Is Agile too Fragile for Space-Based Systems Engineering?

Scott E. Carpenter

North Carolina State University, Department of Computer Science

(919) 413-5083

carpen@ncsu.edu

For the partial requirements of

CSC 510-601, Spring 2013

Dr. Aldo Dagnino
## Contents

Abstract .......................................................................................................................... 4

1. Introduction ............................................................................................................... 6
   1.1. Objective ............................................................................................................. 6
   1.2. Background and Motivation ............................................................................. 6
   1.3. Organization of This Paper ............................................................................. 9

2. Literature Review .................................................................................................... 10
   2.1. Mission-Critical Systems .................................................................................. 10
      2.1.2. Traditional Methodologies ..................................................................... 16
   2.2. Space-Based Systems Development ................................................................. 22
      2.2.1. Flight Systems Development .................................................................... 25
      2.2.2. Critical Systems Development and Formal Methodologies .................... 30
      2.2.3. The Role of Architecture ......................................................................... 31
      2.2.4. Commercial Off-the-Shelf (COTS) ............................................................ 33
      2.2.5. Mission-Critical Failures – Root Cause Analysis ..................................... 34
      2.2.6. NASA’s Faster, Better, Cheaper (FBC) ....................................................... 40
      2.2.7. The Importance of Social Context .............................................................. 42
      2.2.8. Drivers for Development Improvement ....................................................... 46
      2.2.9. Systems Development at SpaceX ............................................................... 50
   2.3. Agile ..................................................................................................................... 57
      2.3.1. Agile Development as Compared to Traditional Methodologies ............. 62
      2.3.2. Combining Agile and CMMI: Destined for Marriage, or The Odd Couple? 64
      2.3.3. Agile Methods ............................................................................................ 67
      2.3.4. Rework ......................................................................................................... 70
      2.3.5. Adopting Agile Methods for Mission-Critical Systems Development ...... 71

3. Discussion and Analysis ........................................................................................... 79

4. Relevant Work ......................................................................................................... 89

5. Future Work ............................................................................................................. 91

6. Conclusion ............................................................................................................... 96

References .................................................................................................................... 98

Biography .................................................................................................................... 102
Figures

Figure 1.1 – A humorous view of visualizing success (Credit: www.businessinsider.com) ...................... 6
Figure 2.1 – CMMI for Development v 1.3 process areas (Credit: www.methodpark.com) ..................... 19
Figure 2.2 – NASA’s systems engineering engine [1] ........................................................................ 20
Figure 2.3 – Global space activities, 2007 [2] ................................................................................. 23
Figure 2.4 – Moore’s Law for processor selection. From [3] ................................................................. 27
Figure 2.5 – Space Shuttle Challenger Explosion (Credit: www.aerospaceweb.com) .......................... 35
Figure 2.6 – The Hubble mirror flaw (Credit: hubblesite.org) ............................................................. 36
Figure 2.7 – NOAA-N Prime failure (Credit: www.spaceref.com) ...................................................... 39
Figure 2.8 – RLV vision (Credit: www.styleofspeed.com) ................................................................. 46
Figure 2.9 – SpaceX launch and control center. [4] ........................................................................... 51
Figure 2.10 – Launch of SpaceX Falcon 1 [4] .................................................................................... 52
Figure 2.11 – SpaceX CEO Elon Musk with the company’s Falcon 9 rocket (Credit: SpaceX) ........... 54
Figure 2.12 – The Scrum cycle (Credit: codeproject.com) ................................................................. 69
Figure 2.13 – Sidky’s three-stage process on agile process suitability [5] ............................................. 76

Tables

Table 2.1 – Examples of critical system failures. From [6] ................................................................. 38
Table 2.2 – Differences between traditional and agile development [7] ................................................ 62
Table 2.3 – Fairley’s iterative rework taxonomy [8] .......................................................................... 71
Table 2.4 – Prescriptive characteristics of various agile practices [9] ................................................. 72
Table 5.1 – Corporation ABC: suitability of agile practices to formal methodology XYZ ................. 94
Abstract

Mission critical systems engineering is especially complex, driven by high safety and performance requirements. Space exploration is one such development domain in which mission criticality defines success. There is often only one chance for mission success, and failure can be fatal.

While the industry’s development organizations continue to focus on safety and performance, key drivers, such as budgetary cuts and industry globalization, are leading them to consider different ways of achieving high-quality results while balancing cost-constraints.

Agile practices are relatively new and produce much interest within all aspects of systems engineering. While there is some limited research in the applicability of agile practices within mission-critical systems engineering, there is not much supporting data with respect to specific applicability of practices within the space industry.

Despite the application of various formal methodologies and engineering standards within the space exploration industry, failures occur. Although proximate causes are often technical, root causes of failure are usually mismanagement of the social context, leading to human error.

Thus, while needing to maintain safety standards, the space industry is driven to consider alternate methodologies by which to achieve success and limit failure. This makes it important to study the existing formal methodologies and engineering standards which are applied within the industry, and to assess whether or not agile methodologies can be usefully applied within the industry.
While some agile practices may be discarded outright, others seem quite useful and should be given consideration. A hybrid approach to combining agile practices with existing formal methodologies and engineering standards should be practiced. This presents several opportunities for future work: case studies should be performed, mappings of specific formal processes to agile practices should be completed, and database of lessons learned should be established, from which recommendations could be made that benefit the industry as a whole.
1. Introduction

1.1. Objective

Objective: I am writing about project management methodologies applied to the development of mission-critical systems because I am trying to show you why methodologies differ between mission-critical and non-mission critical systems development in order to explain to you how agile practices may be helpful or detrimental to the development of mission-critical systems.

1.2. Background and Motivation

Every mission-critical system is produced for one purpose and intended for a specific, limited time of use. As such, there are only two possible outcomes: A) success, and B) failure. The former is preferred.

Yet, given the choices, the path to success is not always as obvious as we wish it to be, as visualized in Figure 1.1, and this befuddlement becomes even more evident when analyzing success and failure of mission-critical systems.

Figure 1.1 – A humorous view of visualizing success (Credit: www.businessinsider.com)
The uniqueness of the mission’s purpose and the objective of success over failure are key characteristics of mission-critical systems which set them apart from other types of systems. As such, the methods by which critical-systems are developed are very different from traditional methodologies applied to more common systems. Spacecraft are but one example of mission-critical systems.

The realization of successful space missions is difficult to achieve. There are many physical complexities which must be overcome, and there are many things which can go wrong within and between the varying sub-systems. “Space is risky. The phrase ‘It ain’t rocket science’ implies that rocket science is pretty hard—and it is. Only a few countries in the world have gotten anything at all into orbit [4].”

Space-based systems development remains a comparably young industry filled with abundant examples of failure [6] [10]. Even today, failure remains a common result of space exploration attempts. Yet, it remains an area of active research to determine whether or not it is possible to improve success rates through better processes and development methodologies [11] [9]. It is thus interesting to study the development methodologies that are practiced within the space industry, to consider the drivers which will shape the industry’s future, and to contemplate the applicability of disparate software engineering techniques, such as agile, within the domain.

Agile was popularized with the Agile Manifesto [12] [13]. Especially within the realm of software development, various agile practices have been applied, with varying levels of success. However, not all agile practices are well-suited for all aspects of development of all types of systems. The purpose of this paper is therefore best expressed by the following question: How
do we determine which agile practices, if any, are desirable and suitable for space-based systems development?

Specifically, this paper shall focus on a review and analysis of development methodologies and practices that have been used within government agencies such as NASA and private endeavors such as SpaceX. A review of how practices have evolved over time shall be explored, with particular interest in the arrival of agile development and the potential suitability for the development of critical spacecraft systems by NASA, SpaceX and other, similar emerging companies.

While many agile practices are indeed used throughout many different software development organizations, information about the application of agile within space development organizations is limited, at best. This is primarily because mission-critical systems are characterized as being large, complex, requiring long development cycles, and having specific desirable system performance qualities which affect the suitability of agile practices to such development. Furthermore, when considering the applicability of agile practices to systems development, care must be taken in determining if adopting an agile practice conflicts with any system/development characteristic or precludes any desired system quality [5]. While this does not obviate the use of agile methods in critical-systems development, it does point out that careful selection of agile practices must be done when considering their applicability to any development. This is especially true within space-systems development, and in fact a very deliberate process must be applied when determine which practices an organization wishes to consider.
This research is relevant to the space industry because there are many budget, cultural, and technical drivers which are reshaping the space development industry. SpaceX’s relatively recent arrival into the domain of space exploration and their string of successes brings further attention to the need to study how the industry is shifting in response to external pressures.

A comprehensive study of the applicability of agile and its practices within space technologies has not been found. As such, further research is needed to explore in depth and characterize agile practices within the space industry. Case studies are needed to document those organizations which have tried agile methods, successfully or otherwise. Research is needed to assist organizations wishing to utilize agile practices in the proper and careful selection of appropriate techniques.

1.3. Organization of This Paper

This paper is organized into five major sections as follows: Literature Review; Discussion; Relevance to the Industry; Future Work; and Conclusions.

The Literature Review is broken into three main sections: Mission-Critical Systems, Space-Based Systems Development, and Agile. In the first section, common methodologies used by organizations to develop mission-critical systems are explored in order to show how development is different between mission-critical and non-mission critical systems. In the next section on Space-Based Systems Development, systems development of space-based systems are described so that similar and differing approaches to past development of mission-critical space-based systems can be shown, especially within organizations such as NASA, the military, and SpaceX. Current issues and the future drivers which will influences changes to development methodologies within the industry are also covered. Lastly, agile methodologies are reviewed
with emphasis on how to decide which ones may be applied to the systems development of space-based systems. Not all agile methodologies can be applied to the development of critical systems and that care must be taken when deciding which methodologies to apply effectively.

The Literature Review is rather lengthy because this research topic covers breadth of general systems development principles with additional specificity to space-based development; examples of space development practices; motivations for changes to such practices; and agile development principles. There is a lot of material to be covered. Each section is therefore organized to lead the reader with an **Objective** at the beginning of each major section, and a **Summary** at the end of each sub-section. The knowledgeable reader may therefore skim the Literature Review as desired.

2. Literature Review

2.1. Mission-Critical Systems

*Objective:* *Common methodologies used by organizations to develop mission-critical systems are explored in order to show how development is different between mission-critical and non-mission critical systems.*

Mission-critical systems development has as its primary goal the creation and delivery of a systems product which must operate flawlessly on its first and every subsequent deployment. Furthermore, in the case of many aerospace projects, there is only one deployment, and so the performance of the system is truly the most important requirement. In the event of failure to deliver a critical payload into space, new rockets can be assembled and replacement satellites and other payloads can be produced, at a cost. However, when system failure results in the loss
of human life, it becomes impossible to get a second chance, and the cost of such failures cannot be quantified.

It is therefore important to study the means by which such critical systems are developed, and the look at drivers which impact the methodologies which are practiced and how they have evolved over time. First, however, we need to set some context and clarify what is meant by mission critical systems development, and the project management methodologies employed in support of these efforts. Presented below are some basic definitions [14] which are applicable throughout the remainder of this paper:

**System**: A combination of parts which form a complex or unitary whole, in which the interrelated components work towards a common objective.

**Systems engineering**: An engineering discipline with the goal of creating and executing an interdisciplinary process which ensures that the customer and stakeholder’s needs are satisfied in a high-quality, trustworthy, cost efficient, and schedule compliant manner throughout a system’s entire life cycle.

**Mission critical**: Refers to any factor that is crucial to the successful completion of an entire project.

**Safety-critical system**: A computer, electronic or electromechanical system whose failure may cause injury or death to human beings.

Numerous volumes already exist on the general study of systems engineering approaches to systems development, including complex products composed of interrelated components. This gives motivation to further narrow the focus to that of mission-critical and safety-critical
systems. To facilitate this, key characteristics must be identified which help separate the specific aspects of critical systems development from that of more general software engineering practices.

**Summary**

- Mission-critical systems development has as its primary goal the creation and delivery of a systems product which must operate flawlessly on its first and every subsequent deployment.
- Key definitions:
  - *System*: A combination of parts which form a complex or unitary whole, in which the interrelated components work towards a common objective.
  - *Mission critical*: Refers to any factor that is crucial to the successful completion of an entire project.
  - *Safety-critical system*: A computer, electronic or electromechanical system whose failure may cause injury or death to human beings.


Many types of systems meet the critical systems definition, such as: medical devices, military equipment, nuclear power systems, and spacecraft instruments. The main differences between these areas are the amount of emphasis within each that may be placed upon “usage, environment, software rigor, change or update frequency, dependability, and maintenance” [14]. For example, medical devices require certification by the Federal Drug Administration (FDA) and therefore the software development rigor is very rigorous, since it requires the review and approval by a governance organization.
The topic of critical systems remains quite broad, and so, the scope is further restricted to that of space-based systems. This includes mission management, spacecraft operations, and ground system operations [15] [16] [17]. Even so, a further narrowing of scope to one of those more-specific areas is not done. Instead, we consider this representative decomposition of space-based systems to be analogous to a three-legged stool, in that all three legs must remain viable in order for the stool to remain standing. Thus, no differentiation is made among spacecraft development, ground operations, or mission management as the most critical. Within the domain of spaced-based systems, faults within any of these areas can lead to a critical failure within the entire system.

Space-based systems are complex and composed of multiple, interrelated, complex sub-systems. To achieve high confidence in a system of embedded systems, development is much more complex than development of monolithic embedded systems [18].

Spacecraft instruments also requires high software development rigor, but because there is not a governing or certification organization, such development instead requires significant review and oversight by the customer. Sub-systems are often built by different sub-contractors, each of which uses their own processes and methodologies to design, create, and assemble their respective systems. Coordination activities between the various agents require special attention to the overall systems engineering and project management. Likewise, integration and testing of the components requires structured systemization as they are assembled into the final products and deployed within the scope of the mission.

Critical systems have as their primary requirement high reliability and thus strict safety and performance criteria. That is, failure is to be avoided.
To achieve safety and reliability goals, critical systems development uses formal methods [19] [20] and favors redundancy [21] within the system design, and testing and documentation are especially important with respect to such systems [14]. Redundancy helps mitigate the possibility of system failure due to overlooked or unforeseen problems. Formal methods provide a structured framework by which work progression can be monitored and verified for accuracy. Thorough and complete documentation is especially important to ensure that requirements and design are complete and well understood.

Traditional engineering-based approaches expect that “problems are fully specifiable and that optimal and predictable solutions exist for every problem [7].” Thus, traditional development methods first elicit and document a fully complete set of requirements before proceeding to architectural and high-level design, development, and inspection [9].

A key problem faced by critical systems development is difficulty in requirements management. Often, critical systems development focuses on the single system deliverable at hand, making it difficult to modify the project focus to absorb changes or to reuse the products which are produced. The inability for re-use outside of a specific problem domain may be attributed to the narrowed specificity of the “architectural designs and fault tolerant control methods [21]”. As an example, in discussing NASA’s communications infrastructure, which needs to accommodate the service needs of each new mission, Bhaskin and Hayden explain:

A vertically organized mission used infrastructure pieces that were designed to that single mission’s requirements resulting in communications assets only useful to that single mission. While it was possible to reuse some of the assets, such as the larger ground antennas, even these had to be modified to
handle new mission requirements as they came along. Since this hardware was designed to operate at the lower frequencies (S- and X-band) and was not designed for flexibility, it has been increasingly difficult to bring the assets up to the capability of the commercial satellite systems that now operate at very high bit rates in the K-band.

As a side-effect of the increasing complexity of requirements, another problem is the difficulty by which dependability of complex systems can even be estimated [21].

Effects of high complexity are not limited to requirements management, but can also be problematic within architecture and design. Although application of general, reusable design patterns is an appealing prospect, it is a better design goal for safety-critical component architecture to “keep it as simple as possible [21].”

Summary

- The types of critical systems are differentiated by their characteristics of usage, environment, software rigor, change or update frequency, dependability, and maintenance.
- Space-based systems include three main areas: mission management, spacecraft operations, and ground system operations. No differentiation is made among these as the most critical, and, in fact, faults within any of these areas can lead to a critical failure within the entire system.
- Space-based systems are complex and composed of multiple, interrelated, also complex sub-systems. In order to achieve high reliability, each component requires high software
development rigor that is much more complex than development of monolithic embedded systems.

- Sub-systems are often built by different sub-contractors, each of which uses their own processes and methodologies, requiring special attention to the coordination activities between the various agents as they assemble respective sub-systems into the final products for deployment within the scope of the mission.

- The requirement for high reliability leads to strict safety and performance criteria. To achieve this, critical systems development uses formal methods and favors redundancy.

- Key problems faced by critical systems development include:
  
  o Difficulty in requirements management.
  
  o Difficulty by which dependability of complex systems can even be estimated.
  
  o High complexity can also be problematic within architecture and design. It is a better design goal for safety-critical component architecture to “keep it as simple as possible.”

2.1.2. Traditional Methodologies

Prior to reviewing specific space-based systems development scenarios, it is useful to complete a brief discussion of traditional software methodologies often used within the space industry.

While finding an appropriate process methodology to apply can be challenging, “having no process will almost guarantee failure [22].”
All organizations have a methodology, which describes how they do business. Unique to each organization, such methodology “includes who you hire, what you hire them for, how they work together, what they produce, and how they share. It is the combined job descriptions, procedures, and conventions of everyone on your team. It is the product of your particular ecosystem and is therefore a unique construction of your organization [13].”

The manner in which a methodology is put into practice is governed by how the organization organizes itself. Common organization paradigms are:

- Hierarchical – traditional top-down chain of command
- Random – group with little or no central command
- Collaborative – group that works by consensus
- Synchronous – coordinate actions without verbal communications

Chances of success for any mission-critical software development effort are “greatly increased” with an integrated, effective combination of good technology and solid process which supports the follow key principles [22]:

1. Effective requirements management and analysis
2. Reusable component libraries
3. Object-oriented methodologies
4. Prototyping
5. Quality assurance
6. Advanced tools and environments

The development process for dependable control system design and evaluation must focus on the following [21]:

1) Fault tolerance based on non-redundancy
2) Avoiding stovepipe systems
3) Software engineering framework that covers requirements, architecture, design and evaluation

As one methodology, the Capability Maturity Model Integration (CMMI) may be characterized as a process improvement approach used to improve performance by many organizations, including Department of Defense (DoD) and defense contractors [23]. CMMI Product Suite is made up of three separate models – CMMI for Acquisition (CMMI-ACQ), CMMI for Development (CMMI-DEV), and CMMI for Services (CMMI-SVC) – which share common goals and practices. The latest release, CMMI for Development v1.3, has 22 process areas (see Figure 2.1), and the degree to which an organization may need to address these process areas can be related to the complexity of the products it produces. While many organizations do not require all process areas to be implemented, “there are some organizations that for some types of software that may be extremely mission-critical they may need to implement all those 22 process areas [24].”
CMMI strives to achieve process consistency, predictability, and reliability. While mainly focused on large projects and large organizations, CMMI can still be fitted to all sized projects because “it is formulated in a very general way that fits diverse organizations’ needs [9].”

Contrary to expectations, studies show that many CMMI processes were, in fact “largely unpredictable and unrepeatable” because [9]:

1. Applicable first principles are not present
2. The process is only beginning to be understood
3. The process is complex
4. The process is changing and unpredictable

As a methodology specific to providing guidance on systems engineering useful to the NASA community, NASA’s Systems Engineering Handbook provides an alternate technical process called the systems engineering engine [1], shown in Figure 2.2.

![Figure 2.2 – NASA’s systems engineering engine [1]](image)

Other methodologies employed within the space industry include: IEC 61508 part 7, which has a list of development methods IEC considers suitable for safety-related software. [21]; NASA’s Software Safety Guidebook, which has comprehensively listed principles for implementation of real-time software, and FMECA and FTA [21].

**Summary**
• All organizations have a methodology, which describes how they do business, but “having no process will almost guarantee failure.”

• Solid process which supports effective requirements management and quality assurance greatly improves the chances of success for any mission-critical software development effort. The development process for dependable control system design and evaluation must focus on 1) a software engineering framework that covers requirements, architecture, design and evaluation and 2) fault tolerance based on non-redundancy.

• Capability Maturity Model Integration (CMMI):
  o May be characterized as a process improvement approach used to improve performance.
  o Strives to achieve process consistency, predictability, and reliability
  o CMMI for Development v1.3, has 22 process areas.
    ▪ “There are some organizations that for some types of software that may be extremely mission-critical they may need to implement all those 22 process areas.”
  o Has processes which studies have shown are “largely unpredictable and unrepeatable” because:
    ▪ Applicable first principles are not present
    ▪ The process is only beginning to be understood
    ▪ The process is complex
    ▪ The process is changing and unpredictable

• Other methodologies employed within the space industry include
  o NASA’s Systems Engineering Handbook (systems engineering engine)
2.2. Space-Based Systems Development

Objective: Systems development of space-based systems are described so that similar and differing approaches to past development of mission-critical space-based systems can be shown, especially within organizations such as NASA, the military, and SpaceX. Current issues and the future drivers which will influence changes to development methodologies within the industry are also covered.

Development of spaceflight over the past five decades had been driven primarily by specific issues related to national security and conservation of the industrial base. The slow growth of commercial ventures is attributed mainly to market and financial constraints [25], rather than any basic limitation of the available technology [2]. Figure 2.3 shows the global allocation of space activities for 2007.
The key barriers to entry in the space industry include [2]:

1. Large investment requirements
2. Operation and utilization cost uncertainty
3. Market demand and elasticity uncertainty / variability
4. Launcher cost, availability, and reliability
The suggestion that budgetary constraints will drive human space exploration programs to use new/emerging commercial services [26] and to adopt new in-space operational techniques is so persuasive that the Obama administration has adopted them as key elements of future U.S. space policy [2].

Current and future growth is primarily limited by launch issues [2]. Elon Musk, founder of SpaceX, in recounting the history of SpaceX, gives insight into key barriers to success: “I quickly found that the biggest obstacle was the cost of the launch [4].” Musk took action to focus on solving this problem, with an emphasis on limiting costs and an engineering focus on reuse. “I gathered a group of engineers from the space industry to find a way to get the launch cost down. We determined that we could do it by optimizing the design for cost and by making the rocket reusable [4].”

**Summary**

- The key drivers of spaceflight development over the past five decades have been characterized by:
  - Specific issues related to national security
  - Conservation of the industrial base
  - Slow growth of commercial ventures
- The key barriers to entry in the space industry include:
  - Large investment requirements and budgetary constraints
  - Operation and utilization cost uncertainty
  - Market demand and elasticity uncertainty/variability
  - Launcher cost, availability, and reliability
2.2.1. Flight Systems Development

An example of typical space-development practices is eloquently given by Nicewarner [3], who further suggests that using more modern software development practices leads to savings in spacecraft software development time and costs.

While Nicewarner focused mainly on the impact to software development costs based on the hardware processor selection, the premise of the relationship between technology selection and software development methodologies likely extends to other technology tradeoff choices [27].

Typical flight software development practices fail to stress the importance of hardware and software interdependencies. In fact, hardware design decisions are often made without regard for the impact on software development costs, and flight software development typically does not begin until after most of the hardware design decisions are done. Such policies of “institutional inertia and risk aversion” result in spacecraft software development practices being “far behind the state-of-the industry [3].”

The traditional spacecraft hardware processor selection leads to tight resource constraints which preclude the use of the following modern development techniques [3]:

- Modem software techniques (e.g. Standard Template Library)
- Modem software tools (e.g. Linux, CORBA)
- Modem software development methods (e.g. object-oriented development)

To further understand the relationship between technology constraints and cost productivity, we may explore the selection of the programming language for spacecraft development. Historically, the preferred choice in spacecraft software programming language included assembly, Fortran, and some Ada, but according to NASA's software cost databases, the
current preferred language is C. Contrastingly, while spacecraft development rarely use C++, current real-time terrestrial systems prefer C++ as the language of choice [3].

Despite the current reluctance to use C++ in spacecraft development, it is worthwhile to consider it because of its widely recognized benefits. C++ offers object-oriented (O0) constructs that are difficult to implement in C. Decades of software engineering have shown that object-oriented software development is a superior method to produce reliable, modular, testable, flexible, and extensible software. Using a language that directly supports object-orientation seems only natural and also supports the recruitment of programmers with sufficient competency [3]. Yet, adoption of C++ as a successful software development language is beginning to take root, and the Joint Strike Fighter (JSF) program was “very progressive in their use of C++ [3]” on a human-rated safety critical system.

Aside from the programming language employed, flight software development processes should also be examined. Traditional flight software development follows a strict waterfall approach, in which functional decomposition is the driving force. Development standards are often followed, such as DoD-STD-2167, DoDSTD2168, and MIL-STD-1815A. In contrast, modern software techniques are object-oriented, where objects and their interrelationships are the driving force, and are a fundamental departure from structured methods. While these methods capture some aspects of functional decomposition, they do not drive the approach or the resulting documentation. Rather, O0 methods presuppose during development that requirements often change and become more refined, and furthermore offer higher productivity, lower cost, and more flexibility [3].

Nicewarner suggests that there is sub-optimal development productivity in space software development, which results from the industry’s reluctance to use modern languages and
processes. Using data supporting a comparison of Moore’s Law between terrestrial and space systems development (see Figure 2.4), Nicewarner states: “[The] rather large discrepancy between space and non-space industry productivity … can be explained in large part by the lack of modern software development processes in space systems [3].”

![Figure 2.4 – Moore’s Law for processor selection. From [3]](image)

However, there is no supporting evidence that the lack of productivity is directly related to only the choice of technology and methodology. For example, Nicewarner fails to examine the inherent technical complexities of the space domain and thus the space-based systems development in comparison to other non-space industry systems. In fact, the data tends to show that space systems development productivity has always lagged behind terrestrial systems development, but the rate of productivity increase has been similar between the two groupings. Thus, it could be that spacecraft software development is simply a more complex domain to tackle, and productivity may also be lower than non-space domains, independent of technology and process.
Critical systems software testing can be counter-intuitive. For example, the testing goal can be stated as: "prove that there are no hazardous software behaviors.” Proving that “there does not exist” is a daunting, often unachievable task, and therefore, testing is performed for as long as the program can afford it, and ends “without any real proof” that all hazardous behaviors have been found and eliminated [3].” After all, even with strict flight software guidelines, we still see cases of hazardous software behavior in space systems [3].

The testing requirements of critical space systems tend to focus on acceptance levels of fault tolerance, instead of fault elimination, and there are various approaches and tradeoffs which can be made in this area of acceptance. For example, instead of Triple Modular Redundancy (TMR), the fault tolerance approach SpaceDev uses is cross-checking, where multiple non-identical processors are connected together so that they can monitor each other. A combination of software and hardware watchdog timers are used to drive a multi-tiered fault recovery strategy that brings the system to a progressively less-capable state and allows ground commands to attempt to bring the system back to full functionality [3].

Flight software development practices discourage the use of Commercial Off-the-Shelf (COTS) components. Strict restrictions are often given, requiring third-party libraries to be extensively tested and “certified”, if they are to be allowed at all. For example, AV Rule 16 states: “Only DO-178B level A certifiable or SEAL 1 C/C++ libraries shall be used with safety-critical (i.e. SEAL 1) code [3].” Such restrictions often preclude the use of open source libraries and other components which are familiar to software developers.

Summary

- Typical flight software development practices fail to stress the importance of hardware and software interdependencies.
- Hardware design decisions are often made without regard for the impact on software development costs.
- Flight software development typically does not begin until after most of the hardware design decisions are done.

- Traditional spacecraft hardware processor selection leads to tight resource constraints which preclude the use of many modern development techniques, including modern tools, techniques, and software development methods, which sets such development apart of other current traditional systems.
  - The preferred choice in spacecraft software programming language has included assembly, Fortran, and some Ada, and C.
  - C++ use used rarely, although it has gained some popularity recently.

- Flight software development practices discourage the use of Commercial Off the Shelf (COTS) components.
  - Such restrictions often preclude the use of open source libraries and other components which are familiar to software developers.

- Traditional flight software development follows a strict waterfall approach, in which functional decomposition is the driving force.
  - Development standards followed include:
    - DoD-STD-2167
    - DoDSTD2168
    - MIL-STD-1815A

- Contrastingly, object-oriented (OO) methods offer higher productivity, lower cost, and more flexibility.
• The goal of critical systems software testing can be stated as: "prove that there are no hazardous software behaviors."
  o Difficult to prove.
• The testing requirements of critical space systems tend to focus on acceptance levels of fault tolerance, instead of fault elimination.
• Nicewarner suggests that industry’s reluctance to use modern languages and processes results in is sub-optimal development productivity in space software development
  o However, there is no supporting evidence that the lack of productivity is directly related to only the choice of technology and methodology.
    ▪ For example, Nicewarner fails to examine the inherent technical complexities of the space domain and thus the space-based systems development in comparison to other non-space industry systems.
• Nicewarner, suggests that using more modern software development practices leads to savings in spacecraft software development time and costs.

2.2.2. Critical Systems Development and Formal Methodologies

Development processes focused on dependability should not limit themselves to one or two methods, but must combine all possible methods to achieve the desired level of dependability - fault prevention, fault tolerance, fault removal, and fault forecasting [21].

Formal software testing methodologies are often employed. For example, development and evaluation of dependability systems with a targeted safety integrity level (SIL) often use software testing standards such as IEEE 829-1998 and some of the American internationally recognized standards including ISA 84 series and MIL-STD-882D [21].
Regardless of the specific methodologies adopted, the formal methodologies followed must be especially effective for critical systems to succeed. There are no “silver bullets” in this realm [14], and challenges remain to uncover ever-more-effective processes and best practices. Alho and Matilli, in presenting their research on mission-critical development processes, state: “Contributions are especially needed for development of cost-effective systems engineering (SE) practices and guidelines for fault-tolerant implementation [21].”

Summary

- To achieve the desired level of dependability - fault prevention, fault tolerance, fault removal, and fault forecasting - development processes must combine all possible methods.
  - Formal software testing methodologies are often employed.
    - IEEE 829-1998
    - ISA 84 series
    - MIL-STD-882D
- The formal methodologies followed must be especially effective for critical systems to succeed.
  - There are no “silver bullets” in this realm.
  - Continued research is needed to uncover “cost-effective systems engineering (SE) practices and guidelines for fault-tolerant implementation.”

2.2.3. The Role of Architecture

Developing space exploration systems of the future will be quite challenging. Such systems will be required to provide highly complex functionality, be software-intensive, and
“must tolerate the subtle faults of asynchronous systems running in a hostile environment and be affordable, reliable, flexible, robust, and gracefully upgradeable.” Current tools, technologies, and techniques are beyond the acceptable limits of current cost resources [28] [15].

Cooke places strong emphasis on developing new software architectures for space-based systems which will lead to dependable architectures and reduce risk. Software architecture (SA) defines the components and structure of a software system, and provides the basis for “design, coding, testing, integration, and planning.” Architecture-driven development speeds development, improves quality, and is essential to risk management. “Without an explicit and carefully defined architecture, risk assessment and mitigation is infeasible [28].”

Ultimately, Cooke concludes that architectural elements, especially interfaces, are of utmost importance to mission critical software systems development. “The architectural interfaces, in particular, become the critical coordinating element in developing software systems by geographically separated teams, organizations, and companies. A prime advantage of such an organizing structure is that the different participants can use independent operating environments—they do not need to have the same processes and tools.” Such focus on software architecture coordination allows an organization to effectively support different processes, tools, project management methodologies, quality structures, cultures, and organizational structures [28].

**Summary**

- Software architecture (SA) defines the components and structure of a software system, and provides the basis for “design, coding, testing, integration, and planning.”
- Architecture-driven development speeds development, improves quality, and is essential to risk management.
“Without an explicit and carefully defined architecture, risk assessment and mitigation is infeasible.”

- Cooke concludes that it is architectural elements, especially interfaces, that are of utmost importance to mission critical software systems development.

### 2.2.4. Commercial Off-the-Shelf (COTS)

Software component re-use of COTS components presents certain problems, as the quality for mission-critical applications is usually not guaranteed, and thus may require additional fault-tolerance considerations [21].

Commercial off-the-shelf (COTS) products are hardware / software products produced by commercial companies, government agencies, and the military and other programs. When integrating COTS products for space-based system, extra precautions need to be taken to ensure the products can endure the harsh environment of space. This includes conditions involving “vacuum, radiation, extreme temperature ranges and lighting conditions, zero gravity, atomic oxygen, lack of convention cooling, launch vibration and acceleration, and shock loads [1].”

To ensure compliance with expectations, requirements must be actively managed when considering COTS products, and care must be taken when deciding if requirements may be relaxed or waived, or in considering if the existing product needs to be modified. NASA’s Systems Engineering Handbook lists the following factors which should be considered when a COTS product is under consideration [1]:

- Heritage, or the original manufacturer’s level of quality and reliability, of the product;
- Critical or noncritical application;
• Amount of modification required and who performs it;
• Whether sufficient documentation is available;
• Proprietary, usage, ownership, warranty, and licensing rights;
• Future support for the product from the vendor/provider;
• Any additional validation of the product needed by the project; and
• Agreement on disclosure of defects discovered by the community of users of the product.

Summary

• Software component re-use of COTS components presents certain problems:
  o The quality for mission-critical applications is usually not guaranteed
    ▪ May require additional fault-tolerance considerations.
  • When integrating COTS products for space-based system, extra precautions need to be taken to ensure the products can endure the harsh environment of space.

2.2.5. Mission-Critical Failures – Root Cause Analysis

We are human. Thus, failure happens.

“Failure is inevitable and inherent. Management techniques can only manage how failure is likely or not likely to occur. Engineering is a human endeavor and because of this we cannot achieve a risk free society as we become more dependent on ever more complex technologies [10].”

Human errors result in all kinds of technical disasters, some fatal. Some of the more recent disasters which many people recall include the fatal disasters of the Challenger (see Figure
2.5) and Columbia Space Shuttles and the flawed mirror on the Hubble Space Telescope (HST). Though not fatal, the debacle of the 4 micron HST mirror flaw (see Figure 2.6) impacted many people associated with all types of association with the project. “A trivial and obvious error overshadowed the accomplishments of thousands of dedicated people [6]!”

Figure 2.5 – Space Shuttle Challenger Explosion (Credit: www.aerospaceweb.com)
To understand mission critical systems, we must also understand their failures, for it is from failure that we can often learn the most. Taking, for example, the investigation into the HST mirror, investigators learned that several tests had given hints of a flaw in the mirror. The investigators wondered why bright people had not pursued, reported and addressed these indicators. They found that schedule and budget pressures caused them to move forward relentlessly while the contractor did not forward the troubled results to NASA. And why was that? The investigation board ultimately informed Congress that the $1.7 billion telescope had a flawed mirror because of “a leadership failure” and that the contractor’s relationship with NASA was “so hostile that they would not report technical problems if they could rationalize them. They were simply tired of the beatings [6].”

Pellerin details several other failures of critical systems, which are summarized in Table 2-2.
### System | Event | Background | Cause of Failure
--- | --- | --- | ---
Space Shuttle | Challenger Explosion | Diane Vaughan wrote the following about the Challenger explosion (The Challenger Launch Decision, 1996). “I present evidence that refutes the traditional explanation that blames managerial wrongdoing... an incremental descent into poor judgment... The revisionist history and sociological explanation presented here is more frightening than the historically accepted interpretation, for the invisible and unacknowledged tend to remain undiagnosed and elude remedy.” | Sociological Conditions
Hubble Space Telescope | Flawed Mirror | Hubble attracted first-rate technical minds. We, the management, created a social context that put these good people in bad places. NASA managers at headquarters and Marshall Space Flight Center, the managing field center for Hubble, including me, relentlessly criticized and pressured our contractors. The contractors, operating from a place of relative powerlessness, engaged in guerilla tactics by withholding troubling information. | Social context management failure
NOAA-N Prime | $200 million satellite destroyed | A crew of technicians destroyed a nearly finished $200 million satellite, “NOAA-N-prime.” They turned a cradle holding the satellite on its side not realizing that another crew had removed the fastening bolts beneath. They ignored a procedure that required verifying that the bolts were in place. The LockheedMartin failure review board asked me to join them in the failure investigation because they thought an institutional culture of sloppiness might be the root cause. I passed because I did not have the available time. They asked me to meet with them after they drafted their report. They said, “We wanted to talk to you because we are reporting that the culture was root cause.” I later read the government review team’s report (2004): “The operations team’s lack of discipline in following procedures evolved from complacent attitudes toward routine spacecraft handling, poor | The "culture" of complacency
Communication and coordination.”

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Crash</th>
<th>The context that typically causes plane crashes is rushing tired pilots in poor weather leading to communications breakdowns (Gladwell, 2008). A typical accident involves seven consecutive human errors. These errors are rarely errors of knowledge or flying skill. They are errors of teamwork and communication.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Korean Air</td>
<td>Crash</td>
<td>During the period 1988 to 1998, Korean Air was crashing at a rate 17 times greater than the industry average. Investigators found that the root cause was transference of the Korean social context into the cockpit. The captain’s social status was so high that the junior officers in the cockpit could only communicate obliquely and deferentially. The captain was flying the plane by himself. Modern jets require a team of two or three for safe flight. No amount of individual training in flying skills would help. Today, Korean Air’s record is as good as the overall industry’s because experts from Aleton (a subsidiary of Boeing) changed the social context to a team of equals.</td>
</tr>
</tbody>
</table>

| Errors of teamwork and cooperation |
| Social context of captain’s status (presumed superior to) crew in the cockpit. |

**Table 2.1 – Examples of critical system failures.** *From [6]*

Figure 2.7 shows the NOAA-N Prime failure.
Figure 2.7 – NOAA-N Prime failure (Credit: www.spaceref.com)

System failures can occur at any stage of the systems development life cycle, and are not isolated to the design, development, integration, and deployment phases. Failures can occur well into the life of the mission. For example, in February, 2013, the NASA ground crew temporarily lost communications with the International Space Station (ISS) crew for nearly 3 hours as result of a software upgrade occurring during the Expedition 34 mission. “Flight controllers were in the process of updating the station’s command and control software and were transitioning from the primary computer to the backup computer to complete the software load when the loss of communication occurred. Mission Control Houston was able to communicate with the crew as the space station flew over Russian ground stations before 11 a.m. and instructed the crew to connect another computer to begin the process of restoring communications [29].”
Summary

- As humans engineering more complex systems, failure is inevitable and inherent.
- All kinds of technical disasters, some fatal, result from human errors.
- To understand mission critical systems, we must also understand their failures.
- Common root causes of mission-critical systems failures include cultural, sociological conditions, and errors resulting from the social context.

2.2.6. NASA’s Faster, Better, Cheaper (FBC)

Traditionally, NASA achieved their goal of reducing risk by enacting a reinforcing loop whereby small changes towards improved reliability were satisfied through increasing cost and schedule [10]. This viewpoint was countered by former NASA Administrator Dan Goldin, who believed that losing 2 out of 10 spacecraft (20%) was an acceptable risk for going faster and testing new technologies. For non-manned spacecraft, he stated, “A project that’s 20 for 20 isn’t successful. It’s proof that we’re playing it safe [10].” Unfortunately, the final record for NASA’s FBC initiative was 6 failures out of 16 attempts (37.5%), including 4 out of 5 within one year (80% in 1999), significantly higher than Goldin’s risk acceptance target.

With the new directive of Faster, Better, Cheaper (FBC), the management teams, when faced with tight schedules and low budgets, abandoned the processes associated with formal systems management and fatally neglected to institute teamwork methodologies [10].

“In FBC, formal systems management and systems engineering is relaxed and replaced by teamwork. This is where the pendulum swings from formal process to tribal knowledge systems. It is believed that team based management can assure reliability and mission success.
These programs avoid extensive systems engineering and systems management as the main means for avoiding risk and guaranteeing mission success. Born from the early manned space ventures and the Air Force’s ICBM program, systems engineering and systems management is viewed as too cumbersome and expensive. FBC programs forego many of the processes and discipline of systems engineering and systems management. Question becomes, what are the appropriate processes for mission success [10]?

At NASA, several FBC programs “failed because they neglected to adopt necessary practices in systems management [10].” The most prominent recurring themes of FBC failures are the lack of reviews, lack of risk management and assessment, testing, and communications [10]. It is perhaps ironic that the departure from formal methods into team based management resulted in failures due to lack of communications.

**Summary**

- Traditionally, NASA achieved their goal of reducing risk by enacting a reinforcing loop whereby small changes towards improved reliability were satisfied through increasing cost and schedule.
  - NASA Administrator Dan Goldin, countered this view by believing that losing 2 out of 10 spacecraft was an acceptable risk for going faster and testing new technologies.
    - “A project that’s 20 for 20 isn’t successful. It’s proof that we’re playing it safe.”
    - The new initiative came to be known as NASA’s Faster, Better, Cheaper (FBC)
- With FBC, NASA management teams faced with tight schedules and low budgets:
  - Abandoned the processes associated with formal systems management
  - Fatally neglected to institute teamwork methodologies
  - Avoid extensive systems engineering and systems management as the main means for avoiding risk and guaranteeing mission success
  - Replace formal process with teamwork and tribal knowledge systems
- Ultimately several FBC programs “failed because they neglected to adopt necessary practices in systems management”. The most prominent recurring failure themes are:
  - Lack of reviews
  - Lack of risk management and assessment
  - Lack of testing
  - Lack of communications

### 2.2.7. The Importance of Social Context

System failures highlight the risks of ignoring the environment and culture of humans working together, or, more commonly, the social context. Space mishaps provide opportunities for the most rigorous investigations. “When accidents kill astronauts and teachers, the inquiry is very thorough [6].”

NASA’s Stephen Johnson asserts that while the failure effects and proximate causes may often be technical, the root causes are mainly due to the social and psychological impact resulting from ineffective communications environments which lead to human error. “Although the statistics have not been studied fully, my sense, from experience in the field and discussions
with other experienced engineers, is that 80 to 95 percent of failures are ultimately due to human error or miscommunication [6].”

Pellerin’s failure analysis of these mission critical systems led him to conclude that it is of the upmost importance for technical teams to focus in on team social context, or environment, which includes collective behaviors and team behavioral norms. His book and coursework on “How NASA Builds Teams” details the approach widely recognized and now practiced within many NASA organizations [6].

Improving social context management is predicated on the criticality of managing the story-line, or narrative, of the work being done. While individual skills and technical abilities are required for mission critical systems development, it is the effectiveness of the team’s social content management that is the most important indicator of the team’s performance. Improving social context management is the key to successfully turning around failing projects and making them effective. “It is more productive to enhance team context than replace team leaders, a far more common management practice when difficulties arise [6].”

Indeed, this concept of “human capital” – as Ehrenfreund describes it – is one of the critical elements to successes in future space development programs. “For the development of a successful global space exploration program, traditional approaches may need to be supplanted by a new paradigm [30] including focus on information exchange, organizational knowledge, and human capital as practiced in high-performance organizations (HPOs) that go well beyond the current international working groups and multinational space efforts [31].”

Management must play a key role in fostering and promoting environments of cross-cultural cooperation. Differences in core values and cultural dimensions, as well as language barriers, will strongly influence negotiations and collaborative endeavors. Strong differences in
the world-views of individualist and collectivist spacefaring countries mean different priorities. A greater cultural awareness will therefore enable the development of better and longer-lasting strategies [31].

Strategies for successful cooperation in space exploration include [31]:

- Open communications
- Improvement in cross-cultural management
- Optimizations in technical capacities and transnational cooperation in aerospace
- Fostering creativity, intrapreneurship and entrepreneurial orientation.

In order to more quickly react to technology changes, to accommodate cultural differences, and to better control costs, space development organizations will have to take a different approach in the future to how they develop systems. In a recent study on the financial planning of space and defense projects, Deloitte Consulting LLP suggests dividing programs into less complex work packages with shorter durations. Organizational changes required to achieve these goals cannot be immediately put in place, but transition from old methodologies to realize such new goals must proceed in a “stepwise implementation [31].”

Summary

- Social context is the environment and culture of humans working together.
  - Includes collective behaviors and team behavioral norms
- Failure effects and proximate causes:
  - Often technical
- The root causes mainly due to:
Social and psychological impact

- Resulting from ineffective communications environments
- Which lead to human error

Improving social context management:

- is predicated on the criticality of managing the story-line, or narrative, of the work being done.
- is the key to successfully turning around failing projects and making them effective.

In order to accommodate cultural differences, space development organizations will have to take a different approach in the future to how they develop systems.

Management must foster and promote environments of cross-cultural cooperation using strategies such as:

- Open communications
- Improvement in cross-cultural management
- Optimizations in technical capacities and transnational cooperation in aerospace
- Fostering creativity, intrapreneurship and entrepreneurial orientation.
- Dividing programs into less complex work packages with shorter durations

Organizational changes required to achieve these goals:

- Cannot be immediately put in place
- Must proceed from old methodologies to new in a “stepwise implementation.”
2.2.8. Drivers for Development Improvement

The high costs of launch remain the primary barrier to entry in space exploration. Thus, cost reduction, or profitability, is the key driver towards reducing entry barriers. The main obstacle to achieving sufficient profitability is the lack of cost-efficient reusable launch vehicles (RLVs). It has long been recognized that the only way of achieving significant improvements in space access is via the use of RLVs because they offer the following: major reductions in marginal costs, better amortization of investments, and higher reliability and safety. It is around such technologies that current systems development efforts focus heavily (see Figure 2.8).

Figure 2.8 – RLV vision (Credit: www.styleofspeed.com)
Government efforts towards RLVs have either missed original goals (Space Shuttle) or been outright failures (X-33/VentureStar X-34), and commercial efforts have been hampered by high development costs [2].

The more recent key events in the U.S space sector have resulted in more reactive changes than pre-emptive ones. For example, recent key critical system failure events, such as in-space satellite collisions and failed launches, have led directly to reactive responses of new industry and national policy changes [32]. Cornell lists the five key events of the last 20 years in the U.S. space industry as:

- The consolidation of the aerospace and defense industries in the 1990’s
- Following failed satellite launches in 1995 and 1996, the charging of Boeing (Hughes) and Space Systems / Loral with the violation with the violation of the Arms Export Control Act and the International Traffic in Arms Regulation (ITAR)
- The creation of Elon Musk’s SpaceX Corporation in 2002
- The collision of the U.S.’s Iridium 33 and Russia’s Kosmos 2251 satellites in 2009
- The cancellation of NASA’s Constellation program in 2010 and President Obama’s new space program proposal

These events are leading nations, academics, companies, and other organization to rethink exactly how space exploration will proceed in the future. Restricted budgets and national policy changes will lead all interested parties to shift towards a globalized, cooperative approach to pursuing space. The resulting changes represent “shifts in the way the American space industry has needed and will need to innovate and change to react to the globalization of space.” [32].
Yet cultural differences between the collective goals of spacefaring countries exist [31]. This further complicates the notion globalization of space, as the needs of the individual stakeholder countries and organizations may not always align, due to political reasons or even language differences.

Future globalization of space efforts must focus on five key performance objectives: quality, speed, dependability, flexibility, and costs. Among these, quality is probably the most important [31].

1. **Quality** can be improved through knowledge exchange between space powers and the space industry at the transnational level.

2. **Speed (sustainability)** could be improved by an efficient and modern management system that aligns national/international stakeholders and allows for a faster decision-making process and provides sufficient funding resources.

3. **Flexibility** could be increased by forecasting problem areas and having several backup solutions at hand, in particular at the technological and funding level.

4. **Dependability (interdependency)** is, however, difficult to reduce and must therefore be optimized in a way that fosters stronger collaboration and strengthens efficient international technology exchange.

5. **Cost efficiency** could be achieved when all other performance objectives are met and, most of all, through successful project management.

Currently, the goals and objectives of space exploration are not time-bound [31]. However, the anticipated direction within mission-critical space systems development is that of decomposition of traditionally large, monolithic systems into smaller time-bound work units.
This presents particular challenges for the space development industry. Business drivers are therefore leading new space age development approaches to systems development by “making use of existing systems and technologies whenever possible and employing small teams that evolve design in small rapid steps (build, test, correct, …) [2].”

How should space companies prepare to be most effective in future systems development efforts? Cornell prescribes that “companies should look to stay ahead of the policies and to become involved more in the regulation as it is being developed,” although this may especially challenging for larger, traditional aerospace- and defense industry- based space companies, which are more difficult to change culturally [32].

**Summary**

- The high costs of launch remain the primary barrier to entry in space exploration.
  - The main obstacle to achieving sufficient profitability is the lack of cost-efficient reusable launch vehicles (RLVs).
- The more recent key events in the U.S space sector have resulted in more reactive changes than pre-emptive ones.
  - These events are leading nations, academics, companies, and other organization to rethink exactly how space exploration will proceed in the future.
    - Restricted budgets and national policy changes will lead all interested parties to shift towards a globalized, cooperative approach to pursuing space.
- Future globalization of space efforts must focus on key performance objectives:
Key business drivers are leading new space age development approaches to systems development:

- making use of existing systems and technologies
- employing small teams that
  - evolve design in small rapid steps (build, test, correct, …)

### 2.2.9. Systems Development at SpaceX

While NASA is certainly a producer of mission critical and safety-critical systems, they are simultaneously involved in a number of development projects, and clearly not all focus on reusability. We must thus also consider entities with a greater emphasis on reusability, and reliability. Space Exploration Technologies (SpaceX) is just that type of group. With the goal of drastically reducing the costs of space access, “SpaceX has looked at each step in the operations chain with the goal to minimize cost while maximizing reliability and repeatability [33].”

SpaceX willingly embraces the usage and, as necessary, customization, of third-party products, when such use “meets very strict cost-benefit trades [33].” Early in the company’s launch and mission operations development, SpaceX realized the potential benefit of using “off-the-shelf equipment and tools such as PC workstations, standard Windows and Linux servers,
and the LabVIEW™ development platform,” in that such readily available systems allowed SpaceX to develop critical monitoring and control systems (see Figure 2.9) while constraining costs [33].

Figure 2.9 – SpaceX launch and control center. [4]

The launch of the Falcon 1 (see Figure 2.10) provides an excellent example of how SpaceX’s goals of space-access cost reduction operations goals are being met through the reliance on commercial off-the-shelf (COTS) software. For the second launch of their Falcon 1 launch vehicle, launch operators relied on a workstation PC and three off-the-shelf LCD displays, while they monitored and controlled the systems using “laptop computers linked up to
central servers running Microsoft Windows Server™ and Linux operating systems.” The arrangement of the equipment was, “laid out in a configuration that best suited SpaceX’s needs and still kept costs low [33].”

![Image of SpaceX Falcon 1 launch](image)

**Figure 2.10 – Launch of SpaceX Falcon 1** [4]

Furthermore, simple customization of existing products saves development time and costs and allows SpaceX to satisfy and even improve upon their existing processes. While “many off-the-shelf analysis and forecasting applications are modified by SpaceX developers for mission use,” one example is the use of Google™ Earth, which has been used to monitor launch vehicle recovery. “Google™ Earth provides an outstanding visualization tool for the operator that is easily portable, has a clear interface definition, and is free of charge.” In fact, the tool was
effectively used by operators to quickly spot and correct the incorrect path of a recovery ship [33].

Beyond launch and mission control, SpaceX development processes are synergistic in their emphasis on teaming. “Specific care was taken to ensure that ‘flight’ software developers (flight code onboard the vehicle) worked closely with the ‘ground’ software developers to ensure proper relationships between spacecraft fault detection, isolation and recovery (FDIR), and ground commanding and anomaly resolution. Failure to respect the close nature of this relationship could actually result in increased operator complexity, confusion, and cost. SpaceX works diligently to eliminate this possible complication by having both software teams not only work (and code) in close coordination, but also simulate and ‘fly’ together to understand each other’s needs [33].” It is highly worth noting that this critical aspect of the SpaceX development methodology extends beyond tools, technologies, and standard documented development processes. SpaceX’s methodology also reaches into the cultural objectives and process areas within the organization which focus on the working environment and need for cooperation among the people – the scientists, technicians, and programmers themselves.

Musk captures the build or re-use dilemma nicely: “We had built an entire rocket from the ground up, with almost no legacy hardware. The reason was simple: If you use legacy parts, you’ll limit opportunities to reduce costs; if you don’t use them, the risk of failure goes up [4].” Figure 2.11 shows Musk with the company’s Falcon 9 rocket, which blasted SpaceX's Dragon capsule into orbit in December 2010.
By first focusing on repeatability, SpaceX has been able to stick to its goals of space-access cost reduction, while working towards the reusability requirement. Musk explains: “In principle, all our future flights should work if we build them the same way. Reusability will come later. It’s hard; nobody has ever really achieved it. Reusability is critical … so this is something we have to get right and hone to perfection [4].”

“Even the space shuttle isn’t really reusable, in that it costs more per flight than it would to buy a new expendable launch vehicle of greater cargo capacity [4].”

As one example of the relatively young company to enter the space exploration market, SpaceX has had to address head-on this reactive need to redefining how space exploration is
done. SpaceX abandons many of the typical space development paradigms and “is taking advantage of being every-thing that the big, American companies are not to be competitive [32].”

As a smaller company with a younger workforce, SpaceX embraces new approaches to the field, such as [32]:

1. An ability to build fast-moving and efficient teams and organizations geared around a core mission and competency,
2. The capacity for innovative ‘out-of-the-box’ thinking with the potential to generate substantial cost reductions over traditional space industry approaches
3. A culture and financial structure which accepts higher levels of technical and market risk

One of the most important aspects of SpaceX and other new space companies is their emphasis on innovation [34]. For example, SpaceX is not influenced by traditional systems designs and instead performs design from the bottom up, while still able to preserve system reliability expectations [32]. For example, with redundancy in mind, the Falcon 1 rocket was designed with nine Merlin 1C engines powering the first stage. If one engine fails, it does not necessarily translate into total rocket launch failure.

Another driver behind SpaceX’s new innovative thinking comes from their cost-cutting focus and small company, flat management organizational structure. SpaceX’s employee size (approximately 900 employees) and “start-up” environmental mentality “allows for more intermingling and collaboration throughout the design process for all systems [32].”

The future of space development will be driven by changing domestic and international policies and will need to be an environment supporting increased innovation [32]. As nations and organizations embrace a globalization approach to continued development, the proper
supportive environment promoting cooperation, sharing, and cultural awareness must be effectively established among the cross-organizational teams that are involved. While there will most certainly be significant technology advances and development process improvement in the future, the strong need for social content management, in support of technology and process, should not be ignored.

**Summary**

- Space Exploration Technologies (SpaceX) emphasizes reusability and reliability (repeatability), with the goal of drastically reducing the costs of space access.
- SpaceX willingly embraces the usage and customization, of third-party products, when such use “meets very strict cost-benefit trades.”
  - Launch operators use Microsoft Windows Server™ and Linux operating systems
  - Critical monitoring and control systems use LabVIEW™ development platform
  - Google™ Earth, which has been used to monitor launch vehicle recovery
- To preserve system reliability expectations, the Falcon 1 rocket was designed with nine Merlin 1C engines powering the first stage.
  - If one engine fails, it does not necessarily translate into total rocket launch failure
- Development processes emphasize teaming.
  - Flight software developers work closely with the ground software developers.
Methodology also reaches into the cultural objectives and process areas within the organization, including the need for cooperation among the scientists, technicians, and programmers themselves.

- SpaceX embraces new approaches, such as:
  - An ability to build fast-moving and efficient teams and organizations geared around a core mission and competency,
  - The capacity for innovative ‘out-of-the-box’ thinking with the potential to generate substantial cost reductions over traditional space industry approaches
  - A culture and financial structure which accepts higher levels of technical and market risk

2.3. Agile

**Objective:** Agile methodologies are reviewed with emphasis on how to decide which ones may be applied to the systems development of space-based systems. Not all agile methodologies can be applied to the development of critical-systems and care must be taken when deciding which methodologies to apply effectively.

Alluding to the difficulty with describing exactly what software development is, Alistair Cockburn, one of the original authors of the Agile Manifesto [12] [13], writes:

“Is software development an art, a craft, science, engineering, or something else entirely? Does it even matter?

“Yes, it does matter, and it matters to you. Your actions and their results will differ depending on which of those is more correct [13].”
The early motivation for Agile is best stated by Cockburn: “The main thing is this: You want your software out soon and relatively defect free, but more than that, you need a way to examine how your team is doing along the way” [13]. Though an appealing prospect, we must also respect that perfection requires repetition. “It is impossible to get all but the simplest products right in one pass [8].”

The foundation of Agile is built upon two primary ideas and four main values [13].

The primary ideas of Agile are:

1. Different projects need different processes or methodologies.
2. Focusing on skills, communication, and community allows the project to be more effective and more agile than focusing on processes.

At its core, the Agile Software Development Manifesto stresses four main values [13]:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Cockburn states that treating software development as engineering or model building activity does not help make meaningful, advantageous project decisions. Instead, he puts forth the notion that it is more profitable to consider software development as a “game with moves” [13] in which the team must consider the tradeoffs among many, small advances, each of which moves towards the ultimate goal.

To understand this further, we must explore the definition of engineering, which The Meriam-Webster online dictionary gives as:
a. the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

b. the design and manufacture of complex products <software engineering>

The first definition states that things are engineered through the application of science and mathematics in accordance with physical laws of nature. Thus, these engineering processes must precisely follow such laws. There is no invention or development, but the repeated application of known rules. The second definition, however, counters the first, by including within the definition of software engineering the idea of “design”, or creation, within complex products. This is not the repetition of known rules, but instead the search for the new, and the desire to learn that which is not known.

Software development is a fluid activity that requires people to practice reflection and “notice small discrepancies wherever they lie and that they communicate and resolve the discrepancies in whatever way is most practical [13].”

People are an important aspect to any project and are “a crucial element of project outcome” because people must communicate in order to share information. Yet, “perfect communications are impossible” within any software project team or organization [13].

Humans will still be humans. Agile does not change that. “Humans are stuffed full of personality. They vary by hour, by day, by age, by culture, by temperature, by who else is in the room. A person can be cooperative with one person at one moment and belligerent the next [13].” Perhaps comedian Ron White best expresses such limitations of humanity when he sarcastically states in his album of the same name, “You Can’t Fix Stupid.”

Cockburn gives the five failure modes of people [13]:

1. Making mistakes
2. Preferring to fail conservatively
3. Inventing instead of researching
4. Being creatures of habit
5. Being inconsistent

Teams, being composed of humans, must accept their inadequacies, and work to effectively communicate their results. In evaluating decisions about the “next move”, teams must examine their work products. In traditional waterfall methodologies, some work products, such as extensive (or “fully complete”) requirements documents and design artifacts, can become very large and complex to evaluate. Teams often devolve into “analysis paralysis” as they begin to over-analyze an existing work product and become risk averse to making a decision to move forward to the next activity. Agile strives to address this by focusing on smaller intermediate work products that are sufficient enough, but not too extensive, so that effective decision-making can be completed. Since the only value of intermediate work products is to help the team decide on their next move, a “focus on light but sufficient intermediate work products should be the primary goal, and Agile aims to help achieve that [13].”

Teams should focus on efficient information transfer through effective human interactions - proximity, information visualization, voice, and questioning. “Every project team should be on a drive to reduce the total energy cost of detecting and transferring needed ideas [13].”

Agile does not resolve all the misfortunes experienced within software development organizations. For example, Agile will not lead to large program production productivity increases. “As much as programming languages may improve, programming will still be limited by our ability to think through the problem and the solution, working through the details of how the described solution deals with the myriad cases it will encounter [13].”
Agile methodologies do not fix the inherent problems of a team. If the team is bad at communications, or testing, or documenting, then adopting Agile methodologies alone will not fix those problems. Certain Agile methodologies will, however, make it easier for the team to identify such problems, so that the team can take corrective actions.

Efficient communications remains the goal, but this does not always demand lowered costs. Project cost increases can be justified if they serve the methodology correctly, and are not merely wasted expenses. For example, high-risk systems require high reliability. “As the degree of potential damage increases, it is easy to justify greater development cost to protect against mistakes. In keeping with the second principle, adding methodology adds cost, but in this case, the cost is worth it. The cost goes into defect reduction rather than communications load [13].”

**Summary**

- The early motivation for Agile is best stated by Cockburn: “The main thing is this: You want your software out soon and relatively defect free, but more than that, you need a way to examine how your team is doing along the way”.
- The Agile Software Development Manifesto stresses four main values:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- People are an important aspect to any project
- Software development is a fluid activity that requires people to practice reflection
- Agile does not resolve all the misfortunes experienced within software development organizations.
For example, Agile will not lead to large program production productivity increases.

- Agile methodologies do not fix the inherent problems of a team.

2.3.1. Agile Development as Compared to Traditional Methodologies

The changes brought about with agile’s arrival on the scene presented a huge challenge to software development practices. “Agile software development represents a major departure from traditional, plan-based approaches to software engineering [7].”

Dybå’s characterization of the main differences between traditional and agile development is given in Table 2.2 [7].

<table>
<thead>
<tr>
<th>Fundamental assumption</th>
<th>Traditional development</th>
<th>Agile development</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Systems are fully specifiable, predictable, and are built through meticulous and extensive planning</td>
<td>High-quality adaptive software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change</td>
</tr>
<tr>
<td>Management style</td>
<td>Command and control</td>
<td>Leadership and collaboration</td>
</tr>
<tr>
<td>Knowledge management</td>
<td>Explicit</td>
<td>Tacit</td>
</tr>
<tr>
<td>Communication</td>
<td>Formal</td>
<td>Informal</td>
</tr>
<tr>
<td>Development model</td>
<td>Life-cycle model (waterfall, spiral or some variation)</td>
<td>The evolutionary-delivery model</td>
</tr>
<tr>
<td>Desired organizational form/structure</td>
<td>Mechanistic (bureaucratic with high formalization), aimed at large organizations</td>
<td>Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations</td>
</tr>
<tr>
<td>Quality control</td>
<td>Heavy planning and strict control. Late, heavy testing</td>
<td>Continuous control of requirements, design and solutions. Continuous testing</td>
</tr>
</tbody>
</table>

Table 2.2 – Differences between traditional and agile development [7]

The five main criticisms of agile development methods are [7]:

1. Agile development is nothing new; such practices have been in place in software development since the 1960s.
2. The lack of focus on architecture is bound to engender suboptimal design-decisions.
3. There is little scientific support for many of the claims made by the agile community.

4. The practices in XP are rarely applicable, and are rarely applied by the book.

5. Agile development methods are suitable for small teams, but for larger projects, other processes are more appropriate.

Agile enthusiasts often strongly resist attempts to incorporate formal methods into their approach [35]. For example, Cohen explains “If one were to ask a typical software engineer whether the Capability Maturity Model for Software and process improvement were applicable to Agile Methods, the response would most likely range from a blank stare to a hysterical laughter” because “CMM is a belief in software development as a defined process …[that] can be defined in detail, [that] algorithms can be defined, [that] results can be accurately measured, and [that] measured variations can be used to refine the processes until they are repeatable within very close tolerances. For projects with any degree of exploration at all, Agile developers just do not believe these assumptions are valid [9].”

With statements like that, the outlook for a union between XP and CMM seems bleak. However, some believe that XP and CMM can live together, if their primary functions are properly separated. For example, if we view XP as a software development methodology [36] and CMM as a software management methodology. That is, “CMM tells us what to do, while XP tells us how to do it.” As an example, NASA Langley Research Center reported a better match with CMM and Agile when the CMM part is worded generally, as in "follow a practice of choice", and not delving into specifics such as, "must have spec sheet 5 pages long [9].”

**Summary**
• Agile software development represents a major departure from traditional, plan-based approaches to software engineering.

• One of the five main criticisms of agile methodologies is the following:
  o Agile development methods are suitable for small teams, but for larger projects, other processes are more appropriate

• Agile enthusiasts often strongly resist attempts to incorporate formal methods into their approach

• Some agile methods and formal methods can be brought together if their primary functions are clearly separated.
  o For example, if we view XP as a software development methodology and CMM as a software management methodology. That is, “CMM tells us what to do, while XP tells us how to do it.”

2.3.2. Combining Agile and CMMI: Destined for Marriage, or The Odd Couple?

Recent literature [37], [38], [39], [40] comments on the practicality of combining CMMI and Agile methods, with conclusions ranging from tremendous enthusiasm to guarded caution.

Citing Scrum as one example Agile practice, Potter describes Scrum as “an example implementation of some of the Maturity Level 2 practices” noting that approximately half of CMMI’s Level 2 Generic Practices of Requirements Management, Project Planning and Project Monitoring and Control are implemented by Scrum.” Furthermore, he cautions against Scrum’s insufficiency of artifacts: “Although the practices of Scrum provide good implementation examples of many Level 2 CMMI practices, one catch is the level of artifacts needed to appraise
at CMMI Level 2.” [37] Furthermore, it is noted that Scrum begins to show gaps in when compared to CMMI Level 3 practices. Specifically, Scrum 1) does not fulfill the CMMI expectation that project data and lessons learned are shared among projects and 2) does not define engineering practices (requirements, design, implementation, verification, integration, and validation) in sufficient detail. Despite such gaps, Potter concludes that “All the remaining practices in Levels 2 and 3 can be implemented while using Scrum [37].

While Potter cautions about Scrum’s insufficiency of artifacts [37], others caution that dependency on artifact producing does not define success. “The success or failure of implementing Agile methodologies has nothing to do with documentation [38].” Instead, successfully transforming a CMMI-centered organization to adopt Agile must include communications within the organization in which corporate policies and stakeholder support are clear, sufficient training is provided, and clear accountability is defined. Regardless, the merger of Agile and CMMI in [38] is again isolated to a subset of the Level 2 and Level 3 practices.

In his comparison of CMMI and Agile, Glazer finds historical similarities in faulting both CMMI and Agile for failures: “Failures with Agile often have similar causes to those in CMMI: failing to account for context, background, and culture, and implementing incomplete components [39].” However, the study points out one of the tenets of Agile, which, although suitable for the consideration of an Agile + CMMI merger, is a tenet which is quite converse to that of mission-critical systems development. That Agile tenet is “fail early and often [39].”

Returning to our premise that the primary functions must be kept separated of “what” to do and “how” to do it, we must consider if a combination of Agile and CMMI will help with such separation. Glazer indirectly supports the separation of primary functions, but firmly states
that “neither CMMI nor Agile include content that replaces thorough engineering practices [39].”

Thus, while it is useful to consider how differing practices can be mapped to one another and in essence combined, it is important to remember 1) the resulting practices may still not prevent failures within the social context, and 2) such resulting methodologies are not a substitute for detailed engineering practices.

**Summary**

- Recent literature reflects on the potential to combine CMMI and Agile practices.
- Several studies [37], [38], [39], [40] give examples where CMMI and Agile practices can be combined, and show various “mapping” exercises between the two methodologies.
- It is common for some aspects of CMMI Level 2 practices, and fewer aspects of Level 3 practices, to be mapped to Agile practices.
- Nonetheless, Scrum, as one example of an Agile practice, shows some gaps when compared to CMMI Level 2 and Level 3, such as:
  - Scrum does not fulfill the CMMI expectation that project data and lessons learned are shared among projects.
  - Scrum does not define engineering practices in sufficient detail.
- Documentation and artifact production are not guarantors of Agile’s success within an organization. Instead, successfully transforming a CMMI-centered organization to adopt Agile must include:
  - Clear communications within the organization on corporate policies and stakeholder support.
Sufficient training.

- Clearly defined accountabilities.

- Failures in Agile are often similar to those within CMMI:
  - Failing to account for context, background, and culture.
  - Implementing incomplete components.

- Separation of responsibilities of “what” to do versus “how” to do it must be maintained when considering combination of Agile and CMMI practices.
  - “Neither CMMI nor Agile include content that replaces thorough engineering practices [39].”

2.3.3. Agile Methods

Some typical agile development methods are extreme programming (XP); dynamic software development method (DSDM); adaptive software development; feature-driven development; lean development; and rapid application development [18].

Dybå lists and describes the following as the main agile development methods [7]:

- Crystal methodologies - A family of methods for co-located teams of different sizes and criticality
- Dynamic software development method (DSDM) - Divides projects in three phases: pre-project, project life-cycle, and post project, and applies nine key principles.
- Feature-driven development - Combines model-driven and agile development with emphasis on initial object model, division of work in features, and iterative
design for each feature. Claims to be suitable for the development of critical systems

- **Lean software development** - An adaptation of principles from lean production, which consists of seven key principles.

- **Scrum** - Focuses on project management in situations where it is difficult to plan ahead, where feedback loops constitute the core element.

Scrum is described by the ScrumAlliance ([www.scrumalliance.com](http://www.scrumalliance.com)) as follows:

Scrum is an agile framework for completing complex projects. Scrum originally was formalized for software development projects, but works well for any complex, innovative scope of work. The possibilities are endless. The Scrum framework is deceptively simple.

The Scrum “cycle” is shown in Figure 2.12.
Garg lists the following as well-known agile software development methods [11]:

- Agile requirements modeling
- Agile modeling & agile model driven development
- Extreme programming
- Refactoring
- Pair programming
- Test-driven development
- Feature-driven development

A common criticism of agile methodologies is that it only works for collocated teams. However, multi-site development has been successfully conducted for years. “The key in multi-site development is to have full and competent teams in each location, and to make sure that the leaders in each location meet often enough to share their vision and understanding [13].”

**Summary**

- Common agile methods include:
  - Extreme programming (XP)
  - Dynamic software development method (DSDM)
  - Adaptive software development
  - Feature-driven development
  - Lean development
  - Rapid application development
- Crystal methodologies
- Scrum

Though agile methodologies are commonly criticized for being ineffective for non-collocated teams, multi-site development has been successfully conducted for years.

- The key to success in multi-site development is to:
  - have full and competent teams in each location
  - make sure that the leaders in each location meet often enough to share their vision and understanding

### 2.3.4. Rework

Agile methods are enacted with expectations of iterative elaboration of work products. Requirements are evolutionary in nature. This often results in rework. Yet, generally, rework is not always a bad outcome. Fairley provides a taxonomy which shows “the good, the bad, and the ugly” of iterative development and rework, shown in Table 2.3.

<table>
<thead>
<tr>
<th>Type of rework</th>
<th>Characteristics</th>
<th>Good, bad, or ugly?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolutionary</td>
<td>Work performed on a previous version of an evolving software product or system to enhance and add value to it</td>
<td>Good—if it adds value without violating a cost or schedule constraint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bad—if it violates a cost or schedule constraint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ugly—if it smacks of “gold plating”</td>
</tr>
<tr>
<td>Avoidable</td>
<td>Work performed on a previous version of an evolving software product or system that developers should have performed previously</td>
<td>Good—small amounts are inevitable; better now than later</td>
</tr>
<tr>
<td>Retrospective</td>
<td></td>
<td>Bad—if it occurs routinely</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ugly—if excessive, it indicates a need to revise work processes</td>
</tr>
<tr>
<td>Avoidable</td>
<td>Work performed to fix defects in the current and previous versions of an evolving software product or system</td>
<td>Good—if total rework is within control limits</td>
</tr>
<tr>
<td>Corrective</td>
<td></td>
<td>Bad—if it results in patterns of special-cause effects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ugly—if it results in an out-of-control development process</td>
</tr>
</tbody>
</table>
Table 2.3 – Fairley’s iterative rework taxonomy [8]

2.3.5. Adopting Agile Methods for Mission-Critical Systems Development

Common agreement does not exist regarding which agile practices, if any, should be practiced in mission-critical and safety-critical systems development.

There are some strong supporters who believe that mission-critical applications development should take advantage of the many improvements agile practices can bring to software development processes [41] [42] [43]. “Today in the field of military embedded systems, developers face the pressure to shorten the life cycle and adapt to the changing requirements without forgoing the quality and reliability required for the safety critical applications. This requirement has led the thinking in the direction of Agile approaches [11].” Candidate agile methods should consider concepts like change management, traceability, test driven development and automated documentation. “The principles of the Agile Manifesto may be the necessary framework required for flexible and fast development for non-software engineering programs given they include vital systems management and systems engineering processes [10].”

Examples of Agile practices which may be evaluated in terms of meeting organizational objectives and dependencies of the practices would include:

- Test-driven development
- Pair programming
- Automated unit tests
- Daily standup meetings
- Collaborative planning
According to Pikkarainen, use of agile methods and practices improved communications and management of requirements [21].

However, not all agile practices are applicable to large, complex systems, those with long development periods, and other characteristics of mission and life-critical systems. In fact, many are not suited for such systems development at all. For those considering adopting an agile process, Cockburn warns: As system criticality increases, there is decreased “tolerance for personal stylistic variations [9].”

Thus, great care must be taken care in determining which all agile practices to adopt within organizations that develop such systems. Yet, it is “not a question of if agile methods can be used in the safety-critical world, but rather a question of how [11].”

It is important to understand the prescriptive characteristics of various agile practices and their applicability to system criticality. Cohen provides insight, as shown in Table 2.4 [9].

<table>
<thead>
<tr>
<th></th>
<th>XP</th>
<th>Scrum</th>
<th>Crystal</th>
<th>FDD</th>
<th>LD</th>
<th>DSDM</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team Size</td>
<td>2-10</td>
<td>1-7</td>
<td>Variable</td>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration Length</td>
<td>2 weeks</td>
<td>4 weeks</td>
<td>&lt; 4 months</td>
<td>&lt; 2 weeks</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed Support</td>
<td>No</td>
<td>Adaptable</td>
<td>Yes</td>
<td>Adaptable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Criticality</td>
<td>Adaptable</td>
<td>Adaptable</td>
<td>All types</td>
<td>Adaptable</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4 – Prescriptive characteristics of various agile practices [9]

Which Agile methodologies, then, are suitable for software development of critical systems? What about, for example, XP? There is nothing in XP itself that should limit its
applicability [9]. Thus, since XP is not necessarily geared for one system or another, it is worthwhile to consider it for critical-systems development.

Mission and life-critical systems are characterized as being large, complex, requiring long development cycles, and having specific desirable system performance qualities which affect the suitability of agile practices to such development. For example, “Maintainability and Reliability are common desired system qualities for mission and life-critical systems [5].”

A common characteristic found in mission and life-critical systems is software complexity. When adopting the evolutionary requirements approach the design of the system also becomes evolutionary since not all the requirements are known ahead of time. Agile development addresses the issue of continuously developing and improving a system’s design by the use of another practice – Refactoring. Refactoring involves rewriting the code to improve its structure, while explicitly preserving its behavior. Therefore, regularly refactoring the code is necessary if evolutionary requirements are adopted [5]. In mission and life-critical systems refactoring is a risky and costly task due to size and complexity of the systems.

A widespread criticism of Agile Methods is that they do not work for systems that have criticality, reliability and safety requirements. High reliability is a desired system quality of mission and life-critical systems because it helps promote safety. One of the approaches used to attain high reliability in a system is the use of Independent Verification and Validation (IV&V). One activity within the IV&V of mission and life-critical systems is a Safety Impact Analysis that is conducted before the design and coding phases. For a safety impact analysis to be conducted, all requirements must be known before any actual development occurs. This
precludes the use of agile, since the evolutionary nature of agile methods doesn’t identify all the requirements a priori [5].

Furthermore, projects developed with agile practices such as XP can adhere to strict (or safety) requirements, since all tests have to be passed before release. In these cases, the agile emphasis on testing, particularly the test-driven development practice of XP, is the key to working with these projects.

Regardless, others, argue that Agile best fits applications that can be built “bare bones” very quickly, especially applications that spend most of their lifetime in maintenance [9] [44].

Potter provides a related study in the applicability of implementing one Agile practice, Scrum, and CMMI together [37] in which he shows a mapping of numerous CMMI Level 2 Specific Practices to Scrum techniques, and further concludes that “all the remaining practices in Levels 2 and 3 can be implemented while using Scrum.” Another study similarly shows successful mapping of some CMMI Level 2 practices, including General Practices, to various Agile methodologies [40]. These studies give good examples of the ability to very carefully, purposefully, and selectively map agile practices to other established methodologies.

Nonetheless, reliable and safety-critical projects can be conducted using agile methods. Performance requirements must be made explicit early, and proper levels of testing must be planned. It is easier to address critical issues using Agile Methods since the customer gives requirements, sets explicit priorities early and provides continual input [9]. Others believe that agile practices can be employed if the safety analysis process measurement is statistically modeled as a set of quantitative measures, because of the repeatability of the analysis [18]. “A
A culture of high reliability at low cost is possible if the proper techniques are adopted as suggested by the principles of the Agile Manifesto [10].”

In mission and life-critical systems, adopting an inappropriate agile practice has detrimental impacts on the system in various phases of its lifecycle as well as precludes desired qualities from being actualized. “For organizations developing mission and life-critical systems, adopting all agile practices is not a viable option because some agile practices are simply not suitable [5].

For proper process consideration, emphasis must be placed on separating codified knowledge (known and repeatable) versus tribal knowledge (unknown, and being discovered) [10]. Whereas repeatable processes benefit most from formal methodologies and engineering standards, agile methods are more suited for knowledge discovery.

For systems having a mix of plan-driven and agile characteristics, hybrid agile/plan-driven approaches are feasible. Garg gives the following aspects of safety critical systems which warrant consideration of adding agile practices to plan-driven approaches [11]:

- Co-design of hardware and software
- Reliability and safety impact analysis
- Preliminary architecture design
- Refactoring
- Configuration control
- Traceability
- Automated documentation
When considering the applicability of agile practices to the systems development, care must be taken in determining if adopting an agile practice conflicts with any system/development characteristic or precludes any desired system quality of mission and life-critical systems. If so, then the practice is considered unsuitable. With this in mind, Sidky gives a three-stage process (Figure 2.13) on how to identify the agile practices that can benefit mission and life critical systems development [5]:

Figure 2.13 – Sidky’s three-stage process on agile process suitability [5]

- Stage 1: Making the Go/No-go decision
- Stage 2: Discarding inappropriate practices
- Stage 3: Determining the right practices to adopt

As an example of the suitability assessment of an agile practice to critical systems development, Sidky considers minimal documentation. Minimal documentation assumes that enhanced informal communication among team members, known as tacit knowledge, is the primary means of knowledge exchange, not formal documents. As efficient as minimal
documentation is as an agile practice, its adoption in mission and life-critical systems is not appropriate for the following two reasons [5]:

1. The typically long (multiyear) length of the development process leads to personnel turnover and re-allocations over the lifetime of the project. Thus technical information for training and support needs to be provided by means of explicit, comprehensive, and persistent documentation.

2. Mission and life-critical systems have maintainability as a highly desirable system quality. Since systems maintenance is usually handled by third-party organizations relying on tacit knowledge to communicate and transfer technical information is not a viable option.

As another example of unsuitable agile methodologies, we consider Evolutionary Requirements, a practice that suggests that requirements should evolve over the course of many iterations rather than being fully developed in a major upfront specification effort. This practice is not suitable for mission and life-critical systems for two reasons: 1) mission critical systems require a safety impact analysis which in turn requires all requirements to be known upfront, and 2) system design also becomes evolutionary, resulting in regular refactoring, which is risky and costly to critical systems development [5].

**Summary**

- Mission-critical applications development should take advantage of the many improvements agile practices can bring to software development processes.
- Not all agile practices are applicable to large, complex systems, those with long development periods, and other characteristics of mission and life-critical systems.
• It is “not a question of if agile methods can be used in the safety-critical world, but rather a question of how”

• XP is one example of an agile practice, which is not necessarily geared for one system or another, and is therefore worthwhile to consider it for critical-systems development.

• For systems having a mix of plan-driven and agile characteristics, hybrid agile/plan-driven approaches are feasible.

• Some feel that reliable and safety-critical projects can be conducted using Agile Methods, if performance requirements are made explicit early, and proper levels of testing are planned.
  o A Safety Impact Analysis requires all requirements be known before any actual development occurs, and this precludes the application of agile methods, since the evolutionary nature of agile methods doesn’t identify all the requirements a priori.
  o However, others believe that agile practices can be employed if the safety analysis process measurement is statistically modeled as a set of quantitative measures, because of the repeatability of the analysis

• In mission and life-critical systems, adopting an inappropriate agile practice has detrimental impacts on the system in various phases of its lifecycle as well as precludes desired qualities from being actualized.

• To assist in the determination of applicability of a candidate agile practice, emphasis must be placed on separating codified knowledge (known and repeatable) versus tribal knowledge (unknown, and being discovered) [10].
Whereas repeatable processes benefit most from formal methodologies and engineering standards, agile methods are more suited for knowledge discovery.

- Sidky gives a three-stage process on how to identify the agile practices that can benefit mission and life critical systems development:
  - Stage 1: Making the Go/No-go decision
  - Stage 2: Discarding inappropriate practices
  - Stage 3: Determining the right practices to adopt

### 3. Discussion and Analysis

Mission-critical systems have key characteristics of mission uniqueness and success objectives, which distinguish their development from other traditional systems. As a subset of mission-critical systems, space-based systems include mission management, spacecraft operations, and ground system operations, each of which is composed of multiple, interrelated, complex sub-systems. A failure within any one of these component areas can lead to a failure of the entire mission. Thus, high software development rigor is required to meet the high reliability requirements and strict safety and performance criteria.

Development of highly complex systems presents special challenges in many areas including: requirements management; dependability estimation; architecture and design; testability and fault tolerance; sub-system integration; and overall project management and coordination. Therefore, methodologies and processes which are used in the development of mission-critical systems must be especially effective and efficient in their ability to resolve a
wide range of potential issues across cultural differences between various sub-contractors, where each autonomous agent practices different methodologies and relies on different standards.

To achieve safety and reliability goals, critical systems development often uses formal methods and favors redundancy. Two examples of formal methodologies are Capability Maturity Model Integration (CMMI) and NASA’s systems engineering engine; the latter is a systems engineering approach specific to NASA development. Both contain numerous process areas documented in great detail, as do many other methodologies and standards that are employed in space-systems development.

Traditional space-systems development approaches follow a strict waterfall approach, in which functional decomposition is the driving force, and over-emphasize tools, technologies, and standards instead. For example, typical flight software development practices fail to stress the importance of hardware and software interdependencies. Hardware design decisions are often made without regard for the impact on software development costs and flight software development typically does not begin until after most of the hardware design decisions are done. Similarly, the preferred choice in spacecraft software programming language has included assembly, Fortran, and some Ada, and C. C++ is used rarely, although it has gained some popularity recently. Nicewarner goes so far as to suggest that there is sub-optimal development productivity in space software development, which results from the industry’s reluctance to use modern languages and processes [3].

True failure root-cause analysis, focusing on the mission-critical systems failures resulting from the social context mismanagement, has only recently come to light. For example, NASA enacted a program called Faster, Better, Cheaper (FBC) with the aim to streamline
(faster) mission development under extremely restricted budget (cheaper). Unfortunately, NASA management abandoned the processes associated with formal systems management and fatally neglected to institute teamwork methodologies. Although NASA tried to replace formal process with teamwork and tribal knowledge systems, ultimately several FBC programs “failed because they neglected to adopt necessary practices in systems management.” The result was 6 of 16 missions (37.5%) failed, including 4 out of 5 (80%) in a single year (1999).

While failure effects and proximate causes are often technical, the root causes are mainly due to social and psychological impact resulting from ineffective communications environments. This can lead to human error. Improving social context management (e.g. the environment and culture of humans working together) is predicated on the criticality of managing the story-line, or narrative, of the work being done, and is the key to successfully turning around failing projects and making them effective.

NASA seems partially to have had the right idea with FBC. They wanted a process different from the traditional ones, and they relied (too much so) in the belief that team based management can assure reliability and mission success. However, critical flaws in the new way of thinking occurred when structured process was abandoned, and effective communications broke down.

Recent events in the space sector are leading nations, academics, companies, and other organization to rethink exactly how space exploration will proceed in the future. Interested parties are shifting towards a globalized, cooperative approach to pursuing space. Key business drivers are leading new space age development approaches to systems development by employing small teams that evolve design in small rapid steps (build, test, correct, …).
To foster and promote environments of cross-cultural cooperation, management must use strategies such as open communications, fostering creativity, and dividing programs into less complex work packages with shorter durations. However, organizational changes required to achieve these goals cannot be immediately put in place and must proceed from old methodologies to new in a “stepwise implementation.”

One recent entrant into the space exploration industry is Space Exploration Technologies (SpaceX), who emphasizes reusability and reliability (repeatability), with the goal of drastically reducing the costs of space access.

SpaceX takes several interesting approaches to the discipline of space exploration technologies, allowing them to achieve success cost-efficiently while maintaining a proper balance of risk management. For example, SpaceX willingly embraces the usage and customization of third-party products, when such use “meets very strict cost-benefit trades.” Also, to preserve system reliability expectations, the Falcon 1 rocket was designed with nine Merlin 1C engines powering the first stage so that if one engine fails, it does not necessarily translate into total rocket launch failure. SpaceX embraces new approaches, such as an ability to build fast-moving and efficient teams and organizations geared around a core mission and competency. This methodology extends into the cultural objectives and process areas within the organization, including the need for cooperation among the scientists, technicians, and programmers themselves.

When considering the drivers for change within space flight development, and without discounting the traditional development approaches which have been employed in the past, the
question remains if agile methodologies can be used effectively in space systems development. There is much disagreement around this question.

The foundations of agile approaches emanates from the Agile Software Development Manifesto, which stresses four main values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile recognizes that people are an important aspect to any project, and recognizes that software development is a fluid activity that requires people to practice reflection. However, agile does not resolve all the misfortunes experienced within software development organizations and agile methodologies do not fix the inherent problems of a team. These seemingly contradictory viewpoints form the basis of the disagreement about the suitability of agile methodologies to mission-critical systems development. That is, agile has strengths which can help address the social context concerns of critical-systems root cause failures, but agile methodologies alone are not a “magic bullet” that can fix all organizational problems. In fact, there is agreement that several specific agile methodologies are not suited for critical-systems at all.

The notion that agile focuses on people and teamwork, and emphasizes communications over process is a compelling one, especially within space systems development. Failures within the social context management are seem as the primary causes of mission-critical failures, and so attempts which truly improve collaboration and focus on people would certainly be helpful.
However, NASA’s FBC program serves as reminder that merely believing in team-based management is not sufficient to eliminate failure. Much more than that is needed.

Any consideration of agile practices can prove quite challenging, and this seems especially true within space exploration development. Agile software development represents a major departure from traditional, plan-based approaches to software engineering. This fact alone presents significant hurdles within mission-critical systems development, where traditional policies of “institutional inertia and risk aversion” prevent adoption and consideration of anything which strays from tried and true practices of accepted formal methodologies. Furthermore, agile enthusiasts often strongly resist attempts to incorporate formal methods into their approach. It seems that, within the space systems development, traditional methodologies and agile practices stand at polar opposites, destined to forever repel one another.

Nonetheless, some agile methods and formal methods can coexist, even within mission-critical space development. The key to their successful marriage is to ensure that the primary function of each is clearly separated. For example, we may view XP as a software development methodology and CMMI as a software management methodology. That is, “CMMI tells us what to do, while XP tells us how to do it.”

Several recent studies [37], [38], [39], [40] demonstrate the potential for combining CMMI and Agile practices, as least for certain CMMI Level 2 and Level 3 practices, while still cautioning that such resulting practices are still not a substitute for detailed engineering practices, and may not themselves prevent failures within the social context.

Still, this concept of separation of function is, in fact, vital to the successful integration of agile with formal methods. For proper process consideration, emphasis must be placed on
separating codified knowledge (known and repeatable) versus tribal knowledge (unknown, and being discovered) [10]. Whereas repeatable processes benefit most from formal methodologies and engineering standards, agile methods are more suited for knowledge discovery. If we were to let agile methods tell us what to do, instead of the formal engineering standards, then we would be setting ourselves up for disaster.

There are numerous agile practices, and the literature does not entirely agree within the enumeration of such practices as to which are, in fact, “agile”, nor does it agree which are the most meaningful to consider. Evaluation of which agile practices may be considered must be done on a case-base-case basis and must take into account the organizational culture, structures and policies, and tolerance for risk and change. Sidky gives a three-stage process on how to identify the agile practices that can benefit mission and life critical systems development:

- Stage 1: Making the Go/No-go decision
- Stage 2: Discarding inappropriate practices
- Stage 3: Determining the right practices to adopt

Sidky’s three-stage process is quite appealing. It forces recognition that some agile processes must be discarded outright, before even considering which practices may even be considered for adoption. The Sidky process is non-prescriptive, and, in fact, general.

Recent studies describe the possibilities of combining together CMMI and Agile practices [37] [40], showing mappings or “coverage” of certain CMMI Level 2 and Level 3 practices by Agile techniques. While not directly applying the Sidky process, these studies give good examples of the ability to very carefully, purposefully, and selectively map agile practices to other established methodologies. However, these studies have treated the combination of
CMMI and Agile practices rather generically, and have been limited to a subset of CMMI Level 2 and Level 3 activities. While some caveats and gaps of Agile are identified, it is insufficient research from which to conclude that some combination of CMMI and Agile would be suitable to mission-critical systems development. Thus, while such research is insightful and promising, great care must be taken when considering the application within critical systems development of Agile practices to CMMI practices, or to any other methodologies and engineering standards for that matter.

For systems having a mix of plan-driven and agile characteristics, hybrid agile/plan-driven approaches are feasible. Mission-critical space systems certainly fit this category. Development areas which rely on known, repeatable, codified processes and engineering standards should obviously retain their formal methodologies and standards. That is, the “what to do” must be left to established standards and formal process which is known to work. However, there is opportunity to explore the application of agile methods to the “how to do it” aspects of development, especially when the “how” is unknown, and a discovery process is warranted.

Nonetheless, it does seem that some agile practices can indeed be discarded outright within space-based systems engineering. For example, evolutionary requirements, minimal documentation, and refactoring are agile methods which are not appropriate for adoption within space-based systems development.

However, several agile methods warrant strong consideration, including: Test-Driven Development, Extreme Programming (XP), and Scrum. Furthermore, the iterative and cyclical
nature of typical agile work cycles make many practices ideal for addressing project critical issues, and safety analysis measurement. Each of these will be explored briefly below:

Because mission-critical space-based systems have strict safety requirements, all established testing scenarios must pass. This easily lends itself to the Test-Driven Development practice of agile, where system testability requirements drive the work.

Nothing about space-based systems development precludes the use of XP. In fact, given the highly technical, complex nature of system elements, and given the resulting failures which have come out of NASA’s FBC, it may be a useful practice to co-develop, especially critical components. Furthermore, XP may be useful to bring together those who in the past would not have worked side-by-side. For example: testers and developers, ‘flight’ and ‘ground’ developers, analysts and implementers, etc.

As an agile practice, Scrum’s specific applicability to mission-critical systems engineering is rarely mentioned within the literature sample identified for this research paper [45]. Thus, Scrum presents several interesting opportunities. It is proposed that Scrum concepts may be suitable for the following situations:

1. Daily stand-up meetings for managing day-to-day work
2. Used as an over-arching project management strategy for sub-contractor coordination activities.
4. Safety analysis, when the results can be measured and the assessment process is repeatable.
The Scrum style helps foster inter-personal communications, which would likely be helpful in coordinating highly technical work, and ensuring the social context of the story-line is understood and carried out. The concept of the 15-minute daily meetings could be useful for managing day-to-day work, while the length of the iteration cycle could be adjusted to an appropriate value for the team’s activities.

Furthermore, Scrum could be used as an over-arching project management strategy for coordinating activities across sub-contractors or as systems integration work is underway. In fact, this would also each sprint to be defined to include those team members actively involved in the current, active systems integration activities. That is, it would help keep unnecessary interference out of the way from management which is not vitally involved in work at hand.

Additionally, Scrum may prove useful to managing critical issues and risks. The assumption that critical issues and risk lists are regularly prioritized and worked on in priority order fits nicely with the naturally iterative cycle of prioritize-plan-work that Scrum follows.

While safety analysis is often considered a non-ideal candidate for agile methods, there is potential for application of agile methods when the safety analysis results can be quantitatively measured through a structured (repeatable) process. In such cases, the iteratively repeatable nature of using Scrum would be quite applicable, and would also be so for any other repeatable, quantifiable processes. Regardless, Scrum does not solve all ailments of any project. To achieve the desired level of dependability - fault prevention, fault tolerance, fault removal, and fault forecasting - development processes must combine all possible methods.

Use of Scrum in mission-critical systems, especially those like space-based development, would require special attention to make sure it effectively used. This is especially
true if teams are not co-located and if the team composition varies from iteration to iteration. Specifically, the daily meeting question of “are there any obstacles that stand in the way of your reaching your goal” would need to be very closely monitored. A process in which individual team members either do not recognize major blocking issues, or are not comfortable reporting them could lead to disaster if that were to be allowed during development of any mission-critical system. That is, this is specifically the place in the Scrum process where great care would need to be spent to make sure that a culture to under-report issues, risks, problems, obstacles – whatever you want to call them – is not allowed to take root.

The most appealing way in which agile practices may be employed within space-based systems engineering involves consideration of hybrid approaches. Agile methods may be intertwined with other proven development processes, engineering standards, and formal methodologies. In such a combined approach, care would need to be taken in separating the duties of agile versus the other processes. As discussed previously, a good fit is one in which formal methodologies and engineering standards are used for known and repeatable processes, whereas agile practices can be geared more towards areas requiring knowledge discovery.

4. Relevant Work

Because of the key drivers which are impacting organizations involved in space exploration and development, those organizations are reacting by changing the ways in which they create their products and solutions. Thus, this research paper is relevant to such organizations, as they give consideration to other methodologies and processes, especially agile practices. Furthermore, this research paper has relevancy also to potential extensions to some of the research referenced within the literature.
First, the space-based development industry continues to undergo changes due to socioeconomic reasons. The emergence into the field and success of commercial ventures like SpaceX supports the notion of these evolving changes within the industry. SpaceX’s focus on cost reduction and emphasis on repeatability has led them to embrace new approaches and build a culture centered on fast-moving and efficient teams geared around a core mission and competency. The timing seems right for finding ways to apply agile methodologies within SpaceX and other space exploration ventures.

The stereotype applied to NASA of being risk-averse, coupled with the failures within FBC indicates a reception to needing to refine and tune development methodologies and processes. Pellerin’s analysis on the root causes of failure being within the social context, and his resulting book and coursework on “How NASA Builds Teams” stresses the important of human interactions, and again, agile practices seem relevant.

Sidky’s three-stage process for determining which agile methodologies are useful within the culture of a mission-critical development organization remains quite interesting. However as it is a general framework, there is little detail within the literature on specific application of agile selection processes within specific space-based exploration organizations.

Published processes and standards, like CMMI and NASA’s systems engineering engine, will continue to be refined. Research on the important of architecture to space-based systems will continue. Consideration will be given to COTS sub-systems, which may soon be developed by any of the field of sub-contractors. All of these reasons will heighten within every related organization the need for considering process by which such changes must be incorporate within
the organization, and study of the potential for using agile practices to assist these endeavors seems prudent.

5. Future Work

Future work resulting from this research paper has potential for both breadth and depth in further research and analysis. There are opportunities in:

1. Extending the research and analysis to other domains aside from space exploration, to understanding if and how the application of agile practices to other domain has any direct applicability to the space exploration domain.

2. Researching specific models which may combine agile with other specific formal methodologies or engineering standards, such as CMMI or NASA’s system engineering engine.

3. Applying Sidky’s three-step process (or one similar to it) to various formal methodologies and standards (e.g. beyond the CMMI mapping studies represented herein) to understand more about which specific agile practices may be used supportively within a much broader set of such methodologies and standards.

4. Documenting case studies from within space exploration organizations which have either tried, or desire to try, agile practices, and further identifying the strengths and weaknesses of practices as they have been attempted within the field.

Admittedly, this research paper narrowed the focus of mission and safety critical systems to an exploration of space-based systems, and has thus given consideration to the applicability of
agile techniques to this specific domain. Breadth of research could be examined by lifting such an arbitrary restriction and conducting much broader research into other domains such as medical devices, military equipment, and nuclear power systems. It would be useful to reflect on how agile practices are applied to their other domains, and if so, if there are any lessons-learned which are transferrable to the space domain. However, it would be necessary to note that other critical-system domains like medical devices have high certification requirements and strong government oversight. Thus, such future work would still need to be tailored to specific needs of each industry, and of course, the systems engineering practices associated with each industry would vary accordingly.

Detailed studies have not be found that describe the potential combined application of agile practices with other common formal methods and engineering standards, such NASA’s systems engineering engine. Although Sidky outlined a three-step process for an organization’s evaluation of agile practice [5], the literature does not provide sufficient detailed examples of where this process, or ones similar to it, have been explicitly applied to the formal methodologies and engineering processes in place within the space exploration industry. In fact, the Sidky process is non-prescriptive and general. Thus, it would be very beneficial to develop a detailed, specific process framework by which existing methodologies and standards could be reviewed and assessed with respect to agile practices, resulting with a formal mapping of each formal process area to candidate agile practices, with an assessment of the usefulness of each agile practice to each formal process area. Such a mapping process could then be applied to any existing process area within any space-based exploration company, with a resulting assessment, or report delivered. A brief sketch of this idea is presented in Table 5.1. This would allow each player within the industry to be able to best answer an organization-specific question: “How
compatible are agile practices to the existing formal methodologies and engineering standards that are currently practiced within my organization?” It would be beneficial to have a repeatable process and formalism of analysis that produced consistent results, and would fill the gap of such analysis which is missing from the industry.
### Corporation ABC

**Suitability of agile practices to a formal methodology XYZ**

**Suitability scale:** 1 (no fit) to 10 (best fit)

<table>
<thead>
<tr>
<th>(XYZ) Process Area</th>
<th>Agile Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Area 1</td>
<td>3</td>
</tr>
<tr>
<td>Process Area 1.1</td>
<td>4 Needs stakeholder re-alignment</td>
</tr>
<tr>
<td>Process Area 1.2</td>
<td>3</td>
</tr>
<tr>
<td>Process Area 1.2.1</td>
<td>1</td>
</tr>
<tr>
<td>Process Area 2</td>
<td>1</td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.1 – Corporation ABC: suitability of agile practices to formal methodology XYZ**
Similarly, the evaluation process could be restricted to the analysis of a specific formal practice within a specific organization. For example: “Is extreme programming applicable to the RLV development within SpaceX?” or “Is scrum applicable to the subcontractor integration processes of sub-systems within Lockheed Martin?”

Lastly, specific case study evaluations are very sparse within the industry regarding organizations who have attempted, or desire to leverage agile methodologies. This is due to two reasons. First, the industry has continued reliance on the use of established practices and standards and is risk-averse to change, especially considering that mission critical systems engineering views agile practices as new, and somewhat unproven. That is, there is continued practice of “don’t fix what isn’t broken” and to adhere to the organization’s established process framework. Second, the output of the industry is highly technical, usually proprietary, and, in fact, often top secret. Thus, the competitive nature of the industry provides little incentive for any organization to share the process by which their products are produced. Processes, especially the highly technical ones, are cherished by the organization and highly protected. Nonetheless, this produces a ripe environment from which specific case studies must be studied, so that a database, or “catalog”, of lessons learned could be collected. From there, subsequent analysis could lead us to more generalized recommendations on the successes and pitfalls of agile practices within the space exploration industry. Such results would take us from the lessons-learned of individual teams within organizations and provide recommendations which benefit the industry as a whole.
6. Conclusion

Mission critical systems engineering is especially complex, driven by high safety and performance requirements. Space exploration is one such development domain in which mission criticality defines success. There is often only one chance for mission success, and failure can be fatal.

Despite the application of various formal methodologies and engineering standards within the space exploration industry, failures occur. Although proximate causes are often technical, root causes of failure are usually mismanagement of the social context, leading to human error.

Key drivers of the industry, such as budgetary cuts and globalization of the industry, are leading the development organizations to continue to focus on safety and performance, while considering different ways of achieving high-quality results while balancing cost-constraints. Some attempts, such as NASA’s Faster, Better, Cheaper (FBC) had good intentions, but failed to manage the social context. SpaceX’s recent arrival into the space exploration industry, and initial successes, signals the need for strong consideration re-establishing the process baseline within the industry.

Agile practices are relatively new. While there is some limited research in the applicability of agile practices within mission-critical systems engineering, there is not much supporting data with respect to specific applicability of practices within the space industry.

While some agile practices may be discarded outright, others seem quite useful and should be given consideration. A hybrid approach to combining agile practices with existing formal methodologies and engineering standards should be practiced. This presents several
opportunities for future work: case studies should be performed, mappings of specific formal processes to agile practices should be completed, and database of lessons learned should be established, from which recommendations that benefit the industry as a whole could be made.
References


Biography

Scott Carpenter is an Information Analyst and Agile Project Manager at Houghton Associates, Inc., and a Ph.D. student in the Department of Computer Science at North Carolina State University. In his current working roles, Mr. Carpenter analyzes a wide variety of technical problems in aviation and aerospace, much of which is conducted at the request of the Federal Aviation Administration (FAA) and involves cost-benefit analysis of complex systems.

Prior to his recent position, Mr. Carpenter managed co-located and separated software development teams at Charles Schwab, Smith-Breeden, and Fidelity Investments. Furthermore, Mr. Carpenter has over 20 years of technical software engineering experience in computer graphics, data communications, protocol analysis, pharmaceutical, code analysis, finance, and aviation/aerospace. He specializes in software development using Java, C++, and SQL, and process management and improvement. He has also worked for IBM, Attachmate, and Quintiles.

Mr. Carpenter achieved his Project Management Professional (PMP) certification in 2001, and is also a Certified Scrum Master (CSM). He has successfully managed numerous projects, using a variety of processes, tools, and techniques. Mr. Carpenter believes that strong team-based collaboration is very important when producing strong technical solutions. He currently favors agile practices, which he has championed and established within several organizations.

Mr. Carpenter earned his BS degree in computer engineering from Case Western Reserve University in 1989, and his MS degree, also in computer engineering, from Case Western
Reserve University in 1993. His master’s project involved an artificial intelligence (AI) based tutoring system. Mr. Carpenter’s interests include: autonomous vehicles, intelligent agents, stochastic simulation, software engineering, business analysis, project management optimizations, and predictive analytical analysis of the art market.

Mr. Scott E. Carpenter was named in honor of the American astronaut, Mr. Scott M. Carpenter, who was selected as one of the original seven Mercury Astronauts. Although they share no direct family history, Mr. Carpenter strongly admires his namesake and expresses his deepest thanks to his parents, as the family names being handed down to sons prior to his birth were primarily limited to “Clarence”, “Eugene”, and “Earl”. Considering that he may have thus been given one of those names, Scott likes very much the name that he was ultimately given.