn but would need to be re-examined as the system continued to grow.

Unlike the prior milestones, a series of events represented this milestone. There was one week to one month between each event, allowing feedback from the events to feed into the ongoing sprints. The group of users using the system during this time remained largely the same, so the team was able to establish a baseline and determine if system performance was improving, staying the same, or getting worse.

3.4. Results

With each milestone, the performance and utility of the system increased. From a user perspective, the system improved the most between Milestone 2 and 3. With the system infrastructure and components stable, the shift in focus to utility and usability allowed the team to make major changes relatively quickly. The hybrid approach allowed the system to mature and adapt, however additional planning and design work up front could have allowed incorporation of some of the larger changes that users asked for leading up to Milestone 3 into the system.

4. Lessons learned and future guidelines

The application of a hybrid approach to the design, development, verification, and validation of the system in this case study allowed the system to stabilize while allowing for new requirements later in development. By maintaining flexibility in paradigms, the team was able to adapt to the different needs and challenges of each milestone. In addition, the willingness of the team to add new artifacts to help guide system development enabled adaptability as focus shifted towards validating the system.

While the hybrid approach worked well over all for this system, there are a number of improvements needed to take fuller advantage of the different engineering paradigms. While there was some level of user involvement from the beginning of the program, there was no dedicated set of users until the third milestone. This made it difficult to establish a baseline and determine progress during and between milestones. In addition, the team did not take into account the impact an approach had relative to system stability or the impact an approach had on overall development and integration pace. This became clear during Milestone 3, when the team developed more new features in a given sprint than could realistically be tested. This led to a many long weekends and a reduction in stability of the baseline. To recover, the team added an integration only sprint where they allowed no new features and put tighter controls in place to normalize the workload across sprints.

In order to more fully take advantage of a hybrid approach, planning at the beginning of system design should occur to think through how much flexibility is necessary for the scope of the system and initial recommendations for when in the process to shift paradigms. This planning would lay out key recommended transition points, what the scope of the system is between each transition point, and how rapidly the system will have to adjust to change to meet deadlines. To do this effectively, teams should develop gates and metrics to determine what success looks like and when a new approach might be useful. Some metrics that the team applied in this area are: rate of integration relative

to development, rate of development relative to requirements and design, and system stability during integration. These metrics all look for bottlenecks in the process that are early indicators of the potential need for a modified or completely different approach, depending on the severity of the issue. This approach would allow teams to react both more quickly and with more forethought to changing needs.

Acknowledgements

The case study listed above represents the work of over 100 people, without whom the development of this system would not have been possible.

References

1. Boehm B. Some Future Trends and Implications for Systems and Software Engineering Processes. *Systems Engineering Journal* 2006;**9**:1-19.
2. Bijan Y, Yu J, Stracener J, Woods T. Systems Requirements Engineering – State of the Methodology. *Systems Engineering Journal* 2013;**16**:267-276.
3. Madni MA, Sievers M. System Integration: Key Perspectives, Experiences, and Challenges. *Systems Engineering Journal* 2006;**16**:000-000.