Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

Enablers and Inhibitors of Expediting Systems Engineering

Supannika Koolmanojwong* and Jo Ann Lane

Center for Systems and Software Engineering University of Southern California, Los Angles, USA 90089

Abstract

Rapid fielding or expedited systems development plays a major role in developing systems to provide a quick response to the organization. Instead of polishing and perfecting all requirements with possible delivery delays, these urgent needs programs must adapt themselves to have a life cycle that is driven by a "Faster, Faster and Faster" concept. Surprisingly, schedule has become more important than cost. Organizations have to fight with the "time to market", "respond to competitor's threats", and "urgent needs". On one hand, project managers will not only have to find a way to make things work through the expedited lane, but also have to understand what factors are the accelerators and what are hindrances to the system development schedules. On the other hand, they have to ensure that these expedited processes will not lead to technical debt, especially in the areas of future flexibility, degradation of existing capabilities, or rework. This paper will report about the enablers and inhibitors and their estimated impact levels of three types of the expedited systems and software development a) New Single System b) Existing Single System and c) System of Systems.

Selection and/or peer-review under responsibility of Georgia Institute of Technology.

Keywords: Expedited Systems Engineering; Technical debt; Inhibitors; Enablers

1. Motivation and Context

In recent years, much work has been done to better understand engineering capabilities within an SoS and how this differs from single system engineering. With this understanding comes the realization that many systems belong to one or more SoSs and contribute to a variety of SoS capabilities and that the trade space for new capabilities can be quite large. Most of the time, SoS engineers focus on capability approaches that will deliver the desired capability quickly and with the least amount of effort. However, this approach, if not carefully managed, can lead to technical debt, especially in the areas of future flexibility, degradation of existing capabilities, or rework. The goal of this research is to a) understand if there are "shortcuts" that can be taken to reduce effort and schedule and b) identify methods to quantify the savings in effort and schedule. Current research is working to identify ways to expedite systems engineering without increasing technical debt. This paper will report on the techniques and approaches to quickly engineering critical capabilities in developing a new single system, an existing single system,

* Corresponding author. Tel.: +1-213740-5703; fax: +1-213-740-4927.
E-mail address: koolmano@usc.edu.

and a system of systems (SoS) environment. The paper is organized into 5 sections. Following this introduction, section 2 provides background information of expedited systems engineering, Valuing Flexibility, and Technical Debt. Section 3 describes the data collection process and top 25 enablers and inhibitors of expedited systems engineering of 3 different project types. Section 4 discusses data analysis results such as the similarities and the uniqueness of each project type. Lastly, section 5 provides conclusion and discusses future work.

2. Background and Related Work

Three major pillars in the system development tradespace are expedited systems engineering, valuing flexibility, and technical debt. These three pillars are similar to the cost-schedule-quality triangle in a sense that when you adjust one factor, the other two will be affected too. To develop a system, one cannot just blindly consider expediting system development alone without thinking about the other two factors. To have a practical and sustainable system, one need to make sure that the system is available when it is needed, in addition, one has to consider how to design a system that is flexible and maintainable, and how to design a system that will not create future debt or future problems. This section describes three key factors that one has to consider when expediting a system development.

2.1. Expediting Systems Engineering.

An important aspect of the system acquisition process is understanding how long it will take to provide the new needed system or system capability. Considerable work has been done in this area to better understand system/capability development schedules, with considerable contributions from parametric cost models for software-intensive systems that are based on historical data. However, many government organizations, including the Government Accountability Office (GAO), have pointed out how much money is wasted on unsuccessful system development programs that often fail to provide the necessary capabilities or if they do provide the capabilities, they are late and cost much more than expected [1]. As a result, the Department of Defense has realized that it is often no longer cost-effective to develop new systems (or system capabilities) using traditional processes and is encouraging researchers to find ways to expedite systems engineering and development. Currently, the engineering community is embracing agile and lean processes as well as system of systems engineering to provide new capabilities through the integration and enhancement of existing systems as ways to rapidly respond to immediate needs. This, though, can lead to single-point solutions that are not flexible enough to meet the next set of needs or incur technical debt due to extensive rework and maintenance costs.

A few of the larger, more success system development organizations are investing infrastructure and product lines [2,3] so that they can respond quickly when new opportunities or world events occur that require immediate solutions. These suppliers realize a return on their investments when they have the needed flexibility built into their system infrastructure and have minimal technical debt due to the quality of the infrastructure core that is built in from the start.

2.2. Valuing Flexibility.

Many organizations treat system development and system life-cycle support as different projects with different budgets. This usage may cause suboptimal system architectures and designs that minimize development costs, but result in inflexible, hard-to-modify systems and higher overall costs for the system's owners [4]. The goal of "flexibility" is to focus on developing a robust foundation for the system that will provide the ability to easily modify existing system capabilities as well as expand system capabilities to meet future needs or allow the system to easily interconnect with other systems to support cross cutting capabilities in systems of systems. Without a certain level of flexibility, systems can quickly become obsolete as technology and stakeholder/mission needs change. However, if too much effort is spent on developing the most flexible systems, it may be at the expense of delivery expediency and simplicity of the system core, resulting in longer development schedules for the first incarnation of the system and poorer operational performance due to the need to sacrifice single capability performance for the ability to perform multiple capabilities (or perform a single capability in multiple environments)

2.3. Technical Debt

Recent research by Steven McConnell [5], Practical Systems and Software Measurement (http://www.psmsc.com/) affiliates, and others are focusing on a concept referred to as "technical debt" and ways to better manage it in the development of systems and software. Ward Cunningham [6] first coined the term and further explained it in his YouTube video [7]. Technical debt refers to delayed technical work or rework that is incurred when shortcuts are taken. Some technical debt is reasonable given time-to-market or other urgent constraints. But over the long term, if the system is to be sustained, the technical debt must be paid back, often with interest (i.e., it is more expensive to fix than it is to do it right the first time). Deferred long-term technical debt can result in fragile, error-prone systems that take excessive time to modify, with more time spent on maintaining existing capabilities than adding or improving capabilities

3. Data Collection

Schedule factor has become more important than cost factor. Organizations have to fight with the "time to market", "respond to competitor's threats", or "urgent needs" [9]. Hence, project managers have to find a way to make things work through the expedited lane. During the 16th Annual Practical Software and Systems Measurement (PSM) Users' Group Conference [10], we organized a workshop called *Expediting Systems Engineering within an SoS* to explore techniques and approaches to quickly engineering critical capabilities. The participants were from commercial, military/defense, and academic sectors. During the workshop, the participants identified best practices or techniques that provide means or opportunities to enable successful expedited system development. The workshop captured 26 to 35 enablers and inhibitors for new single system development, existing single system development, and SoS capability development. After the enabler and inhibitor factors were identified for each project type, each attendee rated each factor "high", "medium" or "low" to indicate the level of impact each would have with respect to expediting in the respective category. Then each rating was weighted and the scores from each attendee are combined to form a rank-ordered list of factors. For the factors where the weighted scores were identical, standard deviation was used as a tie-breaker. The following sections describe in more detail about the identified enabler and inhibitor factors for each of the three categories of systems: new single system, existing single system, and SoS.

3.1. Expediting a New Single System

To develop a new system starting with a clean sheet of paper (e.g., a Greenfield development), the development team has considerable freedom in their choices of architecture, non-developmental items, and engineering processes. Table 1 lists in rank order, the expediting enablers and inhibitors associated with new single system development. With the growing use of agile and lean engineering processes, one might think that this would be the "silver bullet" for expedited engineering. Surprisingly this was not ranked as the highest impact enabler. In addition, the Greenfield development may not speed through "green lights" as quickly as one would like. As shown in the second column in Table 1, there are several factors that often inhibit progress and create red lights that can slow it down.

First, let's discuss the green lights. Rapid Prototyping is rated as the top enabler. Rapid Prototyping can be used as project feasibility evidence or as a tool to show progress, to test a proof-of-concept, to gather feedback, to create shared knowledge, and to acquire commitment from stakeholders. If done well, an early rapid prototype of high-risk components can evolve into the actual system component, minimizing both risk and late rework, thereby compressing the development schedule. Having an integration/experimentation lab, with target hardware and simulators early in the development process , can be used by the developers to converge early upon a feasible design as well as to provide a mechanism to solicit stakeholder inputs and to provide feedback to the stakeholders to show that the development is on the right track. Requirements flexibility refers to the ability to adjust or negotiate requirements if there is an evidence that the original requirements are unaffordable, infeasible, or too risky. Incremental test and incremental delivery are very similar, but they are not the same. Incremental test provides feedback based on its test results from its predefined test cases. Incremental delivery provides stakeholders' feedback based on the launched or delivered product. Incremental delivery of a single system is easier to achieve

compared to the near-impossible incremental delivery of a system of systems. Next, the 9th enabler, one of the wastes in lean development is context switching [8]. If you have to work on more than one project at a time, each time you switch the context, you will have to relearn and re-acquaintance with your new context. This task switching is really a big waste which will slow down your task and prohibit you from expediting the system development.

Inhibitors or hindrances of a new single system development are mostly the same old faces you may see in the classic risk of disasters' list [11,12]. Requirements creep or requirements volatility refers to modification, addition, or deletion of the requirements during the development life cycle. The later the volatility is introduced, the longer the delay it will cause to the system. Unprecedentedness challenges system engineers both in technical and non-technical perspectives. When a project is a one-of-a-kind or an avant-garde project, it is difficult for all stakeholders to check whether the product is developed correctly or is good enough. In addition, extra effort needs to be spent on research, analysis of alternatives, trial-and-error, and coming up with learning curve. Delayed authority to proceed with fixed milestone is one of a surprised, but a common inhibitor. If your project has started, but you are waiting for the authority to work on the project, it will not only halt the project, but it will also cause cascading delays in all project activities. Unfortunately, it can be worse when you know that the deadline is approaching.

Table 1. Top 25 Enablers and Inhibitors for Expediting a New Single System

Rank	***Enablers***	***Inhibitors***
1	Rapid Prototyping	Requirements Volatility
2	Target hardware lab (experimentation / test lab) / test like you fly & simulation	Unprecedentedness
3	Customer /tech requirements flexibility	Delayed authority to proceed/start with fixed milestone
4	Incremental test and feedback	Infeasible schedule/staffing profile
5	Incremental Delivery & feedback	Lack of Domain Experience
6	Decision making authority	Technology Volatility
7	Best people / personnel capability	High numbers of external interfaces
8	Agile/lean approach	Vague Requirements
9	Less context switching when doing multiple projects	Under average people / Personnel Capability
10	Tools and automation	Technology Immaturity
11	Common standard, interface	Conflicting Stakeholders
12	Flexible / tailorable rules	Lack of decision making authority
13	Model-based engineering	large number of subcontractors / stakeholders
14	Building the common architecture/foundation	Lack of development infrastructure
15	Reusing assets	Rules and Regulations
16	Risk Management	Sequential Development
17	Team cohesion	Classification / sensitivity
18	Business process reengineering / process streamlining	Fear to protest the contract award resulting in poor requirements
19	COTS	Personnel Turnover
20	Overnight build	Bad RFP
21	Development process tailoring/adjustment	Developers / subcontractors not co-located
22	Feasibility Evidence, Milestone review	Lack of program empowerment
23	Mature configuration management	Design for reuse
24	Colocation of hw & sw engineers	-ilities standard
25	Crowdsourcing	Contracting limitation

3.2. Expediting an Existing Single System

System maintenance or enhancement is used to refer to the work of modifying, enhancing, and providing cost-

effective support to the existing software [13]. System enhancement includes error corrections, functional enhancements, technical renovations, and reengineering. To enhance or extend from an existing system, some of the main constraints are current system understanding and current system architecture and foundation. The development activities could be either a major or minor modification of a product after delivery. Table 2 lists the enablers and inhibitors for expediting an existing single system.

Common standard or interface is one of the important enablers for system maintenance. If the existing system uses common interface prototype, it can save time in architecting the extensions. However, if the existing system is using the APIs or connectors that are outdated or uncommon, the development team needs to find out the possibility of reconnecting the new system to the existing system. Next, domain knowledge or knowledge about the existing system is also a key to expedited systems engineering. On the opposite side, embedding poor quality of system is one of the obvious inhibitors, which highly overlaps with Technical Debt and Back-propagation. Inheriting bad system such as bad architectures, hardcoded modules, spaghetti code require system engineers to payback the debt or even worse redo or reengineer the existing system. Unaware or having zero knowledge or lack of system understanding reflect big risks on various key project failures such as personnel shortfall, architecture complexity, unreal schedule, and etc. Technology volatility and technology immaturity are also big inhibitors. Whether it is a turning-brownfield-into-greenfield or continuation of greenfield, technology selection becomes constraints that the development team has to overcome and stabilize as soon as it is supported by the feasibility evidence.

Table 2. Top 25 Enablers and Inhibitors for Expediting an Existing Single System

Rank	***Enablers***	***Inhibitors***
1	Target hardware lab (experimentation / test lab) / test like you fly & simulation	Requirements Volatility
2	Incremental test and feedback	High numbers of external interfaces
3	Incremental Delivery & feedback	Unprecedentedness
4	Flexible / tailorable rules	Vague Requirements
5	Agile/lean approach	Embedded poor quality software
6	Rapid Prototyping	Conflicting Stakeholders
7	Common standard, interface	Delayed authority to proceed/start with fixed milestone
8	Customer /tech requirements flexibility	Infeasible schedule/staffing profile
9	Domain knowledge	Technical debt
10	Understanding of the existing system and interfaces	Interoperability / compatibility
11	Best people / personnel capability	large number of subcontractors / stakeholders
12	Less context switching when doing multiple projects	Backpropagation
13	Tools and automation	Lack of understanding of the existing system and interfaces
14	Reusing assets	Under average people / Personnel Capability
15	Team cohesion	Lack of Domain Experience
16	Feasibility Evidence, Milestone review	Technology Volatility
17	Model-based engineering	Technology Immaturity
18	Colocation of hw & sw engineers	Classification / sensitivity
19	Development process tailoring/adjustment	Multiple operational sites with different configuration / platform / OS
20	Risk Management	Outdated / stovepipe technology
21	Decision making authority	Personnel Turnover
22	COTS	Lack of decision making authority
23	Business process reengineering / process streamlining	Architecture constraint / heritage

24	Outsourcing / surge support	Rules and Regulations
25	Mature configuration management	Bad documentation

3.3. Expediting a System of Systems

System of Systems Engineering (SoSE) is considered to be a multidisciplinary area and it includes management and organizational aspects as well as technical aspects of systems engineering at the System of Systems level [14]. As these systems become larger and larger, the constituent-systems are operationally independent and managerially independent and it requires extra effort and schedule to coordinate and synchronize among several units. Table 3 shows the list of enablers and inhibitors ordered by their impact level.

Reuse is a possible shortcut with caveat for expedition. If those assets are designed for reuse, it will really speed up the development process. However, if you are reusing bad assets, you will have to spend avoidable effort to fix the defects and to figure out the solution, eventually you may end up with developing them from scratch. The 10th enabler, compared to a single system or an existing system, team cohesion seems to affect the most in SoS since it requires high collaboration between individuals and teams. Synchronization and Stabilization will show various SoS teams their feasibility evidence that their progress are approaching towards their goals. Having a common architecture and foundation is a good stepping stone to ensure that they have an established and strong backbone for the development. Lastly the 18th enabler, it is impossible for SoS teams to collect their information by using tacit knowledge of team members, hence Constituent Documentation is indeed a great enabler.

Unique inhibitors of SoS such as high number of external interfaces, infeasible schedule, large number of subcontractors, lack of communication between teams, poor/ unknown heritage/pedigree, conflicting stakeholders, reflects the nature of SoS diseconomy of scale.

Table 3. Enablers and Inhibitors for Expediting a System of Systems

Rank	***Enablers***	***Inhibitors***
1	Customer /tech requirements flexibility	Lack of Interoperability
2	Rapid Prototyping	Lack of / incompatible standard & protocol
3	Target hardware lab (experimentation / test lab) / test like you fly & simulation	Requirements Volatility
4	Incremental test and feedback	Unprecedentedness
5	Common standard and protocol	High numbers of external interfaces
6	Reusing assets	Infeasible schedule/staffing profile
7	Tools and automation	Inability to test across systems
8	Common standard, interface	Delayed authority to proceed/start with fixed milestone
9	Best people / Personnel Capability	Lack of Domain Experience
10	Team cohesion	Technology Volatility
11	Synchronization and Stabilization	Technology Immaturity
12	flexible / tailorable rules	Vague Requirements
13	Model-based engineering	Large number of subcontractors / stakeholders
14	Building the common architecture/foundation	Lack of development infrastructure
15	Agile/lean approach	Lack of communication between teams
16	Incremental Delivery & feedback	Classification / sensitivity
17	COTS	Poor / unknown heritage/pedigree
18	Constituent Documentation	Bad RFP

19	Decision making authority	Personnel Turnover
20	Development process tailoring/adjustment	Under average people / Personnel Capability
21	Risk Management	Conflicting Stakeholders
22	Less context switching when doing multiple projects	Poor extendibility
23	Colocation of hw & sw engineers at SOS level	Sequential Development
24	Overnight build	Overspecified requirements
25	Business process reengineering / process streamlining	Lack of constituent expert

4. Observation and Discussion

This section leverages the list of enablers and inhibitors by comparing the similar components and unique factors of the three systems.

4.1. Considerable overlap between the enablers and inhibitors across three kinds of systems

Table 4 shows top 10 inhibitors of a new single system, an existing single system, and a system of systems. As highlighted in bold and italic, these inhibitors are similar across all three systems. For example, Requirements Volatility, Unprecendentedness, Delayed authority to proceed/start with fixed milestone, Infeasible schedule/staffing profile, and High numbers of external interfaces, these factors are common inhibitors that can delay any project. On the other hand, each type of system has its own specific inhibitors as highlighted in underlined texts. For a new single system, Under-average people and Technology Immaturity are the major hindrances that may not matter most in the other two systems. For an existing single system, it is pretty clear that if the existing system has poor quality, it is a major setback for the system enhancement, which leads to the next unique inhibitor, technical debt. Inheriting a debt or substandard system, the system maintenance team needs to spend extra effort and schedule in refactoring or restructuring the current system and in turn prevent them from accelerating their project development. Lastly, for a system of systems, distinctive inhibitors are lack of interoperability and inability to test across systems. These two inhibitors really reflect the key basic concept of SoS. With so many systems that an SoS has to work with, if they don't have a good foundation, each system will gradually grow apart and will not only stop them from moving forward, but also force them to step back to solve the basic foundation problem. For an SoS, at various points in project development, the teams need to stabilize and synchronize their works, which will provide a close loop feedback to the teams. If there is no one at check points or they are not able to check anything at milestones, it shows that the teams are walking blindly. As a result, decision makings will be delayed or uncommitable.

Table 4. Top 10 Inhibitors

A New Single System	**An Existing Single System**	**A System of Systems**
Requirements Volatility	***Requirements Volatility***	Lack of Interoperability
Unprecedentedness	***High numbers of external interfaces***	Lack of / incompatible standard & protocol
Delayed authority to proceed/start with fixed milestone	***Unprecedentedness***	***Requirements Volatility***
Infeasible schedule/staffing profile	Vague Requirements	***Unprecedentedness***
Lack of Domain Experience	Embedded poor quality software	***High numbers of external interfaces***
Technology Volatility	Conflicting Stakeholders	***Infeasible schedule/staffing profile***
High numbers of external interfaces	***Delayed authority to proceed/start with fixed milestone***	Inability to test across systems
Vague Requirements	***Infeasible schedule/staffing profile***	***Delayed authority to proceed/start with fixed milestone***
Under average people / Personnel Capability	Technical debt	Lack of Domain Experience

Technology Immaturity	Interoperability / compatibility	Technology Volatility

Similarly, there are also major overlaps in expediting enablers as reported in Table 5. Rapid Prototyping, Having a target lab, increment test and feedback, customer/ tech requirements flexibility are among common best practices that one can use to expedite the system development. Similar to the inhibitors, there are several enablers that contribute more in a particular system, but not at the others. An SoS requires extremely high collaboration between teams, hence it really needs a good team cohesion. For an existing single system, to know and understand the system that you have to enhance is a major accelerator otherwise the team needs to study the structure, architecture, its operational concepts and it usually takes time to come up with such a learning curve. For a new single system, since there is no problem of extending current system or extremely high number of parties to deal with, the team can really move fast if they have authority to make decision and to commit with decision and be accountable to their responsibilities.

Table 5. Top 10 Enablers

A New Single System	**An Existing Single System**	**A System of Systems**
Rapid Prototyping	***Target hardware lab / test like you fly & simulation***	***Customer /tech requirements flexibility***
Target hardware lab / test like you fly & simulation	***Incremental test and feedback***	***Rapid Prototyping***
Customer /tech requirements flexibility	Incremental Delivery & feedback	***Target hardware lab / test like you fly & simulation***
Incremental test and feedback	Flexible / Tailorable rules	***Incremental test and feedback***
Incremental Delivery & feedback	Agile/lean approach	Common standard and protocol
Decision making authority	***Rapid Prototyping***	Reusing assets
Best people / personnel capability	Common standard, interface	Tools and automation
Agile/lean approach	***Customer /tech requirements flexibility***	Common standard, interface
Less context switching when doing multiple projects	Domain knowledge	Best people / Personnel Capability
Tools and automation	Understanding of the existing system and interfaces	Team cohesion

4.2. Many inhibitors are the flip side of its corresponding enablers.

Table 1 – 5 show that many of the inhibitors are a result of lacking its corresponding enabler factors. For example, Incremental Test and Feedback versus Inability to test across systems, and best people versus under average people. On the other hand, requirements flexibility versus requirements volatility reflects the grey line between adjustable and evolving requirements. Common standard versus flexible rules is another example that system engineers need to find the balance or sweet spot in expediting the system development.

5. Conclusion and Future Work

We noticed that there was considerable overlap of the enablers and inhibitors across new system/existing system/System of Systems, but that the impact ratings tended to differ. On the other hand, many inhibitors are the "flip side" of a corresponding enabler. Our next steps will focus on understanding the overlapping factors, gathering more data from different domains, and investigating how to balance Expediting Systems Engineering, Technical Debt, and Flexibility. The ultimate goal is to find ways to minimize engineering effort and schedule without incurring unacceptable levels of technical debt and to improve cost models to better inform expediting alternatives.

References

1. U.S. General Accountability Office, “Defense Acquisitions: Assessments of Selected Major Weapon Programs,” March 2012.
2. Engineering Design Forum. 2012. http://www.phoenix-int.com/forum_2012/index.html.
3. Royce, W. 2004. Software Project Management: A Unified Framework. Addison-Wesley.
4. B. Boehm, J.A. Lane, R. Madachy,Total Ownership Cost Models for Valuing System Flexibility, Proceedings,, CSER 2011, March 2011.
5. McConnell, S. 2008. Best Practices White Paper: Managing Technical Debt. http://www.construx.com/Page.aspx?cid=2801.
6. Cunningham, W. 1992. The WyCash portfolio management system, Experience Report. Oopsla’92.
7. Cunningham, W. 2009. Debt metaphor. http://www.youtube.com/watch?v=pqeJFYwnkjE.
8. Poppendieck, Mary and Tom Poppendieck. Leading Lean Software Development, Addison Wesley, 2009
9. S. Koolmanojwong and B. Boehm, "The Incremental Commitment Model Process Patterns for Rapid-Fielding Projects" In Proceedings of International Conference on Software Process, Paderborn, Germany, July 2010
10. The 16th Annual PSM Users’ Group Meetings and Workshops, 30 July – 3 August 2012 , Portsmouth, VA, http://www.psmsc.com/events.asp, accessed August 7, 2012
11. B. Boehm, Software Risk Management: Principles and Practices, IEEE Software, Volume 8, Issue 1, January 1991
12. S. McConnell, Avoiding Classic Mistakes, IEEE Software, Volume 13, Issue 5, September 1996
13. V. Nguyen, B. Boehm, P. Danphitsanuphan, Assessing and Estimating Corrective, Enhancive, and Reductive Maintenance Tasks: A Controlled Experiment, Proceedings of 16th Asia-Pacific Software Engineering Conference (APSEC 2009), December 2009
14. J.A. Lane, "Imapacts of System of System Management Strategies on System of System Capability Engineering Effort," PhD Dissertation, Department of Industrial and Systems Engineering, University of Southern California, May 2009