Nurturing Systems Thinking: Developing a Framework Based on the Human Factors Analysis and Classification System (HFACS) to Improve Systems Development Processes

Arjun Vijayanarayanan

Follow this and additional works at: https://commons.erau.edu/edt

Part of the Aviation Safety and Security Commons, Ergonomics Commons, and the Systems Engineering Commons

Scholarly Commons Citation
https://commons.erau.edu/edt/144

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.
Nurturing Systems Thinking: Developing a framework based on the Human Factors Analysis and Classification System (HFACS) to Improve Systems Development Processes

by

Arjun Vijayanarayanan

A Graduate Thesis Submitted to the

Department of Human Factors and Systems

In Partial Fulfillment of the Requirement for the Degree of

Master of Science in Human Factors and Systems

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Spring 2011
Nurturing Systems Thinking: Developing a Framework Based on Human Factors Analysis and Classification System (HFACS) to Improve Systems Development Processes

by

Arjun Vijayanarayanan

This thesis was prepared under the direction of the candidate’s thesis committee chair, Kelly J. Neville, Ph.D., Department of Human Factors and Systems, and has been approved by the members of the thesis committee. It was submitted to the Department of Human Factors and Systems and has been accepted in partial fulfillment of the requirements for the degree of Master of Science in Human Factors and Systems.

THESIS COMMITTEE:

Kelly J. Neville, Ph.D., Chair

Albert J. Boquet, Ph.D., Member

Scott A. Shappell, Ph.D., Member

MS HFS Program Coordinator

Department Chair, Department of Human Factors and Systems

Associate Vice President for Academics
# Table of Contents

Introduction .......................................................................................................................... 1

The Longwall Mining Case Study ......................................................................................... 5

Goguen on How to Address the Social Aspects of Systems ............................................... 8

Systems Engineering Quality Improvement ......................................................................... 10

Approaches for Achieving Quality in Systems Engineering ............................................... 11

The waterfall model .............................................................................................................. 13

The Spiral development model ............................................................................................. 14

Capability Maturity Model Integration® (CMMI). ................................................................. 17

Limitations of traditional approaches to quality assurance ................................................. 21

Systems Thinking Frameworks in Other Domains .............................................................. 22

Reason’s “Swiss Cheese” Model of human error causation. ................................................. 23

Human Factors Analysis and Classification System. .......................................................... 24

Approach .............................................................................................................................. 33

Method .................................................................................................................................. 34

Overview ............................................................................................................................... 34

Participants ............................................................................................................................ 35

Knowledge elicitation approach. ......................................................................................... 35

Data Analysis ......................................................................................................................... 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coders</td>
<td>37</td>
</tr>
<tr>
<td>Data-driven coding</td>
<td>37</td>
</tr>
<tr>
<td>Reliability assessment</td>
<td>40</td>
</tr>
<tr>
<td>Results and Discussion</td>
<td>41</td>
</tr>
<tr>
<td>Correspondence Between Coders</td>
<td>41</td>
</tr>
<tr>
<td>Data-Driven Coding</td>
<td>42</td>
</tr>
<tr>
<td>The Systems Thinking Framework (STF)</td>
<td>44</td>
</tr>
<tr>
<td>Organizational influences</td>
<td>47</td>
</tr>
<tr>
<td>Project management</td>
<td>49</td>
</tr>
<tr>
<td>Team processes</td>
<td>52</td>
</tr>
<tr>
<td>Technical activities</td>
<td>55</td>
</tr>
<tr>
<td>External pressures</td>
<td>60</td>
</tr>
<tr>
<td>Factors to look out for</td>
<td>61</td>
</tr>
<tr>
<td>Encouraging Attention to Human Aspects of Systems</td>
<td>62</td>
</tr>
<tr>
<td>Conclusion</td>
<td>65</td>
</tr>
<tr>
<td>References</td>
<td>66</td>
</tr>
<tr>
<td>Appendix A: Description of the various categories in HFACS</td>
<td>71</td>
</tr>
<tr>
<td>Appendix B: Retrospective Interview Questions</td>
<td>80</td>
</tr>
<tr>
<td>Appendix C: Coded Interview Transcripts</td>
<td>84</td>
</tr>
</tbody>
</table>
Appendix D: Calculation of Percent of Correspondence between Coders.......................... 103

Appendix E: Taxonomy of Team Processes .................................................................. 106

Appendix D: Team Dimensional Training Prebrief......................................................... 108

Appendix G: Description of the final set of data-driven codes................................. 109
List of Tables

Table 1. The Standish Group’s “CHAOS Report” ................................................................. 1
Table 2. CMMI Framework .................................................................................................. 19
Table 3. Social Psychological Influences Assessed by Paletz et al. ...................................... 28
Table 4. Characteristics of the System Development Projects Described by the Participants ... 36
Table 5. Interview Transcripts as Data Elements ................................................................. 38
Table 6. Categories of HFACS Used to Code Data ............................................................. 39
Table 7. Examples of Data Elements Coded Using HFACS Categories ............................... 40
Table 8. Correspondence Percentages ................................................................................ 41
Table 9. Initial Set of Codes ............................................................................................... 42
Table 10. Coded Interview Transcripts .............................................................................. 84
Table 11. Process Used for the Calculation of Percent of Correspondence between Coders ... 103
Table 12. Taxonomy of Team Processes ........................................................................... 106
Table 13. Team Dimensional Training Prebrief .................................................................. 108
Table 14. Final Set of Data-Driven Codes with Description ............................................... 109
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The waterfall model</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>The spiral development model</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>CMMI maturity levels</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>The “Swiss cheese” model of human error causation</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>The HFACS framework</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>Inclusion of self-motives and progressive commitment in the HFACS framework</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>Inclusion of self-motives and progressive commitment in the HFACS framework</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>The STF structure</td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>Organizational influences</td>
<td>47</td>
</tr>
<tr>
<td>10</td>
<td>System development project management</td>
<td>50</td>
</tr>
<tr>
<td>11</td>
<td>Team processes</td>
<td>53</td>
</tr>
<tr>
<td>12</td>
<td>System development technical activities</td>
<td>56</td>
</tr>
<tr>
<td>13</td>
<td>Categories of unsafe acts of operators</td>
<td>71</td>
</tr>
<tr>
<td>14</td>
<td>Categories for preconditions for unsafe acts</td>
<td>73</td>
</tr>
<tr>
<td>15</td>
<td>Categories for unsafe supervision</td>
<td>76</td>
</tr>
<tr>
<td>16</td>
<td>Categories for organizational influences</td>
<td>77</td>
</tr>
<tr>
<td>17</td>
<td>The HFACS framework</td>
<td>79</td>
</tr>
</tbody>
</table>
Abstract

Author: Arjun Vijayanarayanan

Title: Nurturing Systems Thinking: Developing a framework based on the Human Factors Analysis and Classification System (HFACS) to improve systems development processes

Institution: Embry-Riddle Aeronautical University

Year: 2011

Large systems engineering projects have an astonishingly high failure rate. The reasons hypothesized for such a high failure rate include the neglect of nonsalient system elements such as social and human or organizational aspects of systems. Social and human factors have long been known to be critical elements of systems that are frequently ignored (e.g., Goguen, 1994). Systems engineering processes can benefit and be improved by effective utilization of a framework that helps developers expand their attention and efforts beyond the salient aspects of the system and the development process. In the aviation field, the Human Factors Analysis and Classification System (HFACS) is used to facilitate the consideration of non-salient, easily overlooked influences on the potential for aviation mishaps. This system has improved the effectiveness of the aviation accident investigation by helping investigators perform a thorough analysis of the system factors that may have contributed to the accident. HFACS helps aviation organizations improve their quality assessment and monitoring by making explicit the relationships between a wide range of organizational factors and accident risks. In this research, a framework similar to HFACS was developed for the systems engineering domain.
The purpose of the framework is to guide and improve systems engineering projects. This research was conducted using qualitative methods to identify the elements and structure of a framework for quality improvement in system engineering. Data extracted from interviews and systems engineering literature was assessed in a bottom-up manner to identify emergent patterns and in a top-down manner using HFACS-based themes. The framework developed from this research can be used to guide systems development organizations analyze both the obvious and the latent reasons behind a project’s failure. This would help systems development teams to better understand the causal factors underlying a systems development failure and look out for them in the future. Using the framework, organizations and development teams can better understand the positive effects of considering all elements of a system, including the social and human factors that may not be obvious.
Introduction

Late in his career, accomplished computer scientist and mathematician Joseph Goguen sought to bring attention to the importance of addressing the relationship between social and technical factors in systems engineering. Goguen (1994) noted that “large projects have an embarrassingly high failure rate” (p. 165), and suggested the neglect of social factors has something to do with it. The Standish Group’s “CHAOS Report” provides information on software project failures and the factors that lead to failure. The project failure rates reported during 1994-2009 can be seen in Table 1.

Table 1


<table>
<thead>
<tr>
<th>Year</th>
<th>Failed Projects</th>
<th>Challenged Projects</th>
<th>Successful Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>31%</td>
<td>53%</td>
<td>16%</td>
</tr>
<tr>
<td>1996</td>
<td>40%</td>
<td>33%</td>
<td>27%</td>
</tr>
<tr>
<td>1998</td>
<td>28%</td>
<td>46%</td>
<td>26%</td>
</tr>
<tr>
<td>2000</td>
<td>23%</td>
<td>49%</td>
<td>28%</td>
</tr>
<tr>
<td>2004</td>
<td>18%</td>
<td>53%</td>
<td>29%</td>
</tr>
<tr>
<td>2006</td>
<td>19%</td>
<td>46%</td>
<td>35%</td>
</tr>
<tr>
<td>2009</td>
<td>24%</td>
<td>44%</td>
<td>32%</td>
</tr>
</tbody>
</table>

*Note. Adapted from “The rise and fall of the chaos report figures” by J.L. Eveleens and C. Verhoef, 2010. IEEE Software. Vrije Universiteit Amsterdam, 6, 30-36, p. 31.*
In the above table, *successful* projects are projects completed on time and within budget showing all the features that were intended; *challenged* projects are projects that were completed with an overshooting of both budget and time estimates and also showing fewer features than intended; and *failed* projects are projects that were cancelled before completion.

The statistics are especially alarming in light of the costs of large systems and software engineering projects. For example, the US Internal Revenue Service (IRS) spent $4 billion on a system that, as described by an IRS official, does “not work in the real world,” (Marketplace, 1997).

Other examples of systems development failure include:

- Federal Bureau of Investigation’s (FBI’s) Virtual Case File (VCF) project which was supposed to automate the FBI’s paper-based work environment, allow agents and intelligence analysts to share vital investigative information, and replace the obsolete Automated Case Support (ACS) system. The bureau had to scrap the US $580 million project. Factors that may have contributed to the VCF’s failure include: poorly defined and slowly evolving design requirements; overly ambitious schedules; and the lack of a plan to guide hardware purchases, network deployments, and software development for the bureau (Goldstein, 2005).

- A computer-aided dispatching system was developed for the London Ambulance Service (LAS) to replace the manual system. The LAS received calls; dispatched ambulances based on an understanding of the nature of the calls and the availability of resources; and monitored progress of the response to the call. This new system was designed to
include an automatic vehicle locating system (AVLS) and mobile data terminals (MDTs) to support automatic communication with ambulances. Immediately after becoming operational, the call traffic increased heavily and the system could not keep track of the location and status of units and incorrect databases were formed. Instances of the ambulance crew arriving at their location to find the patient dead and the ambulance answering a “stroke call” as much as 11 hours after the call illustrated the ineffective functioning of the system. Eight days after the deployment, the system was seized completely (Finkelstein & Dowell, 1996).

- NIMROD was a large, UK government–funded, early warning defense system project. Huge amounts of money were poured in and many hundreds of people were hired (Bush, 1997). What looked like a magnificent project in the making, was not really so. This project involved the development of both novel hardware and software. There were problems with both areas, and also with the compatibility of the two. After ten years of work and 100 million pounds spent, the funding was cut and a version of the U.S. Airborne Weapons and Control System (AWACS) was adopted. Reasons behind the project’s failure could be the often increasing requirements, ineffective change management, unrealistic schedules and inefficient communication among development staff (Bush, 1997).

- The Theater Battle Management Core System (TBMCS) is an air command and control (C2) system that performed standardized, secure automated air battle planning and execution management for Air Force, multi-service, and allied commanders in the theaters of operation worldwide. According to Collens and Krause (2005), some of the
problems faced which may have contributed to the failure of the initial system development effort (a follow-on effort was considered successful) included:

- The initial high level system architecture design affected the design and development of the system.

- The requirements management and allocation process for producing the first release of TBMCS was inefficient.

- The system and subsystem design was severely hampered by the complexity of interfacing with legacy applications. Application vendors were not funded and did not give away the code.

- Miscalculation of the maturity and complexity of commercial and third party software products.

- Lack of understanding of how the system would be used and employed by the different groups of users.

- Difficulty in the integration of such a complex system.

- Testing on TBMCS was conducted in contexts that lacked real-world complexity and demands.

- Another complex system development effort that did not fare well was the mechanization of mines in England in the 1950s (Trist & Bamforth, 1951). Innovation and automation were given top priorities and social aspects were neglected. The introduction of this new method was followed by unchanged productivity and severely
deteriorated relations among miners and within the mining community. Factors that could have attributed to the failure of the system included the neglect of social and psychological phenomena (Trist & Bamforth, 1951).

In each of the above systems development examples, a variety of factors were at play, interacting in complex ways. Note that in each case, the developers neglected or experienced difficulty with factors relating to users (user requirements, interface with technology, demands, work practices and work culture). To shed more light on the types of factors that can influence the outcome of systems development efforts, the last of the systems development examples will be described in great detail.

**The Longwall Mining Case Study**

An example of the neglect of relationships between key system elements in systems engineering can be seen in the sociotechnical analysis and case study of the mechanization of mining in England (Trist & Bamforth, 1951). In this example, the effects of new technologies on workers’ interdependencies and relationships were not considered prior to the introduction of the technologies.

The traditional mining method in England had, until mechanization in the mid-1900s, been the “hand got” method. According to Trist and Bamforth, “Most competent authorities appear to be agreed that the ‘hand-got’ methods that preceded the longwall provided the face-worker with a social balance that has since been lost” (p. 4). Hand-got work groups had “responsible autonomy”; they worked in self-paced, self-selected interdependent working pairs to which one or two other people might be attached. Even though the miners had simple
equipment, they had multiple tasks to perform which required multiplicity of skills. They took pride in doing their duties and relied on themselves alone for achieving work goals. Workmates developed close relationships and strong loyalties that extended to one another’s families.

The social organization of hand-got teams reflected decades, if not centuries, of adaptation to work demands. The organization was especially well adapted to the stressful and high-risk conditions of the underground mining environment. Teammates looked after one another and could adapt their work pace and strategies flexibly to challenging conditions that emerged (e.g., conditions such as loss of control over the roof, reduced working heights at the typically 3-ft high coal face, and faults affecting the continuity of the coal seam).

With the introduction of coal-cutters and mechanical conveyers, mining operations in England were changed so that the mining work was done by large teams working across a broad face and multiple coal seams. Thus, mechanization replaced the small self-reliant teams with large groups, usually composed of forty to fifty men, performing large scale activities side-by-side by without any interdependency. They worked in “functional isolation” (p. 30) from the other men on the same shift. The coal-getting task was performed in three sequential shifts: the cutting, ripping and filling-off shifts. The ease and productivity of work during the two later shifts depended on the work of the shifts preceding them. As an example, the cutting shift was required to cut the coal to specific dimensions. If the cutters failed to meet those dimensions, the jobs of workers in the next stage became more difficult. The filling-off shift, where the coal was moved on to the conveyer and where the largest number of men was employed, was where troubles encountered across all shifts would accumulate. Although the work of the
earlier shifts was closely related to the work of later shifts, Trist and Bamforth note that the
tasks were done in strict isolation and the workers never even met.

The sequential nature of the shifts did not accommodate common challenges of mining;
rather, it assumed that each job could be performed with equal proficiency by each miner,
regardless of age or health, and on each day, regardless of mine conditions. Neither time nor
adequate pay was allotted for handling challenges. Arrangements were not made to assist
those with lower stamina. This increased stress on workers and increased tension among
workers and between shifts. Tension between shifts developed because later shifts were not
privy to the challenges that resulted in reduced work output or quality by a preceding shift and
which increased work demands for them. Inter-team conflicts were common with the handgot
mining team, but they improved intra-team solidarity. In contrast, both inter- and intra-shift
conflicts began to arise within the longwall mining team and miners found themselves on their
own in a system that reinforced individualism.

Mechanization did not require a miner use his varied skill set; nor did it respect the
‘underground expertise’ of the older miner. Every group did the same specific task every day;
this task and the worker’s productivity became the basis for worker status within the new
system. The payment method, for example, pay per hole, pay per yard, and day wage, also had
an effect on the worker’s job satisfaction. Apart from the specific task a miner was assigned,
special tasks called “bye-work” would be required at times. Bye work was work that had to be
done to make possible the smooth execution of mining tasks across shifts—e.g., fixing a roof.
Miners did not feel they were compensated fairly for bye work, nor for their usual work when
faced with challenging conditions—which was not uncommon. This had a tremendous effect on the worker’s attitude towards work. Moreover workers in unfavorable natural conditions (e.g., low ceilings or poorly prepared faces) tended to be less likely to put forth good effort, out of annoyance over the conditions and because the low-quality work could be blamed on the bad conditions.

Trist and Bamforth assert that the social changes brought about by mechanization were largely to blame for subsequent low productivity, low morale, and a high turnover rate. Even “widespread incidence of psycho-somatic and kindred neurotic disorders” (p. 30) among miners was seen, which highlights the “human” costs of neglecting psychological and sociological factors under mechanization.

Goguen on How to Address the Social Aspects of Systems

Thus far some examples of system development failures have been discussed. This and the following sections will describe more about the goals of systems engineering, in particular the systems thinking concept.

Goguen implicates as a root cause of the very high system development failure rate the neglect of key system elements during engineering projects. In particular, Goguen points to the social elements of the system—the technology users and the users’ organization—as routinely neglected and poorly addressed by systems developers. Additional evidence of this tendency to neglect human and social aspects can be found in the effort to add a human view to the Department of Defense Architectural Framework (DoDAF; Baker, Stewart, Pogue, & Ramotar, 2008; Bruseberg, 2008; Handley & Smillie, 2008). Goguen used ethnomethodology as a tool to
Ethnomethodology is a method for understanding the ways in which culture shapes and imbues the work practices, goals, and priorities of people in an organization by analyzing their accounts of their day-to-day experiences, their work artifacts and their communications. It is a descriptive method and does not engage in the explanation or evaluation of the particular culture undertaken as a topic of study. Goguen (1997) stated, “Ethnomethodology tries to reconcile radical empiricism with the situatedness of social data, by looking closely at how competent members of a group actually organize their interactions” (p. 10).

Goguen used ethnomethodology to explain the principle of accountability and the principle of orderliness. According to the principle of accountability, the members in a group are held accountable for their actions depending upon where their group is placed in the society or, in our context, in the organization. Thus, the behavior and interaction of members of a work group are constrained by the nature of accountability imposed by the group. According to the principle of orderliness, social interaction and behavior are orderly and can be understood with respect to contextual and cultural constraints. Thus, a group’s interaction can only be fully understood in the context of that particular group, which is the essence of ‘qualities of situatedness’. Thus to understand a system, the work group should be analyzed. The failure to take into account this human side of the system leads to system designs that are poorly matched to an organization’s work culture and practices. Trist and Bamforth make this point.

All analysis of the mining system problems were analytical (dry); neglecting the social aspects of the system. Thus, ethnomethodological approaches play a vital role in the development and
achievement of *systems thinking*, an important systems engineering perspective that will be explained in the next section.

**Systems Engineering Quality Improvement**

According to the International Council on Systems Engineering (INCOSE, 2006), a system is “a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results” ("Definition of a System”, para. 1).

Note that people are included in this definition as a core system element. Thus although people and related social factors tend to be neglected by systems developers, the systems engineering profession identifies them as key factors to be addressed. Furthermore, the systems engineering profession emphasizes the importance of systems thinking - the consideration of a system in a holistic way, in terms of all its components and their relationships. Again, the inclusion of people and social issues in the systems engineering problem space would seem to be a goal of the profession and, by extension, of any group endeavoring to develop or adapt a system.

To help developers engineer systems in ways consistent with the goal of systems thinking and the definition of a system, a framework is proposed. This framework will help developers monitor whether they are engineering the system in a balanced way, that is, whether they are addressing all systems elements and their relationships and whether they are using processes, methods, and tools that support a balanced and holistic approach. In doing so
the framework should help developers handle the social aspects of systems as well as they handle the technical aspects. Goguen’s concerns might be reduced and, likewise, the number of unsuccessful system development efforts and unusable or user-hostile systems might be reduced.

Systems thinking helps developers consider the mutual dependence of all system components. It promotes team effort by integrating all the disciplines and forms a well defined development process. Systems thinking also helps systems developers understand both the emergent characteristics of the system and various development and acquisition risks, which thereby helps them achieve the final goal efficiently. Thus systems thinking strives to provide a quality product in which all elements, including users, interact well with and support one another.

Approaches for Achieving Quality in Systems Engineering

According to Goguen (1997), “The very rapid rate of change of requirements, which is so typical of large projects, implies an even more rapid rate of change for specifications. This makes many formal methods very difficult, perhaps even impossible, to apply in practice” (p. 115).

Goguen (1997) stresses the point that systems engineering must be highly flexible, supportive of emergent requirements and open, in the sense that it has to adapt to the changes in the customer needs.

An evolutionary systems engineering approach, an approach that addresses both the social and the technical aspects of systems, is thought by Goguen to be the key to systems that
integrate users with technology, and which are more likely to produce systems development successes. An evolutionary approach, in which a system is gradually developed in its context of use, is more consistent with the realities of requirements and system life cycles than currently popular approaches.

The requirements for a large system are tough to determine; the requirements only become clear when the system is successfully operating in its social and organizational context; requirements evolve as system development proceeds, and a reasonably complete and consistent set of requirements for a large, complex system emerge as it is used. Determining whether some system meets its true requirements is the outcome of a complex social process that typically involves negotiation, and may involve legal action. Thus it is usually entirely misleading to think of requirements as pre-given.

According to Goguen (1997), “lifecycle phases cannot be fully formalized. Indeed, the activities that are necessary for a successful system development project cannot always be expected to fit in a natural way into any system of pre-given categories.” (p. 7) and “the requirements phase of a large system development project is the most error-prone, and these errors are the most expensive to correct” (Boehm, 1986 Davis, 1990 as cited by Goguen).

Goguen’s claim about the neglect of social factors in system development efforts, is echoed in the work of Robert Hoffman, a human factors methodologist who notes that popular software and system engineering approaches (e.g., waterfall, spiral, and agile development frameworks) are noticeably lacking in guidance for integrating technology designs with their users and use environments (e.g., Hoffman & Elm, 2006). Two of the most popular and
influential systems engineering tools, the waterfall development model and the spiral development model are discussed below, followed by a discussion of a popular approach to systems development quality assurance called the Capability Maturity Model Integration (CMMI). Each is presented as an example of a popular and accepted approach to quality assurance in systems engineering that neglects human and social factors of systems.

The waterfall model. In the waterfall model, the process proceeds from one phase to the next in a purely sequential manner. For example, when the requirements specification is fully completed, one proceeds to design. This design should be a plan for implementing the requirements given. When the design is fully completed, an implementation of that design is made by coders. Towards the later stages of this implementation phase, separate software components produced are combined to introduce new functionality and reduced risk through the removal of errors. The waterfall model can be seen in Figure 1.

![Figure 1. The waterfall model. Adapted from “Managing the Development of Large Software Systems” by Royce (1970). Proceedings of IEEE WESCON 26, 1–9, p. 329](image)
The waterfall model is defective in large and complex systems development projects where the requirements constantly keep changing and where stepwise completion of the processes is not a reality because later stages affect the work of earlier stages. While using the waterfall model, if clients change their requirements after the design work ends, the design will not be modified to accommodate the new requirements.

The **Spiral development model**. The spiral development model, as seen in Figure 2, is a software development process introduced by Boehm (1986) combining the benefits of prototyping (an iterative process of building a model of a system) and the waterfall model in an effort to combine advantages of top-down and bottom-up development approaches (A top-down approach is essentially the breaking down of a system to gain insight into its compositional sub-systems. Here an overview of the system is first formulated and then each subsystem refined in greater detail until the entire specification is reduced to base elements. A bottom-up approach to development is done in a data-driven, context-specific manner and not based on generic pre-determined structure).

The spiral model is intended for large, expensive and complicated projects. In every iteration, project risks are effectively identified and handled. A preliminary design is created for the new system. A first prototype of the new system is constructed from the preliminary design. This prototype is evaluated in terms of its strengths, weaknesses, and development risks. Now the requirements of the second prototype are defined. Planning and designing the second prototype now begins followed by constructing and testing the second prototype. Therefore the spiral development model is a model where timelines are taken care of and the
product can be delivered as scheduled. This model is not usually understood well and
development teams end up following the waterfall model, even though they claim that they
used the spiral development model. That is, they develop a system one module at a time via
the waterfall stages and consider the sequence of module development to be spiraling. In
contrast, Boehm envisioned that the entire system would be iteratively refined in a holistic way
(Boehm, 1986).
Figure 2. The spiral development model. Adapted from "A Spiral Model of Software Development and Enhancement" by Boehm, 1986. ACM SIGSOFT Software Engineering Notes, 11(4):14-24, p. 16.
**Capability Maturity Model Integration® (CMMI).** Capability Maturity Model Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes that ultimately improve their performance. CMMI was developed by a group of experts from industry, government, and the Software Engineering Institute (SEI) at Carnegie Mellon University. CMMI can be used to guide process improvement across a project, a division, or an entire organization. It helps to integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes.

In a CMMI appraisal, an organization is appraised and given a maturity or a capability level of 1-5 as seen in Figure 3. There are three classes of appraisals, namely A, B and C, focusing on identifying improvement opportunities and comparing the organization's practices to the CMMI best practices. The appraising norms should conform to the *Appraisal Requirements for CMMI* (ARC) document. The ARC document defines the requirements considered essential to appraisal methods intended for use with CMMI models. Appraisal methods used in the ARC document may be applied for different purposes, including assessments for internal process improvement and capability evaluations for supplier selection and process monitoring.
Additionally, the Standard CMMI Appraisal Method for Process Improvement (SCAMPI) is designed to provide benchmark quality ratings relative to CMMI models. It is applicable to a wide range of appraisal usage modes, including both internal process improvement and external capability determinations. SCAMPI also satisfies all the ARC requirements. The CMMI model framework can be seen in Table 2.
## Table 2

*CMMI Framework*

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
<th>Area</th>
<th>Maturity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQM</td>
<td>Requirements Management</td>
<td>Engineering</td>
<td>2</td>
</tr>
<tr>
<td>PMC</td>
<td>Project Monitoring and Control</td>
<td>Project Management</td>
<td>2</td>
</tr>
<tr>
<td>PP</td>
<td>Project Planning</td>
<td>Project Management</td>
<td>2</td>
</tr>
<tr>
<td>CM</td>
<td>Configuration Management</td>
<td>Support</td>
<td>2</td>
</tr>
<tr>
<td>MA</td>
<td>Measurement and Analysis</td>
<td>Support</td>
<td>2</td>
</tr>
<tr>
<td>PPQA</td>
<td>Process and Product Quality assurance</td>
<td>Support</td>
<td>2</td>
</tr>
<tr>
<td>OPD</td>
<td>Organizational Process Definition</td>
<td>Process Management</td>
<td>3</td>
</tr>
<tr>
<td>CAR</td>
<td>Causal Analysis</td>
<td>Support</td>
<td>5</td>
</tr>
</tbody>
</table>

CMMI focuses on three areas of interest:

- CMMI for Development (CMMI-DEV), addressing product and service development.
- CMMI for Acquisition (CMMI-ACQ), addressing supply chain management, acquisition and outsourcing processes.
• CMMI for Services (CMMI-SVC), addressing delivery services within and out of the organization.

The intended benefits from using CMMI include:

• The organization's activities are explicitly linked to its business objectives.
• Customers can be informed about various practices performed in the organization and their adherence to CMMI processes.
• Increased visibility into the organization's activities which helps to ensure that the product or services provided meets the customer's expectations.
• CMMI appraisal increases performances in the areas of cost, customer satisfaction, quality, productivity, and schedule.

Thus CMMI appraisals are based on whether the business and engineering processes specified by an organization are actually followed by that organization.

The systems engineering practices discussed above (waterfall model, spiral development model and CMMI) are formal models that reinforce a focus on managerial and technical aspects of a system such as technology development, schedule, and budget. These models do not help developers address or consider social and human aspects and they attempt to simplify and constrain complex problems and the simplification process runs counter to addressing social and human aspects. For example, Goguen explains requirements as being social, open and emergent, but the CMMI and waterfall models do not permit emergence of requirements or adaptation to changes, two critical processes that help handling complex
problems (Norman, 2004). Limitations such as these were highlighted in a study by Hoffman, Neville, and Fowlkes (2009) and will be discussed in the next section.

**Limitations of traditional approaches to quality assurance**

Current engineering practices neglect relationships and dynamics among system elements in favor of a focus on individual system components. To study the importance such relationships between system elements, a cognitive task analysis (CTA) was conducted to study the work domain of IT systems development (Hoffman, Neville, & Fowlkes, 2009). Documentation analysis conducted as a part of the CTA involved examining over 50 documents on topics related to development and acquisition of IT. Interviews were conducted in which six experienced systems and software engineers were asked about systems development challenges they had faced. Systems development challenges that emerged included difficulties in identifying user needs, coordination within development team, coping with complexity, budget and schedule, and most importantly, difficulty in accommodating changing information and requirements across the development effort.

The CTA highlighted the insidious effects of changing nature of requirements. The authors stated, “Traditional systems development paradigms require early requirements definition, and all software-writing activities flow from that. Subsequent changes in requirements often present “back to square one” situations, are therefore ignored or taken as evidence of poor requirements gathering.” The development teams must understand the emergent, open, contingent nature of requirements and the associated constraints. The authors suggest, “Requirements assessment should be an ongoing process across the
development effort, so that it detects ways in which practitioners adapt then work methods and new prototypes.”

**Systems Thinking Frameworks in Other Domains**

The idea behind this research is to develop a framework that helps the system and software development teams to focus on both salient and less salient elements of a system including the technology components, humans in the system, system environment setting, the organizational and cultural values and practices, and the interfaces between components. The development of the system using a holistic approach is intended so that better quality systems can be engineered.

The next section presents existing formal models that reinforce systems thinking and which therefore have had an influence in the development of a “systems thinking” framework for systems development. In the aviation domain, analysis of breakdowns in an organization can be particularly well understood by the “Swiss cheese” model proposed by Reason (Reason, 1990), as seen in Figure 4. Reason’s model has served as the foundation for the Human Factors Analysis and Classification System (HFACS) framework, which is used in the aviation domain and also in the medical domain (Wiegmann & Shappell, 2003). After describing HFACS, a recent effort by Paletz, Bearman, Orasanu, and Holbrook (2009) to extend HFACS will be discussed.

The third “systems thinking” framework to be discussed is Leveson’s (2009) Hierarchical Accident Model, which was used to evaluate the causal factors in a mission interruption of the Solar Heliospheric Observatory (SOHO) (Leveson, Lundqvist, Stringfellow, & Weiss, 2009).
Reason's “Swiss Cheese” Model of human error causation. James Reason (1990) describes four levels of factors that can contribute to a system breakdown in the aviation industry (too often referred to as “human error”, a misnomer used by Reason as well) in the “Swiss Cheese” Model.

![Diagram of the Swiss Cheese Model]

Figure 4. The “Swiss Cheese” Model of human error causation. Adapted from “Human Error,” by Reason, 1990. New York: Cambridge University Press.

Reason presents four levels of an organization that contributes to an error, namely; *Organizational influences, Unsafe supervision, Preconditions for unsafe acts, and Unsafe acts.*

According to Reason, factors at all levels of an organization contribute to an error, even though the error was committed at one particular level. Thus, aircrew, maintainers, supervisors and management are involved in the causal factors leading to the accident/incident in aviation. The
first organizational level of Reason’s model, describes *Unsafe Acts* that refer to the aircrew errors that lead to an accident. They are referred to as *active failures* because most causal factors are uncovered in this level. The *latent failures* refer to the actions and conditions that set stage for and make the accident possible. Categories of latent failures include *organizational influences, unsafe supervision,* and *preconditions for unsafe acts.*

*Organizational influences* describes factors such as political factors or fiscal factors that play a major role in shaping the organizational climate. Reason suggests that the organizational policies play a role in the causal sequence leading to a failure. *Unsafe supervision,* describes the inefficient supervision or inappropriate practices of the supervisors which might lead to a failure. *Preconditions for unsafe acts* involve work conditions that foster mental fatigue and inefficient communication and coordination among the crew members and also between all the members of the organization.

Reason’s model guides the accident investigation processes by addressing all the levels in the organization in a causal sequence to failure and making the accident analyst look for causal factors in both the active and the latent levels of the organization.

*Human Factors Analysis and Classification System.* “*Human beings by their very nature make mistakes; therefore, it is unreasonable to expect error-free human performance*” (Wiegmann & Shappell, 1997).

Based on Reason’s (1990) “Swiss Cheese” model of system failures, the Human Factors Analysis and Classification System (HFACS), as seen in Figure 5, was developed. HFACS is useful
in accident investigation because it helps analysts consider the latent failures within a causal
web of influences. The HFACS framework breaks down each level of the “Swiss Cheese” model
into various categories thereby providing a wide range of causal factors to be considered and
addressed for effective and efficient aviation accident investigation.

HFACS categorizes “Organizational Influences” into errors based on Resource
Management, Organizational Climate, and Organizational Processes. Similarly “Unsafe
Supervision” is classified into four categories: Inadequate Supervision, Planned Inappropriate
Operations, Failure to Correct Problem, and Supervisory Violations. “Preconditions for Unsafe
Acts” includes Environmental Factors, Conditions of Operators, and Personnel Factors.
Environmental Factors again include Physical Environment and Technological Environment.
Conditions of Operators includes Adverse Mental States, Adverse Physiological States, and
Physical/Mental Limitations. Personnel Factors includes Crew Resource Management and
Personal Readiness. “Unsafe Acts” is classified into two categories: Errors and Violations. Errors
are again categorized into Skill-based Errors, Decision Errors, and Perceptual Errors. Violations
on the other hand are categorized into Routine Violations and Exceptional Violations
(Wiegmann & Shappell, 2003).

A comprehensive description of all the 19 causal categories described in HFACS can be
seen in Appendix A. The HFACS framework provides the accident analysis investigators with a
tool for comprehensively identifying and classifying the human causes of aviation accidents.
An application of HFACS can be seen in helicopter emergency medical service (HEMS), which plays a vital role in the U.S. health care industry. Since 1998, there has been a troubling increase in the number of accidents and fatalities associated with this group because of human related errors (Boquet et al., 2009). Like other aviation operations, skill-based errors comprised the majority of the unsafe acts, followed by decision errors, violations and perceptual errors. Accidents that involved violations were three times more likely to be associated with a fatality (Wiegmann & Shappell, 2003). Thus HFACS was used to shed light on the human error involved in HEMS accidents, allowing for a greater description than is typically associated with standard reporting.

While the HFACS improves system failure classification is should be clear from the earlier arguments that social factors are neglected. Paletz, Bearman, Orasanu, and Holbrook (2009) introduced additional categories to the HFACS structure. Paletz et al. state, “Social psychological phenomena have long been known to influence attitudes and behavior but not been highlighted in accident investigation models.” The authors attempted to make the HFACS taxonomy more complete by identifying and including social factors that contribute to system breakdown. Social pressures contribute to poor decision making by leading the pilots to underestimate potential dangers (Paletz et al., 2009).

Paletz et al. studied the consequence of social pressures on the pilots flying in Alaska, who often fly missions in adverse weather conditions and work within minimal infrastructure, and often provide others with basic necessities. They used the critical incident interviewing technique (Flanagan, 1954; Klein, Calderwood, & MacGregor, 1989) to interview 28 pilots who
were asked to describe a challenging decision situation involving weather when they were pilot in command. The interviews were audio-recorded and later transcribed. The initial coding was conducted in an iterative data driven, bottom-up fashion and resulted in the identification of 31 pressures which were then coded into five categories (the first four categories were social psychological influences) namely: informational social influence, the foot-in-the-door persuasion technique, normalization of deviance, the internal self-motives of impression management and self-consistency, and other (included pressures which were not strictly social or psychological).

The four social psychological influences are explained in Table 3. These categories (involving social psychological influences) were then grouped into themes based on HFACS, as seen in Figure 6 and Figure 7.

Table 3

*Social Psychological Influences Assessed by Paletz et al.*

<table>
<thead>
<tr>
<th>Social Psychological Influence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Informational Social Influence</strong></td>
<td>Informational social science is a type of conformity, where a person look to others for cues concerning correct behavior when he/she is unsure of the correct way to behave.</td>
</tr>
<tr>
<td><strong>Foot-in-the-door Persuasion Technique</strong></td>
<td>Foot-in-the-door technique (FITD) is a compliance tactic that involves getting a person to agree to a large request by first setting them up by having them agree to a modest request.</td>
</tr>
<tr>
<td><strong>Normalization of Deviance (Progressive commitment)</strong></td>
<td>Normalization of Deviance is a phenomenon based on insidious small, progressive changes. It is an incremental acceptance of a progressively lower threshold, e.g., of safety, by a group of people.</td>
</tr>
</tbody>
</table>
People are motivated to act in ways that will earn them respect and social acceptance. On the other hand if their actions are conflicting with their beliefs, the person often tends to adjust their beliefs.

The Paletz et al. study enunciates the importance of social psychological phenomena in real flight operations and can be extended to pilots flying outside of Alaska. The author’s report that these factors are generally over-looked in accident investigations and that existing accident reports do not serve as a good source to study the role of social and psychological phenomena in accidents. The Paletz et al. study was therefore important in that they attempted to include rare pressures in HFACS to highlight the possible role of these social psychological factors in accidents.

**The Hierarchical Accident Model.** Another accident analysis model designed to help analysts consider a comprehensive range of factors is the Hierarchical Accident Model,
developed by Nancy Leveson (2009). The Hierarchical accident model was used to evaluate the contributing factors in a mission interruption of the Solar Heliospheric Observatory (SOHO) (Leveson, Lundqvist, Stringfellow, & Weiss, 2009), which was a joint effort between the National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA).

The SOHO launch took place on December 2, 1995, and the vehicle performed helioseismology and monitored the solar atmosphere, corona and winds. According to Leveson et al. (2009), a simple chain of events model is inadequate for analyzing accidents in complex systems. Leveson et al. advocated the addition of a hierarchical abstraction that expands the types of contributing factors considered. Three levels of hierarchical abstraction were used in the SOHO accident analysis. Level 1 describes the mechanism of the accident, that is, the chain of events. Level 2 includes the conditions that allowed the events in the first level to occur. Level 3 includes the factors referred to as the root causes or systemic errors. Systemic errors include:

- Flaws in the safety culture, which include overconfidence and complacency, not understanding software risks, inadequate emphasis on risk management, and incorrect prioritization of changes.
- Ineffective organizational changes, which include diffusion of responsibility and authority, absence of a system safety program, limited communication channels, and poor information flow.
- Ineffective technical activities, which include flawed or inadequate review process, inadequate specifications, inadequate software or systems engineering, inadequate system
safety engineering, simulation environments not matching operational environments, and inadequate cognitive engineering and feedback.

The hierarchical accident model was also used to identify systemic factors in Ariane 5 and Titan IVB-32 accidents. Similarities in the systemic factors in the three of the above mentioned accidents, including the SOHO mission interruption, can be used to develop prevention strategies for avoiding future accidents.

**Premise Underlying this Research.** The above discussions shed light on different kinds of frameworks used in accident investigation and illustrate their job of bringing to light a number of contributing factors and conditions leading to an accident. These factors can then be used in an organization to understand their influence better, discourage the negative and encourage the positive factors, and look out for them in the future to avoid accidents. By focusing attention to a comprehensive set of latent and conspicuous factors, including preconditions to an accident or an undesirable situation, the influence of preconceptions, tendency to premature closure (i.e. making decisions early), and other biases may be reduced. The framework helps the organization to improve the quality of their work and product.

Thus, the premise underlying this research is that software and systems development project teams will benefit from a similar type of framework that helps them attend to all key elements of the system being developed and associated engineering activities. The framework may also help an organization develop and nurture the various engineering activities to develop quality teams, efficient teamwork, and hence improve quality in their work. In particular, a comprehensive systems development framework could help organizations attend not just to technical, budget, and schedule factors but also to human and social factors as a part of the
development work and engineering activities. In other words, the present work is a first step towards a tool that should foster systems thinking in systems development.

Approach

In a recent issue of the journal Systems Engineering, authors Valerdi and Davidz (2009) lament the lack of empirical research in the field of systems engineering. They suggest that four challenges have, in particular, discouraged researchers from tackling systems engineering topics: the immaturity of the field of systems engineering, a lack of appreciation for empirical social science research that is qualitative in nature, a lack of access to data from systems engineering teams, which operate in real-world settings over extended periods of time, and a lack of accepted measures of quality in systems engineering.

This empirical research was conducted to develop a framework for systems development. An ethnographic approach was used for data collection and analysis. The approach was ethnographic in that data obtained using naturalistic methods were assessed to learn about systems engineering culture, challenges, and factors that influence engineering activities.
Method

Overview

In this systems engineering research project, transcripts of semi-structured interviews with engineers were analyzed for the purpose of developing a framework that can improve quality in systems engineering the way HFACS framework has been used to improve safety in aviation. The framework developed in this research effort is expected to serve as a foundation that can be revised and built upon in future efforts.

Methods advocated by Valerdi and Davidz (2009) for studying systems engineering include semi-structured and unstructured interviews, qualitative research methods that are popular because of the rich and relatively unbiased (i.e., less influenced by the researcher) data they typically produce. Data used in the present study were collected using a semi-structured interview method referred to as Critical Decision Method (CDM, Flanagan, 1954; Klein, Calderwood, & MacGregor, 1989). The CDM is a procedure for gathering practitioner knowledge used to detect, respond to, and navigate difficult events. This technique was used by Paletz et al. in their study, which was discussed earlier.

In a CDM, the interviewee recalls and describes a specific memorable past event in detail. By relying on the recall of specific memorable past events, the researcher can collect data grounded in actual experience rather than introspective, generalized reports. Data collected using the CDM were broken down into data elements/chunks, which were coded. Patterns in the data were noted with respect to the situation and a hypothesis, or grounded theory, was formulated. This is consistent with Grounded Theory Research. Grounded Theory
Research is a qualitative research approach where data are collected and evaluated before hypotheses are derived (Strauss & Corbin, 1990). Each hypothesis, or theory, derived from the data was validated using investigator triangulation (Johnson, 1997), a strategy used to promote qualitative research validity where multiple researchers interpret the collected data. Another strategy used to bolster validity is maintaining a link between findings and data in its raw form to the extent possible. Maintaining this link allows readers to judge researcher interpretations of the data.

Semi-structured interview data were analyzed for this project. The semi-structured interview data analyzed for this project were collected by Hoffman, Neville, and Fowlkes (2009). Hoffman et al. analyzed these data to identify challenges faced by systems development teams; in the present work, the data were re-analyzed to identify systems development goals, activities, challenges, characteristics, and factors that affect quality of the systems thinking in systems development organizations.

**Participants**

Hoffman et al. conducted semi-structured interviews with six systems development practitioners. Four were trained as software engineers and two were trained as systems engineers. Five had 20 years or more of experience and one had 15 years of experience.

**Knowledge elicitation approach.** Each engineer was asked to recall a challenging and memorable engineering project and talk through it. Three of the engineers were asked to describe day-to-day activities. The three other engineers answered a set of open-ended questions, as seen in Appendix B, about team structure, goals, activities, and processes. The
projects chosen by the engineers are described in Table 4. Interviews were approximately 2 hrs in duration. Interview data were audio recorded; audio recordings were subsequently transcribed.

Table 4

*Characteristics of the System Development Projects Described by the Participants*

<table>
<thead>
<tr>
<th>Employer</th>
<th>Project team</th>
<th>Role in team</th>
<th>Project Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major Corporation</td>
<td>Medium-large; three to four teams of three to five people; multiple organizations.</td>
<td>Systems engineer as overall technical lead</td>
<td>Over 18 months, develop a demonstration of an upgraded legacy training system</td>
</tr>
<tr>
<td>Major Corporation</td>
<td>Small-medium; two team leads; two to five people per team</td>
<td>Software engineer as team lead</td>
<td>Over 2 months within a 2-year project, develop a control system that ties the &quot;blocks&quot; together and interfaces with 20 legacy messaging protocols</td>
</tr>
<tr>
<td>Major Corporation</td>
<td>Large; three to four teams of seven to ten people; multiple organizations</td>
<td>System engineer as team lead</td>
<td>During 12 years of a two decade acquisition program, work on proposals and a simulation tested for early testing of integration among system components</td>
</tr>
<tr>
<td>Small Business</td>
<td>Small-medium; one team of seven to nine people</td>
<td>Lead software engineer</td>
<td>Over 3 years, develop a decision support tool for helicopter crews.</td>
</tr>
<tr>
<td>University</td>
<td>Medium-large; two to five people in each of four to five teams</td>
<td>Software engineer as team lead</td>
<td>As part of ongoing maintenance and evolution of a web based library service, port the system from Unix to Linux, over a six-month period</td>
</tr>
</tbody>
</table>
Small Business | Medium; two to four people in each of three to four teams | Software engineer as overall technical lead | Continuously evolve a multipurpose command-and-control system


Data Analysis

To code the data, the element of analysis was defined. These elements were individual sentences or chunks of narrative that express a single idea or concept; examples can be seen in Table 5. The data elements were coded twice: using data-driven codes and using codes from the HFACS framework.

**Coders.** Multiple coders (three in number; Coder A, Coder B, and Coder C) were trained to code the data. Coder A performed both the data-driven coding and coding using the HFACS categories, Coder B performed the data-driven coding and Coder C performed the HFACS coding. Coder A was “HFACS certified.” He completed a HFACS super-user training seminar. To train for the data-driven coding, coders A and B performed four iterations of coding. They coded a subset of the data each time and then convened to compare codes and to develop a shared understanding of the codes. To train for the HFACS based coding, Coder C reviewed HFACS articles and performed two iterations of coding followed by feedback from Coder A.

**Data-driven coding.** Data analysis methods focused on identifying patterns and relationships in the data. The data were first reviewed broadly to identify themes and possible patterns and relationships. An initial set of data-driven codes were developed based on the initial review and were used to code the data. More codes evolved during the second and third
iterations of reviewing the data, and by going through relevant literature. Using the data-driven coding approach, data elements were iteratively coded in a bottom-up, or data-driven, manner by Coders A and B.

Table 5

*Interview Transcripts as Data Elements*

<table>
<thead>
<tr>
<th>Data Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>At &lt;my company&gt;, they have extensive processes. That has to do with the CMMI thing.</td>
</tr>
<tr>
<td>And we’re a level 4 wanting to become a level 5</td>
</tr>
<tr>
<td>Especially at level 5, that’s the highest. When you’re at that level...there’re benefits to it, but there are some frustrations with it, too at times.</td>
</tr>
</tbody>
</table>

When data could not be coded using the existing codes, new codes were proposed and introduced into the coding process. The codes developed by Coder B were handed over to Coder A to compare against his set of codes. Coder A reconciled differences and produced the final set of codes.

**HFACS coding.** The top-down approach involved coding the data elements with HFACS categories. A subset of high-level HFACS categories was identified as potentially relevant to systems thinking and systems development. These categories were used to code the data and are listed in Table 6. HFACS categories such as “Preconditions for Unsafe Acts”, “Unsafe Acts”, and “Unsafe Supervision” are not clearly relevant to systems thinking in the systems development domain and so were not used as codes. Coder C handed over their codes to Coder
A to check for corroboration between the codes. Coder A reconciled the differences and decided on the finalized set of codes to be included in the framework.

Table 6

*Categories of HFACS Used to Code Data*

<table>
<thead>
<tr>
<th>HFACS Categories</th>
<th>Activities involved within the category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Management</td>
<td>Resource management deals with managing budget resources and excessive cost cutting.</td>
</tr>
<tr>
<td>Organizational Climate</td>
<td>Organizational climate includes: organizational structure dealing with chain of command, delegation of authority, and the communication channels present within the chain of command; and organizational culture which includes organizational customs, values, beliefs, attitudes, norms, and rules.</td>
</tr>
<tr>
<td>Organizational Process</td>
<td>Organizational processes includes managing schedules, procedures adopted to meet standards, and risk management</td>
</tr>
</tbody>
</table>


Three examples of coded data elements coded using HFACS categories are shown in Table 7. Appendix C presents a set of example data elements and assigned codes.
Table 7

Examples of Data Elements Coded Using HFACS Categories

<table>
<thead>
<tr>
<th>Coder A codes</th>
<th>Coder C codes</th>
<th>Final codes</th>
<th>Data Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organizational Process</td>
<td>Organizational Process</td>
<td>Organizational Process</td>
<td>At &lt;my company&gt;, they have extensive processes. That has to do with the CMMI thing.</td>
</tr>
<tr>
<td>Organizational Climate</td>
<td>Organizational Climate</td>
<td>Organizational Climate</td>
<td>Team was a relatively large group and was a changing group</td>
</tr>
<tr>
<td>Organizational Climate</td>
<td>Organizational Process</td>
<td>Organizational Climate</td>
<td>From the company's point of view it's tougher because there are so many people on a project it is tougher to tie you to the performance of the project and it's much more subjective.</td>
</tr>
</tbody>
</table>

Reliability assessment. To assess the reliability of coding, the percent of correspondence between codes assigned by coders in each pair was determined. The process adopted to calculate percent of correspondence between the coding pairs can be seen in Appendix D. This process was again used to calculate the percent of correspondence between coders in the HFACS based coding process.
Results and Discussion

Correspondence Between Coders

The percent of shared codes for each of the six interviews can be seen in Table 8. Coders A and B had 74.71% of data-driven codes in common, and Coders A and C had 81.81% of HFACS codes in common.

Table 8
Correspondence Percentages

<table>
<thead>
<tr>
<th>Interview Transcript</th>
<th>Coders A-B (Data-Driven)</th>
<th>Coders A-C (HFACS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>58.50%</td>
<td>73.07%</td>
</tr>
<tr>
<td>B</td>
<td>64.66%</td>
<td>84.15%</td>
</tr>
<tr>
<td>C</td>
<td>73.94%</td>
<td>85.70%</td>
</tr>
<tr>
<td>D</td>
<td>94.87%</td>
<td>84.60%</td>
</tr>
<tr>
<td>E</td>
<td>95.00%</td>
<td>96.67%</td>
</tr>
<tr>
<td>F</td>
<td>61.37%</td>
<td>66.67%</td>
</tr>
<tr>
<td>Average</td>
<td>74.71%</td>
<td>81.81%</td>
</tr>
</tbody>
</table>
Data-Driven Coding

The initial set of codes obtained following a data-driven approach is shown in Table 9.

Table 9

*Initial Set of Codes*

<table>
<thead>
<tr>
<th>Technical Issues</th>
<th>Social Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate Multiple Companies</td>
<td>Team Change Impacts</td>
</tr>
<tr>
<td>Requirements Emerge</td>
<td>Superior Personnel Pressures</td>
</tr>
<tr>
<td>Coordinate Multiple Teams</td>
<td>Organizational Structure Effects</td>
</tr>
<tr>
<td>Resist Changes</td>
<td>Team Structure/Size Issues</td>
</tr>
<tr>
<td>Management Culture Issues</td>
<td>Configuration Management</td>
</tr>
<tr>
<td>Political Factors</td>
<td>Timeline Pressures</td>
</tr>
<tr>
<td>Systems Development Process Effects</td>
<td>Defective Future Plan</td>
</tr>
<tr>
<td>Budget Constraints</td>
<td>System Testing/Evaluation Issues</td>
</tr>
<tr>
<td>Company Law Constraints</td>
<td>Superior Company Dominance</td>
</tr>
<tr>
<td>Customer Issues</td>
<td>Resource Issues</td>
</tr>
<tr>
<td>Present Market Demand Effects</td>
<td>Requirements to Code Transformation Issues</td>
</tr>
<tr>
<td>Management Change Effects</td>
<td>Pressures from Smaller Companies</td>
</tr>
<tr>
<td>System Integration Issues</td>
<td>Project Management Issues</td>
</tr>
<tr>
<td>Procedural Issues</td>
<td>System Complexity Issues</td>
</tr>
<tr>
<td>Quality Assessment Method issues</td>
<td></td>
</tr>
</tbody>
</table>

These initial codes were grouped into higher level categories which are:

- Systems development team
- Systems development project management
- Systems development technical activities
• External pressures

Initial codes that did not map well to the data were dropped and replaced by codes that captured the content of the interviews. Dropped initial codes consist of:

• Resistance to change
• Management culture issues
• Company law constraints
• Management change effects
• Procedural issues
• Team-change impacts
• Resource issues
• Requirements to code transformation issues

The entire set of final codes is included as factors in the framework, as long as they mapped to data in at least one interview. Codes that mapped to data elements in only one or two of the interviews were flagged. These flagged codes were kept along with the codes receiving stronger support because weak evidence for them in this particular study does not disprove their relevance to systems engineering. Future research can shed more light on their potential roles and, in the meantime, their inclusion in the framework may enrich the framework’s support for systems thinking. The codes which had weak support were:

• Acquire Domain Knowledge (Used to code four data elements in one interview)
• Team Member Characteristics (Used to code two data elements in one interview)
• Contracting (Used to code nineteen data elements in two of the interviews)
The data contained a great deal of information regarding teamwork. Teamwork taxonomies by Marks, Mathieu, and Zaccaro (2001) and Smith-Jentsch, Zeisig, Acton, and McPherson (1998), seen in Appendix E and F respectively, were identified and used to perform a “finer-grained” analysis on teamwork. From this teamwork analysis, the codes that mapped well to the data and found relevant to systems development are as follows:

- Mission Analysis and Planning
- Goal Specification
- Team Monitoring and Back-up Behavior
- Coordination
- Conflict Management
- Initiative
- Information Exchange

The final set of codes including their descriptions, can be seen in Appendix G. These codes or factors, and their organization in this initial proposed framework will be discussed in the section below.

The Systems Thinking Framework (STF)

The proposed Systems Thinking Framework (STF) consists of four tiers namely, organizational influences, project management, team processes and technical activities. Each tier has a set of factors associated with it. The external pressures factor envelops all the tiers. Factors that were minimally supported by data (factors used only in one or two of the six interviews) are outlined using dashed lines.
The hierarchical representation of the STF shown in Figure 8 specifies four major sources of influence on the technical activities, which is in the lowest tier. These influences are changes and associated pressures in the external environment, organizational characteristics, project management activities, and teamwork strategies. These sources of influences are organized in the framework according to the extent to which they interface directly with and directly impact the technical activities. The structure represents the way in which different layers of an organization can influence the way work is conducted, a concept borrowed from the HFACS framework.
Figure 8. The STF structure. *Note.* Dashed lines around nodes indicate that a factor was found only in one or two of the interviews. All the other factors were found in at least three of the interviews.
Each of the STF tiers will be discussed below. The discussion will focus on the high-level factors in each tier. The lower level factors are defined in Appendix B.

**Organizational influences.** The top tier, *Organizational Influences*, as seen in Figure 9, consists of factors from HFACS that were found to be relevant to systems development. The factors within this tier can be seen in Figure 9.

*Figure 9. Factors in the Organizational influences tier. Note. The factor indicated by ** indicates a factor with weak support.*

*Development philosophy* deals with philosophies, strategies advocated by the organization for development and acquisition processes, and their influences on the system development process. For example, one engineer, when discussing about the systems
development processes and lifecycles, stated “I don’t like very long development cycles. I prefer shorter development cycles that involves developing in groups and components and testing those.” Another engineer, when asked about the systems development process used in the project he was describing, stated, “We used object-oriented programming.”

The development philosophy adopted can affect the organizational processes and procedures involved in a systems development effort. The system development process adopted in a particular effort, should take into consideration the allocated budget and the time for project submission, which are usually specified in the contractual requirements.

Organizational climate refers to the working environment and factors within the organization that influence worker performance, such as the organization’s structure, delegation of authority and responsibility, communication channels, openness to feedback, and rigidity of processes. An organization’s policies and culture echo its climate. An engineer, for example, stated, “The other lead and I had been introduced to everyone. We had a lot of autonomy. The buck stopped with us. The managers were clueless. We, the two leads, had decent trust and a decent rapport.” A healthy organizational climate should nurture mutual respect and a healthy interaction between management and the technical personnel. Attention to organizational climate factors could lead to changes in organizational policies that would promote better teamwork and improve the quality of the product developed by the team.

Organizational process refers to the business decisions and rules that guide and determine the day to day activities and processes that take place in an organization. As an example, CMMI standards require an organization to follow a set or procedures. An engineer,
describing about CMMI, reported, “We have a project plan, test plans, schedules, etc., that are required by CMMI.” The engineer added, “The formalities are difficult.” Procedures that influence the day to day activities might prove burdensome to the development teams. Frequent checks and calibrations on the organizational processes would help the teams to perform better.

**Project management.** The project managers should have a good awareness about the complete systems development project. For example, one engineer reported “I’m the lead person on this one project, and I probably have the best picture of this whole system.” Project management influences the efficient working of the development team and hence the production of quality products. A team lead said “I hated the Program Managers that didn’t understand anything about what they were trying to build.” The development team’s understanding of the processes involved in the systems development project might be affected if the project manager’s knowledge of the domain is inefficient. The activities performed by a project manager needs to be continuously checked. Thus, as an influence on the activities performed by the development team, *project management* will be discussed below.

The second tier, project management, as seen in Figure 10, is composed of *quality management, customer and user expectations management, complexity management, development team selection* and *contracting.*
Figure 10. System development project management

Quality management activities in this framework involve risk management and configuration management. Risk management deals with risk identification, assessment, and prioritization; strategies adopted to manage risks; and the employment of resources in response to unexpected events. Configuration management deals with documenting requirements, changes in design, and operational information throughout a system’s lifecycle. It is well known that improper documentation and management of the emerging requirements increases the risk of a flawed design and development process and consequently contributes to the development of a poor quality product and low customer satisfaction.

Talking about quality management, an engineer reported, “We’d do a mockup and the business side would give feedback... we’d do a couple of iterations of this depending on the complexity.” Poor quality management can increase the workload of the development team, and lead to situations where the budget and schedule have not been met, thereby leading to failed systems development efforts. The importance of efficient planning of budget and
schedule was also seen in the earlier discussion of “CHAOS REPORT,” where both budget and schedule were the criteria behind a project’s success or a failure.

Managing changes in customer expectations and demands can be vital for a successful systems development project. The activities performed by the systems development team depend on the requirements of the customer. A project manager should effectively communicate with the customers and address their expectations. For example, an engineer stated, “People playing this role need to speak user and systems domains. It should be someone who understands how software is designed, coded and built.”

Development team selection is another aspect for project management. An engineer, describing about the development team in his organization, stated, “Within the team, people were specialized but able to take over if someone was out.” The work done by the development teams need to be regularly assessed by the project manager.

*Complexity management*, dealing with complexity of the system development work, complexity of the system and its infrastructure, and strategies adopted to cope with complexity, is another factor that might affect the work performed by the development team. The complexity of a systems development project can be increased by changing customer expectations, bigger development teams, poor configuration management, complex system designs, time pressures, and workload pressures etc. Involving customers and getting their feedback during the design and development process would help the development teams cope with changing requirements. For example an engineer, describing about increasing complexity
because of changing customer requirements, stated, “We talk to the customer and do a post-mortem with them... As programs get bigger, change is more painful.”

_Contracting_ deals with policies and decisions regarding contract agreements. For example, en engineer reported, “So we were on the F-22 team which ended up winning out of the prototyping phase, they actually built two planes. Prototyped them, and then ours got selected for doing electronic warfare.” The contract agreements will usually decide the budget and schedule allocated for a particular project. The systems development team should be aware of these contract agreements to plan the design and development of the product. Periodic checks could be used to ensure if the project manager is aware of and attending to the contract details.

**Team processes.** The third tier of the framework _Team processes_, as seen in Figure 11, is composed of communication and coordination, conflict management, mission analysis and planning, team monitoring and back-up behavior, and building common ground and awareness. Team processes deal with various strategies the team members use to coordinate to accomplish a task effectively. The interview data contained a great deal of information regarding teamwork. Teamwork taxonomies by Marks, Mathieu, and Zaccaro (2001) and Smith-Jentsch, Zeisig, Acton, and McPherson (1998), seen in Appendix C and D respectively, were used to assess the teamwork related data. The data tended to relate to conflicts, tensions, team monitoring, information exchange, and shared understanding among team members, focused on technical aspects of a system.
In general, the importance of team processes tends to be neglected by the systems engineering community.

Figure 11. Team processes

*Communication and coordination* refer to communicating information in order to coordinate task sequences and timing. An example of interview data related to this factor is, “has to be communication at some point between software and cognitive systems engineers so that neither diverges too far from the other or that the application is headed towards the unusable.” An engineer, in response to solving a problem due to poor communication amongst team members, said “Usually, if there was a communication breakdown, the person’s understanding of the problem would be sent out by email, detailing the problem to everyone involved, followed by a meeting.”

*Conflict management* involves strategies adapted to deal with conflicts; *preemptive*, which aims to prevent, control, or guide situation to preempt possible conflict, and *reactive*, which involves working through task-related and interpersonal agreements among team
members. The interviews suggested that internal conflicts are prevalent within the development team. For example, an engineer stated, “We always had conflict with the operations people.” Another engineer, when asked about possible conflicts between development teams, said “Between functional and implementation teams. I’ve had quite a few. This is a common one.”

*Mission analysis and planning* deals with the interpretation and evaluation of the team’s purpose, identification of main tasks, assessment of available resources, detection and response to changes in operative environment. It also involves identification and prioritization of goals for mission accomplishment. For example, an engineer describing about systems development planning processes reported, “There was a simulation working group. So it had a representative from the X system, from the radar system, from Company Y. And we met once a quarter sometimes more often. And a lot of them were out of where Company Y was. So we had quite a few meetings as a group to say ok, we had to agree on this fidelity of this simulator and how we were going to interface it and how they were going to structure this. So there were a lot of meetings to discuss that and try to work those things out. So then we would all work together when it was final.”

Mission analysis also involves advocating a specific source of action, identifying team priorities, and assigning tasks. For example, an engineer stated, “So the first thing they wanted to do was before any of the other software was built, because they also did the build approach, they said we want you to build a simulator, that’s going to simulate all of your interfaces.”
Team monitoring and Back-up behavior involves cross-checking or monitoring teammates for the purpose of assisting proactively, and providing/requesting assistance. Talking about the importance of regularly assessing the work of development teams, an engineer reported, “I think code reviews that are very beneficial, they are also called peer reviews, is a more generic name. You can peer review a document, or code, or hardware design. So there are many peer reviews all the way through the steps. I’ve seen that work very well.”

Building common ground and awareness involves updating and educating the team members about the day to day processes in the organization, changes in requirements, changes in development methods, and creating awareness about contractual requirements like budget and schedule. For example an engineer reported that they “taught one another new technologies for knowledge transfer.” Efficient communication and coordination can bring about common ground with respect to system development activities and plans. One engineer stated “At the end of every week, the leads made a report of goals, deadlines, and activities from the past week. They gave this to supervisors and distributed among team members. We used verbal communication to clarify, and this kept us pretty well informed. We were responsible for day to day operations and the supervisor needed to know. This is how we communicated concerns and needs to others and tied together teams.”

Technical activities. The last tier of the framework Technical activities, as seen in Figure 12, includes requirement allocation and management, designing and developing the system architecture, testing and evaluation process, integration of the components, and deployment.
Figure 12. System development technical activities

Requirements management involves methods adopted to deal with the identification of requirements, communication and allocation of requirements, and management of changing requirements. Also involved are the problems faced in these methods. An engineer while talking about requirements stated, “It’s basically your high level system requirements. So we actually were following that traditional you know, common process. You start by getting your high level requirements identified and then what we were attempting to do is take those requirements and those exercises and say, we’ve got a phased implementation approach that goes out over 6 phases. In phase 1, you’re going to get these requirements, phase 2 you’ll get these requirements, phase 3 you’ll get these requirements and show how each one of those maps to our plan. Novel concept you know, tell them what they’re going to get, when they’re going to get it. Set all their expectations. And from there we started to drive our system
architecture that we were going to put in place for phase 1.” Lack of user input, incomplete requirements, and ineffective management of changing requirements could result in development of a product with lesser functionality and therefore lack of customer satisfaction. This necessitates the practice of monitoring the requirements regularly.

*Design and development* of the system includes the development process used; design criteria, strategies and methods adopted; and issues dealt with during the process. McConnell (1996, p. 63) states, “Design serves as the foundation for construction, project scheduling, project tracking, and project control, and as such effective design is essential to achieving maximum development speed.” An example given by an engineer, “And, you know, we were trying to define the architecture at the same time we were capturing the requirements, cause we had the first spiral drop we were trying to meet. So we were trying to get that core infrastructure and architecture in place such that we could meet that first delivery.” Here the engineer demonstrates the design and development process adopted. The requirements were collected through the design and development phase, and the team was trying to meet the schedule requirements.

*Testing and evaluation* refers to testing methods employed and issues related with testing and evaluation. Referring to the testing processes involved in his system development process, an engineer stated, “They wanted to do that testing. And so that’s one of the areas where I got quite involved. We ended up having to develop these very complicated procedures and stuff to try to do something similar to that but never did match the real plans we had. So that is where we kind of went a different path and had to come up with these work-arounds to
do the actual testing.” Testing is necessary to be done regularly to evaluate the developments in the design process and check if the design requirements and the contractual requirements are being met. Based on the results of testing and evaluation, decision makers can assess a system’s readiness to advance to the next phase of development, risk reduction, and requirements refinement.

Integration and interfacing includes issues related to integration of system components like hardware, software and related processes. For example, one of the engineers, describing the importance of system integration, reported, “Well that was the goal of the whole system. What I worked on was an early delivery of that where we just did the interfaces to try to make sure that all worked out. And then builds would actually be with the pilot. But yes it was early integration.” Another engineer stated, “No tearing down the house to rebuild with new materials—our materials were so different. We had to understand how parts do their thing to support interfaces between parts.” In the above quote, the engineer reports the importance of early analysis of interfaces between various system elements and how they have to be integrated. Integration of system elements need to be monitored regularly to avoid the risk of incompatibility of system elements which might lead to an overshoot of budget and schedule. Proper documentation of the requirements, design reviews, coordination and timely integration of the system elements may be useful to reduce the risk of incompatibility of all the system elements at the time of review.

Deployment deals with issues that arise when the product is being delivered and deployed at the operational environment. As an example, an engineer reported, “The first
phase was developing these prototypes. The second one was developing the full plane. And I just saw on the news the other day they have delivered the second full squadron of those planes. We were developing the real thing the air force would use, the true final product. That’s why it is called the actual manufacturing stage. Where out of that they start producing the planes.” Through this quote the engineer suggests that the delivery of the final product to the customers is absolutely vital as it decides whether the product is satisfactory to the customers and that if the company can manufacture more products.

Deployment also includes maintenance, integration with the operational environment, and ease of usability by users. Blanchard and Fabrycky (2011) state, “Traditionally, systems are designed and developed, tested to ensure compliance with the initially specified contractual requirements, delivered to the customer for operational use, and then often forgotten” (p. 166). They suggest that the complete maintenance cycle, supply support provisions, and warehousing requirements are also important factors that need to be considered.

*Acquiring domain knowledge* involves acquiring knowledge about the project and the product and a thorough understanding of the processes. Referring to domain knowledge an engineer stated, “We’d just jointed the railroad legacy. There was catch up in domain knowledge (i.e., the preceding/legacy system). The domain knowledge is what’s difficult. We trusted that the manager’s view of the domain is accurate. He told us about the domain by outlining system requirements: ‘here’s what the old system did and here’s what the new must do.’ We were sitting there with a whiteboard, passing around books...domain books were heaped on us—e.g., about vendor’s automatic braking systems and messaging protocols.”
engineer describes the process of educating the team members about the product and customer requirements. He reports that the development team believes that the project manager has adequate knowledge about the system and contractual requirements.

Regular monitoring and assessment of the work done by the development team, regular meetings to promote effective communication among the team and with the customers, thorough understanding of the project, processes involved, the customer requirements, and knowledge about integration of the end product might help the systems development team to perform efficient tasks.

**External pressures.** *External pressures*, as seen in the framework, envelopes all the tiers and influence the system development team. For example, in an organization trying to meet CMMI standards, the project managers and development team are pressured by the high level management to do quality work efficiently enough to meet their high standards. For example, one engineer reported “We have a project plan, test plans, schedules, etc. that are required by CMMI.” He again stated, “In a fire drill, the formalities are difficult; we have adapted lightweight versions of the processes. Done incorrectly, CMMI can kill your program.” Another engineer said “Especially at level 5, that’s the highest. When you’re at that level...there’re benefits to it, but there are some frustrations with it, too at times.” Through this quote, the engineer describes about the increased pressures to meet CMMI standards. Regular checks and calibrations have to be performed to monitor the work done by the development team and check whether the processes associated with meeting an industrial standard is met.
The next section will discuss about some of the problems dealt by the engineers, from all the six interviews taken together, in their respective systems development project, and which may affect the activities performed by the development team and could possibly lead to a system development failure.

**Factors to look out for.** Two engineers describing about problems related to managing requirements, reported about the problems arising due to ambiguous language used in documentation. One of them states “Use cases are problematic because they are ambiguous with respect to language”, and the other stated “Well some of the big things are the English language, if you write out a requirement and you can word it out and it sounds perfect to you, it gets the point across, someone else can read that and just get a totally different spin on it.” A solution to this problem could be face to face communication with the customers. For example one engineer reported “We would find out what the requirements were through just face to face discussion.”

One engineer, talking about change management, reported “For almost all systems, the number one risk is change. No software systems start with a set of requirements and that’s what’s done. True requirements are said to be impossible to attain.” Another engineer said “As programs get bigger, change is more painful.” Suggestions to deal with change include - effective change management, understanding the emergent nature of the requirements, effective configuration management, and effective interaction with the customers. An engineer reported “Requirements creep during the development cycle. A lot of it is necessary, but it should be limited. You have to have someone act as a liaison between people setting
requirements and the developers. My supervisor was the liaison and people high up the company set requirements—pretty general requirements for the most part.”

In an interview, talking about the “Design and Development,” factor in the “Technical Activities” tier, an engineer stated, “The main problem is that people try to develop quickly without thinking through the ramifications. Then there is the other extreme—spending too much time on the design. The middle ground is to narrow down to only those features and requirements that are necessary, but also putting quality into the software. Use use-cases to cover the various things that might happen,” is a serious problem encountered in systems development. He again stated “You can’t build in enough to address everything that might happen. Try to limit the scope of the project by accounting for the most likely problems with the first build. If things come up later you do have to take care of them.”

The section below will discuss about the framework’s attention to the human aspects of the system.

**Encouraging Attention to Human Aspects of Systems.** A main goal behind developing the systems thinking framework was to help developers attend to the large number of factors that can influence the outcome of their work and, especially, the factors that can influence how well a development team attends to social aspects of systems and to the integration of social and technical aspects. To this end, framework factors that can contribute directly to the integration of social and technical aspects have been identified post hoc. These factors consist of ‘Acquire Domain Knowledge’ on the ‘Technical Activities’ tier, all teamwork strategies
identified under ‘Team Processes’, and the factor ‘Complexity Management’ under the ‘Project Management’ tier. Justification for selecting these particular factors is as follows:

**Acquire Domain Knowledge** – Acquiring domain knowledge is a type of technical work that involves learning about the environment and activities in which a system will be used. Acquiring domain knowledge is essential to both understanding the human aspects of a system and integrating them with the technical aspects. This component of systems development is where Goguen’s recommendations for using ethnomethodology can be brought to bear.

**Teamwork strategies** – In the interviews, teamwork strategies were typically cited as supporting teamwork among development team members as they worked on a system’s technical components; the same strategies could facilitate teamwork between members working on technical and human components of a system. Because people working on technical and human system aspects often are trained in different disciplines guided by differing systems development philosophies, the teamwork strategies may be especially valuable—even vital—to the success of their working relationships.

**Complexity Management** - Complexity management encompasses strategies that are used to either reduce or cope with the complexity of a system being developed or of the systems development process. When the human aspects of a system are considered, their consideration tends to introduce a great deal of complexity into the development process and those aspects furthermore represent complexity that is often ignored within the system being developed. Consequently, complexity management activities may be critical to the success of a team that addresses a system’s human elements.
In fact, the interview data did not address human aspects of systems or human-technology integration, with very few exceptions. Similarly, the interviewees tended not to relate the three factors listed above to human-technology integration. For example, a domain analysis of the environment and activities in which a system would be used was described in only one interview. One other interviewee used the term ‘domain analysis’ but it was in the context of gaining an understanding of the engineering domain and engineering precedents. The teamwork factors, when described by the interviewees, tended to relate to conflicts, tensions, team monitoring, information exchange, and shared understanding among team members focused on technical aspects of a system. Complexity management typically involved technology decomposition and technology requirements management.

Support at the organizational and program management levels of the systems thinking framework would likely be required for the three factors above to be used in support of human-focused development activities and human-technology integration. Future research could evaluate this hypothesis and future versions of the systems thinking framework may further develop the roles and influence of these levels. More human-related activities may also be added to the framework as additional research is conducted and the relationships between specific activities and the probability of successful outcome are better understood. Generally speaking, the integration of human and technology elements tends to occur well when development work includes opportunities for human and technology elements to shape each other (e.g., Benbya & McKelvey, 2006; Norman, 2004).
Conclusion

With respect to the success of any systems development project, an engineer reported “Quality, schedule, and functionality are success factors. Longevity; is it used in the long term – this is another factor of success.”

This research made possible the development of a framework for analyzing straightforward and underlying, or latent, reasons behind the neglect of critical systems elements. Use of the framework should help development teams understand factors that can lead to a development failure and look out for them in the future. The STF was developed to improve systems thinking in systems engineering projects. The framework is expected to help developers attend to a broader spectrum of system elements during development and should reduce the risk of system development failure.

Use of the Systems Thinking Framework should help development teams to better understand causal factors that can lead to development failures and look out for them in the future. Systems engineering processes and projects should therefore become more effective and more likely to result in a system in which all elements integrate well. Based on this present research, we suggest a certain set of factors that systems development teams should look out for; other factors may be added based on future work. Future work should additionally investigate risks and strategies associated with the framework factors so that more specific and useful guidance can be built into the framework.
References


Appendix A: Description of the various categories in HFACS

Unsafe Acts

The unsafe acts of operators can be classified into two categories: errors and violations, as seen in Figure 8. Errors include skill based errors, decision based errors and perceptual errors that describe the mental or physical activities of aircrew or the pilots. Violations, on the other hand, refer to the willful disregard for the rules and regulations that administrates the safety of flight. They include routine and exceptional violations.

![Figure 13. Categories of unsafe acts of operators. Adapted from “A human error approach to aviation accident analysis: The Human Factors Analysis and Classification System” by D. A. Wiegmann, and S. A. Shappell, 2003, p. 51. Burlington, VT: Ashgate Publishing Company.](image)

**Errors.** Errors can be classified into three types: skill based, decision, and perceptual errors. Each of these is summarized below.

**Skill-based errors.** Skill-based errors usually occur when a pilot performs an action without conscious thought or attention. These are basic actions that he does every day while
flying. Errors such as the breakdown in visual scan patterns, task fixation, and the inadvertent activation of controls are examples of skill-based errors.

**Decision errors.** Decision errors represent errors caused due to poor choices made because of lack of knowledge or due to intentional behavior. Decision errors can be grouped into three general categories: procedural errors, poor choices, and problem solving errors. Procedural decision errors occur in aviation, because it has a number of rigid set of rules and procedures to be followed (e.g., Orasanu, 1993). But sometimes in case of emergencies, inexperience or time-critical situations, the pilots apply or decide to deviate from a procedure. This may sometimes be a poor choice or sometimes even solve the problem.

**Perceptual errors.** Perceptual errors occur when sensory input is degraded; they include visual illusions, spatial disorientation or misjudgment of an aircraft’s altitude, attitude, or airspeed. Here, the illusion or disorientation is not considered as a perceptual error; but the pilot’s erroneous response to the illusion or disorientation.

**Violations.** Violations are classified into two distinct forms, as summarized below.

**Routine violations.** This form of violation tends to be customary by nature and often tolerated by governing authority (Reason, 1990). For example, supervisors or the controllers in the tower tolerate pilots who fly into marginal weather, thereby creating a possibility of a mishap.

**Exceptional violations.** Unlike routine violations, exceptional violations appear are unpardonable actions, not necessarily indicative of individual’s typical behavior pattern or condoned by management (Reason, 1990). For example, pilots engaging in prohibited
maneuvers. These errors are considered exceptional because they are neither typical of the individual making them difficult to predict. These violations have more probability of resulting in a dramatic accident.

**Preconditions for Unsafe Acts**

Two major subdivisions, as seen in Figure 9, of unsafe aircrew conditions were developed; substandard conditions of operators and the substandard practices they commit

---

**Substandard conditions of operators.** Substandard conditions of the operators are characterized by the categories summarized below.

- **Adverse mental states.** Adverse mental states relate to those mental conditions that affect the performance of the air crew such as the loss of situational awareness, task fixation, distraction, and fatigue due to sleep loss or other stressors including increased workload. Also, personality traits and attitudes such as overconfidence, complacency, and misplaced
motivation are also included in this category. If members of the aircrew, maintenance
department and the controllers are mentally fatigued, the possibility of an occurrence of an
error increases. Again, overconfidence, arrogance and impulsivity will influence an operator to
violate procedures.

**Adverse physiological states.** Adverse physiological states refer to those medical or
physiological conditions that prohibit safe operations such as visual illusions, spatial
disorientation, physical fatigue etc. When a pilot is ill, but still goes on to fly the effects of
visual illusions and spatial disorientation may affect his performance in the cockpit. Thus, it
becomes necessary for the supervisor to examine the condition of a pilot before he enters the
cockpit. If a supervisor fails to do this duty, the consequences may be appalling.

**Physical/mental limitations.** Physical/mental limitations refer to those instances when
mission requirements surpass the capabilities of the individual at the controls. For example,
while flying at night the human vision is severely impaired, yet the pilots sometimes do not take
enough precautions while doing their maneuvers or doing take off and landing. Unfortunately,
when precautions are not taken, the result can be catastrophic. Or sometimes, a pilot may not
be blessed with quick response time due to inexperience or due to genital reasons or his body
does not suit the requirements of a high- G environment. Now, when they are in an emergency
situation demanding them to respond quickly, they are helpless.

**Substandard practices of operators.** Substandard practices of the operators are divided
into two categories as summarized below.
**Crew resource mismanagement.** Crew resource mismanagement accounts for occurrences of poor communication and coordination among supervisors or the air traffic controllers and the aircrew or the maintenance personnel. Effective communication and coordination should be present among the personnel for flow of important information and for producing a quality aircrew. The lessons learnt from the Valujet case study necessitate the importance of an efficient aircrew and effective coordination and communication among the personnel.

**Personal readiness.** Personal readiness failures occur when an operator or controller is not physically or mentally prepared for duty. For example, violations of crew rest requirements leading to physical fatigue or adverse mental states, and self-medicating will affect pilot or aircrew performance. These might have calamitous consequences. Thus pilots and other crew members should be physically and mentally fit and ready to do their tasks efficiently.

**Unsafe Supervision**

As explained before, efficient supervision plays a major role in producing high quality aircrew. Errors in this level often lead to miserable consequences. Unsafe supervision, as seen in Figure 10, can be classified into four categories: inadequate supervision planned inappropriate operations, failure to correct a known problem, and supervisory violations.
Inadequate supervision. A supervisor is required to provide guidance, training opportunities, leadership, and motivation to produce an efficient air crew. In cases where proper training was not given, leading to aircrew coordination skills being compromised and if the aircraft hits an emergency situation, the probability of an error being committed and the probability an accident would increase drastically. Also, a bad supervision leads to an incident of a violation, again increasing the probability of occurrence of an accident/incident.

Planned inappropriate operations. Planned inappropriate operations relates to errors committed when the crew is put into risky situations or when their work schedule becomes overwhelming leading to physical and mental fatigue in normal and emergency situations. The supervisor should be well organized with his time schedules and should be able to provide high quality trainings to the crew to tackle any situation in the best possible manner. Any shortcomings in these will have a fatal consequence.

Failure to correct a known problem. Failure to Correct a Known Problem refers to those instances when a problem among individuals, equipment, and training are made known to the supervisor, but still remain uncorrected. The failure to correct the deficit on the part of the
supervisor will have a dire consequence in the end. For example, when a pilot is ill but still given a flight status by the supervisor will lead to adverse mental and physical states while at flight ultimately leading to a fatal accident. Thus, it is considered a violation on the part of the supervisor, if he fails to correct a behavior or a problem which leads to an unsafe atmosphere.

**Supervisory violations.** Supervisory violations refer to those instances when supervisors violate the rules and regulations. For example, letting a person fly without a pilot license is an exceptional violation on the part of the supervisor.

**Organizational Influences**

These latent levels of human errors are known to influence the effective working of an organization. Reason’s model also allows us to investigate the influences of these levels in the causal sequence of the failure. Organizational Influences, as seen in Figure 11, are usually related to resource management, organizational climate, and operational processes.

![Organizational Influence Diagram](image)


**Resource management.** Effective resource management includes the allocation and maintenance of organizational assets such as human resources, monetary assets, and
equipment and facilities. An efficient resource management usually results in effective time and cost benefits. However in times of severe economical downfall, when funds are being cut, the supervisors might not be motivated to work efficiently or might be forced to compromise on the aircrew training. Also, when adequate facility is not being provided for training and maintenance, the skill level and hence the quality of the aircrew, and also the performance of the aircraft are put at stake.

**Organizational climate.** Organizational Climate refers to the working environment within the organization such as the organization’s structure, delegation of authority and responsibility, communication channels, that influences the worker performance. An organization’s policies and culture echoes its climate. Policies for hiring and firing, promotion, raises, sick leave, overtime plays a major role in the performances of its personnel.

Cultural elements such as values, attitudes of an organization, also play a major role in satisfying the social needs of the personnel. Thus, when these needs are satisfied by the organization, its workforce will be motivated to work efficiently bringing about a positive safety culture. Deficiencies in a healthy organizational climate increase the probability of an accident.

**Organizational process.** Operational Process refers to business decisions and rules that administer the day to day activities and processes that take place in an organization. For example, establishment and application of operating procedures for maintaining checks and balances between the workforce and management. These procedures determine the cadence of the operations within the organization by directly influencing work schedules, deadlines etc. We have already discussed the consequences of the deficiencies found in the above mentioned factors. An organization should also be proactive and have official procedures to tackle
contingency situations. This would help the other levels of the organization to continue working efficiently even in emergency situations and provide a safe environment for the pilots.

Appendix B: Retrospective Interview Questions

The following list of open-ended questions was used by Hoffman et al. (2009) in their interviews with three of the six systems engineers.

- What is the most complex system you have been involved in developing?
- What is the most complex system development team you have been a part of?
- What specific systems development project has had the most influence on your approach and strategies for complex systems development?
- What overarching systems development models and philosophies were used?

For the chosen development effort:

- What systems development model was used?
- Please describe the organizational structure used to support the effort.
  - How large was the organization/team?
  - Was it a hierarchical or a flat organizational structure?
  - How was work divided among groups?
  - How large were groups/teams?
  - How was group/team membership determined?
  - Did members of a given group/team have different or shared task responsibilities?
- What processes and mechanisms were used to communicate and coordinate within and across teams, e.g., with regard to:
  - Specific development sub-objectives/milestones?
The overarching design/vision?
Progress?
Changes in requirements?
Design updates?
Customer/user feedback?
Schedule changes?
Design deficiencies?
Other...

In the selected effort, who or what team was responsible for designing the aspects of the system that would affect user performance, i.e., that would be interfacing with users? In particular, what types of expertise did team members have?

What did this team contribute to the system design?
Was this aspect of system development successful?
Did the contributions of this team make it into the developed system?
Did their contributions add to the success of the effort? In what ways?
By what processes did their contributions become incorporated into the larger system design?
By what processes did their contributions become incorporated into developed product?
To the extent they were successful, what contributed to their success?
What were impediments to their successful contribution?
➢ What did this team do right in terms of coordinating with the team and contributing to the design?

➢ What could this team have done to better coordinate with the team and contribute to the design?

What articles were adopted to support the effort (e.g., what different types of plans, diagrams, etc? How were requirements documented and communicated)?

What procedures were used to minimize communication/coordination breakdowns? and what procedures were used to recover from communication/coordination breakdowns? (within and across teams)

How to develop effective functional requirements?

What methods were used to identify and build requirements?

How do you handle changing requirements?

How did the business side come upon their expectations/requirements?

What processes and mechanisms were used to assess quality?

What were the sources of conflict within the development team as a whole during the effort?

What strategies were used to resolve conflicts or developed to avoid future conflicts?

Was this development effort considered successful? Based on what criteria would you consider it successful?

Was the effort successful from the standpoint of the intended users? Was it delivered to the users? Did it help them perform their work, allow them to do their work more quickly or efficiently, or allow them to do and/or understand more? What was the
experience of the intended users once the system was delivered (if it was delivered)?

Did it meet user expectations? How was user satisfaction assessed?

Were changes to the system requested following delivery? If so, what types of changes?

How were they accomplished?

What specific systems development experience, class or book has had the most

influence on your approach and strategies for complex systems development?

Based on your general experience, what do you see as the most common problems and

most serious problems encountered in systems development?

What strategies do you think might help prevent or minimize these problems?
## Appendix C: Coded Interview Transcripts

### Table 10

**Coded Interview Transcripts**

<table>
<thead>
<tr>
<th>Data-Driven (Bottom-up)</th>
<th>HFACS (Top-down)</th>
<th>Interview Chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-OC-IPC</td>
<td>RM</td>
<td>Two lead engineers, including me, were stolen from another team.</td>
</tr>
<tr>
<td>DT-OC-AM DT-OS-D</td>
<td>OC</td>
<td>So we had 2 leads, 4 regulars, and 4 contractors, all of whom had not worked together. There was no established trust among these groups, and their management.</td>
</tr>
</tbody>
</table>
| DT-OC-DP DT-OS-D        | OC               | • We divided the work.  
• I took a part—a big box from the high-level view.  
• It was a type of work where if you did it using classical 1-up solutions, it’d have been boring work. I wanted it to be more interesting. Sees that now as his motivation.  
• I’d use more elegant ways than classical methods. These would be object-oriented techniques using advanced methods like design patterns (I had been studying these things on my own time).  
• I want the challenge of moving methodology ahead. |
| SDTA-D                  | NIL              | I developed a high-level framework for communicating to the 20 protocols/system blocks based on XML. We used an XML messaging program manager instead of 20 different messaging protocols. |
| DT-OS-D                 | NIL              | [Prompted to return to timeline of the story]  
• First, we joined the team and divvied up the work  
• I immediately knew what sort of approach I wanted to use.  
• Probably 2nd week, I chose the XML based approach.  
• Then I went off and did the work. |
| DT-OC-IPC               | OC               | • During the first few weeks, I got to know a young engineer. I hooked up with him as my protégé. He was assigned to another lead. I persisted to get this guy on my team. May have traded two contractors for him. |
That guy was in my face: “What do I do?” and forced me to lay down a framework so I could feed and engage him.

I had to think of the “little boxes” inside my big boxes so I could describe his work logically. There is not a good way to communicate in Software-ese: “I need these inputs and these outputs, and this is how I need it on the inside.”

He’d work for two days and then come back to me.

[I think he was prompted here regarding previous comment about scared manager/boss]

Bits and pieces of what we were doing trickled out as we worked so he [the manager] could tell we were doing something adventurous.

He didn’t demand details. I didn’t provide details or communicate. I believe you should do everything you can without involving your management. But we had no history [of working together]. He chose to trust me rather than confront me, but he was very stressed the whole time I found out later.

In the last two weeks, it seemed to be coming together. We trickled out details to my peer lead. He was impressed and told his contractors and then they wanted it [the XML framework] for something else. The contractors went away and tested it and I was surprised when they came back and said ‘we like it, and it will save us lots of time.’

You mentioned other contractors and team members...that you traded 2 contractors for someone. Did you interact with others? The other lead and I had been introduced to everyone. We had a lot of autonomy. The buck stopped with us. The managers were clueless. We had decent trust and a decent rapport (the 2 leads). He got there a before I did and had already grabbed Jared (the young software engineer mentioned above). I came in a week later and he had all the people. Managers told him to give me people. I joked for a week about how all I needed was Jared and he could have the rest. Finally I told him I’m serious, You get 4 contractors, I get Jared. At that point, I had the guy (Jared), the autonomy, the concept. Nothing could stop me.
Did you interact with the other lead? Did he have to know what you were doing? • We’d have meetings and see each other once in a while. I later realized maybe he could use what we were developing.

• We were on completely parallel paths. I didn’t think our work was relevant to his at first; didn’t know what he was doing. Didn’t know he could use it…but he later benefited from it tremendously.

• The overriding purpose of segregating blocks (of high-level diagram) is it allowed us to work in parallel and independently until integration.

• We had weekly formal meetings and took advantage of the fact that the managers were clueless. Used them to report resource issues—were contractors being used appropriately.

Schedule slippages would have been reported.

• With highly divided duties, you don’t have to talk. The things we had in common were:
  o Software had to integrate into computers on the locomotive
  o Integration of his and my pieces

• There was a thread between him and me—whether software would run on embedded PCs. This was a risk for us both.

• Risk mitigation is a parallel thread (for leads). As your teams are working, Leads work on risk items.

What did you do on a daily basis? What did you work on, what did Jared work on? How did you work together? He didn't have the luxury of being handed a stack of material to study and then meeting to go over them.
| DT-OS - M  | NIL  | The first one is, you know, in terms of difficult or challenging - when you get into structured environment like we were on...tactical trainer. We were tasked to replace the shore based systems that they have both in the Atlantic & Pacific... [inaudible]...perfect routine... [inaudible] VMH [inaudible]...... |
| DT-OS-M  | OC   | It was a government led effort so you had 6 companies with the government technical lead so all the companies were kinda running off doing their own little things and you weren’t getting consistent direction from the government picking |
| DT-TC-GS | NIL  | the government lead saying, go this way, go that way. |
| DT-TC-COOR  | OC   | So you have a lot of different approaches and ideas and nobody centralizing them all in one path folder. So how do you pull something like that together when nobody’s in charge, if you will? |
| DT-OS-LD  | OC   | Your government lead is supposed to be in charge but they are not strong enough as a leader to kinda really grab the team and say, we’re gonna go that way. |
| SDTA-REQ | OC   | Especially if your customer doesn’t know exactly what you’re looking...what they’re looking for and the lots of different alternatives and options that are out there. |
| DT-OC-IPC  | RM   | And you get 6 different companies coming in selling them on the way they wanna go because it suits them the best. How do you pull that together? |
| DT-OS-LD  | RM   | There’s really no set answer for that you know, how to do that. You hope that your government lead nominates somebody and says [inaudible] they’re in charge and move out. We’re gonna listen to them and move forward but it doesn’t always work that way. |
| Nil  | NIL  | Most of the other things were just technical challenges more than problematic challenges and how to actually implement your process through something like this. |
You may have a system design process and development process that you want to put in place but if company A over here doesn’t necessarily follow those you’re gonna get different products that are gonna come out from that type of thing so, ahmm... that’s probably the best one.

I can relate back to the same type of thing where ahmmm you’re getting ahhm when you have those distributed teams and one person has a solution that you don’t necessarily agree with because it promotes their ahh agenda, or their product or you know, those types of things that you feel going this direction would be better off.

In defense, another example, of [totalship] training, things that we’ve done in the past where you have real legacy people that are involved with a program they’re very stone pipe in the way they think and you’re coming in with a new technology or new way to approach it. They don’t always see that as the right way because it’s gonna replace them as the entrenched one with the government customer and ahhh, so you constantly fight when you try to introduce new technologies and new ways of doing things that they don’t always want....(laughs)... welcome those...there’s even your government customers for some times ahmm, they don’t recognize the power of some of the capabilities.

And I think we ran into that with Gertz a little bit. Our [inaudible ]solution that we had was a complete change from what they’re used to and I don’t think the presentation came across as well as it should have in a proposal effort...(laughs). Ahmm, you know, so you run against the political factors of you know, hey, we like this subcontractor over here and we like their reputation so we’re gonna go that way and not necessarily go what we’ll call disruptive technologies and allow you to introduce new ideas new concepts that goes away from what they’re used to. Replace some of those legacies, capabilities that are there.

I can go back a few years to some stuff that we’d done on, for the Army...on like close combat tactical trainer and it’s very similar types of problems where you’ve got an integrated development team trying to execute and at that time Lockheed Martin was the prime trying to get...and they had supposed oversight over all the other companies trying to get things to execute on schedule and those types of things.
But they weren’t very effective at integration?

Not initially. I mean any large kinda program, I mean, you can bid it as a 3-year, 4-year program, it’s gonna take you 6 years to get it done. When you’re talking integrating multiple companies like that and ahh if all the requirements where your customer is not exactly sure what they want and you doing a spiral development so you do something they get a chance to review it and make comments well, those comments that they’re providing based on that review you’re already 6 months into the next spiral and they’re making cuts so you’re constantly playing catch up. And so you’re spitting spiral spiral may be December ’05 but you’re getting comments back in June of ’05 and there’s no way you’re gonna get it rolled in until that final delivery so you’re gonna be a year or better out from when you’re supposed to be finally delivering.

The government has to recognize those types of things when…when they’re providing that feedback in the spiral processes, that it’s not gonna end on a set date they have to have plans out. A lot of them have gotten a lot better at that, at planning things that go past the PCPs or change requests or things like that they’re … you know, you’re not done December ’05 you’re gonna make a drop in December ’05 and there are gonna be changes that need to be…you gotta plan and budget those types of things.

We were an open system so we didn’t build you know like the traditional Lockheed Martin proprietary solutions that nobody else could play in unless they wanted to give you that information which they typically didn’t want to do.

At <my company>, they have extensive processes. That has to do with the CMMI thing.

And we’re a level 4 wanting to become a level 5
Especially at level 5, that’s the highest. When you’re at that level...there’re benefits to it, but there are some frustrations with it, too at times.

Maybe we’ll start off with you telling us a little bit about your background and career. Do you have a label, like a systems engineer?

That’s my current job title. Backing up, I have an Industrial Engineering (IE) undergrad, a Master’s in IE within manufacturing engineering. I was doing a lot with automation. Started with General Electric (GE) in their automation program. They had a big push in “factory of the future.” So they were trying to train some experts and I got selected for that. And I moved around. Every six months they moved you to a different automation area.

But then as I was starting to graduate from the program, GE said oh, factory of the future is not as big as we thought it would be. So we’re closing that whole department. ...so the training program got cancelled.

So then I left there and ended up going to a small company for a year. And that was when the tech bust happened and got laid off from there.

Spiral and those... What we’re planning on for this current project is doing—they call it modified spiral in the sense of it’s more of, we’re kind of taking different builds, and build that, and then go back, and do the second build, and the third build. They’re overlapping with each other but they are staggered so that you can be adding lessons learned from the first one back into the second and third ones.

The first one only does simple communication between all of the devices. Might have it do a simple self test or something. Second one adds more functionality and the third one ads full functionality.
In general, when you’re using this modified spiral, how do you develop the requirements?

We try to develop requirements for the whole system. Because you can’t really get into the spiral unless you really know what you’re building for the whole system. And even the preliminary design which is the second step, we pretty much do all of that because you want the whole design structure that will deal with the whole functionality. As you get into detail design is where you start separating it down.

For our system, our hardware is going be (more and more systems are going this way today) is going to be pretty generic hardware there are going to be regular P.C’s that we’ll be running on. There will be some dedicated cards that we are running on to filter the incoming signals. But it’s going to be fairly standard. Years ago there was a very tight connection between the hardware and software.

So like one of the discussions we are having right now is laying out the schedule, and always in the years in the past we have had all the key reviews where you review the requirements, and you do the preliminary design review and the critical design review. So as you go through it’s always been hardware and software. In the Hunts program we are proposing that we are going to stager those. The hardware is running much ahead of the software because it’s like I said, pretty standardized. They’re using some off the shelf stuff. But they were saying “How can you do that? You need to keep them in line” and it’s because of the way we are using more of these work station type approaches. The software designs are going to run in generic work stations. It’s not as critical to have them as closely tied together.

Ok, one area that was where we kind of went a different way than was an approach was when I was working on the F-22 program down here I was kind of like leading the hardware integration and getting the system working than doing the low level work and all that.
And there was a different group developing the functional software that was going to run in all the hardware. And they had planned on doing these fairly detailed testing of everything and they had planned to do it as you would normally do it.

With a software debugger where you could insert values, and then control the flow of the code and verify, when you put these designated inputs in you get this output and you can test the functionality of the software. The problem was that the debuggers system we had since this was custom hardware that was that sophisticated and you also had parallel processes and things going on. So we ended up with a lot of problems about being able to configure it as well as it wasn’t as controllable.

They wanted to do that testing. And so that’s one of the areas where I got quite involved. We ended up having to develop these very complicated procedures and stuff to try to do something similar to that but never did match the real plans we had. So that is where we kind of went a different path and had to come up with these work-arounds to do the actual testing.

I don’t know as much as you’d say this is a different path, the current projects I’m working on, in the past certainly they are, and most of the places I’ve worked, the customer normally comes in and says “Ok here are the system level requirements, this is what we want you to build a system for. And we are giving you the requirements.” On this modernization project they actually came, they got money from congress, and the Navy said “Well we want to modernize these. Go figure it out.” So they actually brought us in early that’s what we have been doing the last year, determining the requirements for the system so we were helping them determine how best to modernize it so we are coming up with our own requirements.

So we developed all these requirements given to them, and we reviewed them and discussed. And now we are actually doing the job and it’s really strange because now all the stuff we wrote is coming back as requirements.

This project was complex and had a complex team—collaboration among 15 different universities
<table>
<thead>
<tr>
<th>NIL</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>We were taking daily feeds of news and magazines and automatically loaded them into the website; we’d mark them up for reading level and categorize. It was redone incrementally. We wrote the whole thing.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PM-CxM DT-OC-DPSDTA-D</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the chosen development effort:</td>
<td></td>
</tr>
</tbody>
</table>

3. What overarching systems development model/s and philosophy/ies were used?  
- I tried to keep things modular, tried to adhere to a rough framework...system was broken into chunks, any chunks.  
- We used object-oriented programming...a lot of the code was old...hundreds of providers were using their own format and we had to translate that...“filter” it using our own filter.  
- The object-oriented approach helped with re-use.  

<table>
<thead>
<tr>
<th>DT-OC-DP EP-S</th>
<th>OP</th>
</tr>
</thead>
</table>
| We tried to make things backward compatible; we had to work with both old and new.  
- Any time we changed something that ran across multiple modules, we had to test.  
- It was more of a spiral development effort...make changes to pieces then roll them out to production—then make changes to other pieces, etc.—tried to produce subsets at a time rather than hold everything back.  
- One major change was that we changed to distributed multi-host software that required change throughout the software—6 months of development; 1 month of test. |

<table>
<thead>
<tr>
<th>DT-OS-D</th>
<th>OC</th>
</tr>
</thead>
</table>
| 4. Please describe the organizational structure used to support the effort.  
- My team was focused on getting feeds and loading them into a database. It had about six people on it.  
- Another team was focusing on how to retrieve the search results. I had 2 to 3 people.  
- Another team of 3-4 people was handling web-based development.  
- There were a lot of off-shoots to different types of products.  
- Each team had a lead; some people handled daily reports generated; another (person) handled feeds from satellites, etc., within the team people were specialized but able to take over if someone was out. |
5. What processes and mechanisms were used to communicate and coordinate within and across teams, e.g., with regard to:

<table>
<thead>
<tr>
<th>Team 1</th>
<th>Team 2</th>
<th>OC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-TC-CfM</td>
<td>DT-TC-COOR</td>
<td>- Within the team:</td>
</tr>
<tr>
<td>DT-TC-BCG</td>
<td></td>
<td>o If we had a problem, we’d go to one another</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o We had monthly meetings within the team</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Kept track of goals and taught one another new technologies for knowledge transfer.</td>
</tr>
</tbody>
</table>

- Between teams:
  o Team leads and supervisor had weekly meetings on current objectives; this was a formal communication
  o Lots of informal communications
  o At the end of every week, the leads made a report of goals, deadlines, and activities from the past week. Gave this to supervisors and distributed among team members. We used verbal communication to clarify, and this kept us pretty well informed. We were responsible for day to day operations and the supervisor needed to know. This is how we communicated concerns and needs to others and tied together teams.

6. What procedures were used to minimize communication/coordination breakdowns? What procedures were used to recover from communication/coordination breakdowns? (within and across teams)

<table>
<thead>
<tr>
<th>Team 1</th>
<th>Team 2</th>
<th>NIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-TC-CfM</td>
<td>DT-TC-BCG</td>
<td>Usually, if there was a problem due to a communication breakdown, the person’s understanding of the problem would be sent out by email, detailing the problem to everyone involved, followed by a meeting.</td>
</tr>
</tbody>
</table>

7. What artifacts were adopted to support the effort (e.g., what different types of plans, diagrams, etc? How were requirements documented and communicated?)

<table>
<thead>
<tr>
<th>Team 1</th>
<th>Team 2</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-TC-BCG</td>
<td>PM-QM-CM</td>
<td>- whenever we had a new content provider. We’d send the new content provider a form in which they would specify their data format, and then we’d sign off on it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- We’d write up documentation for the software. Big sections of comment on the use of a module were written at the beginning of the code for that module.</td>
</tr>
<tr>
<td>PM-QM-CM</td>
<td>DT-OS-D</td>
<td>In the case of a big change or a new technology, we’d write standalone documentation for it. The Lead ensures this happens.</td>
</tr>
<tr>
<td>DT-TC-BCG</td>
<td>SDTA-T</td>
<td>DT-TC-IE</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A number of different reports were generated weekly:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o My report: a high level report of activities, problems, goals, and deadlines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o One of my people: What documents came into the system and from where, broken down statistically (how many documents, what types...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Time to time we were asked to write special reports...usually for business reasons, and sometimes for technical reasons, e.g., if a server was becoming overloaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. What methods were used to identify and build the ‘full’ set of requirements?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We would find out what the requirements were through just face to face discussion…”the business people” would tell us what they were after. Our users were internal.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We had procedures in place for a new content provider. We’d make a pass at the software and then send it to a QA person to make sure there were no problems. The business side would look at the final product to see if it was what they had in mind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>How did the business side come upon their expectations/requirements?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>They identified more formal requirements based on market research done on the business side. They’d look at other products, talk with users, and do some formal surveys.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. What methods were used to create the system design and the design artifacts?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- We’d do a mockup and the business side would give feedback...we’d do a couple of iterations of this (mockup-&gt;feedback) depending on the complexity.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical details that weren’t visible to the outside were our prerogative...we just had to ensure we met certain constraints:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Had to ensure compatibility with other modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Search time couldn’t be more than 5 seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Disk space, server resource issues (resources brought in to handle expansion)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Team</td>
<td>Answer</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>10. What processes and mechanisms were used to assess progress/status? and 11. What processes and mechanisms were used to assess quality?</td>
<td>SDTA-T PM-QM OP</td>
<td>We did assessment via an automated system we built to monitor the availability of the system/website and to make sure that what gets turned up in the searches makes sense.</td>
</tr>
<tr>
<td>1. What is the most complex system you’ve been involved in developing? - What about this system made it complex?</td>
<td>NIL NIL</td>
<td>C3Core, because it’s been developing for close to 10 years and due to ever-changing user requirements, functionality sets, and architecture, and making sure it’s set up for future changes.</td>
</tr>
<tr>
<td>2. What is the most complex system development team you’ve been a part of? - What about this team made it complex?</td>
<td>DT-OC-DP SDTA-REQ PM-ChM NIL OC</td>
<td>C3Core, because over the course of 10 years, people come and go bringing different skills and tool sets. It’s a very dynamic team.</td>
</tr>
<tr>
<td>3. What specific systems development project has had the most influence on your approach and strategies for complex systems development? - At a high level, describe how the experience has influenced you.</td>
<td>PM-CxM DT-OS-D RM</td>
<td>The biggest thing is getting everyone synchronized; forming group consensus is very very difficult. Programs don’t scale linearly; as the number of people grows, the amount of information increases exponentially. Teams with more than about 10 people don’t work because you can’t get consensus. Divide the team into subdomains or projects and coordinate between groups. If you don’t subdivide the team correctly, it won’t work (allocate work to appropriate people...)</td>
</tr>
</tbody>
</table>
For the chosen development effort:

<table>
<thead>
<tr>
<th>SDTA-D</th>
<th>OP</th>
<th>4. What systems development model was used? Evolutionary and Spiral type of design. Each program (add-on to C3Core) brings its own constraints.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM-BS</td>
<td>NIL</td>
<td>5. Please describe the organizational structure used to support the effort. Small teams of 1-4 people working on a task using very short 1-2 week spirals of development. Short spirals help keep people from diverging in their thinking; provide frequent checkpoints. Everyone has an expert domain area (e.g., GIS system, database), and everyone is cross-pollinated into other areas, e.g., when a subsystem spills into the GIS system the two associated teams cross-pollinate (=how?).</td>
</tr>
<tr>
<td>DT-OC-DP</td>
<td>RM</td>
<td>Team is not big enough to have dedicated designers, testers, coders; Generally, the team is involved through the requirements phase, design phase, etc. with the best team member for each phase designated as the point person. The team is augmented with external supervision in the guise of QA. This role reviews requirements, design, and implementation.</td>
</tr>
<tr>
<td>DT-OS-D</td>
<td>OP</td>
<td>6. What artifacts were adopted to support the effort? Informal processes are becoming more formalized. We have a project plan, test plans, schedules, etc. that are required by CMMI. Before, we used white-board sketches, oral schedules, weekly progress meetings (which still happen). CMMI helps or hurts? In a fire drill, the formalities are difficult; we have adapted lightweight versions of the processes. Done incorrectly, CMMI can kill your program; it’s easy to generate docs; hard to generate good docs. A bad doc can really throw a team off.</td>
</tr>
<tr>
<td>NIL</td>
<td>OP</td>
<td>CMMI helps or hurts? In a fire drill, the formalities are difficult; we have adapted lightweight versions of the processes. Done incorrectly, CMMI can kill your program; it’s easy to generate docs; hard to generate good docs. A bad doc can really throw a team off.</td>
</tr>
<tr>
<td>PM-QM-CM</td>
<td>OP</td>
<td>Classic lesson learned: when I was a project manager at DuPont, the heating system steam pipe came to the heating system with a gap of 6 inches. I couldn’t sign off on the system, but the engineer said, “Well, look at the document,” and sure enough... Once an artifact is created it has to be maintained.</td>
</tr>
</tbody>
</table>
Paperwork should lead you forward, not be a diary.

<table>
<thead>
<tr>
<th>PM-QM-CM</th>
<th>OP</th>
<th>CMMI should involve creating an artifact trail so you can backfeed a design implementation all the way back to the money. Level 3 (level used by interviewee) involves gathering artifacts.</th>
</tr>
</thead>
</table>

7. **What methods were used to identify requirements?**

*Requirements come at different levels. We get high-level requirements...well what does this really mean?*

Requirements include:
- Program/Project Requirements
- Software Requirements
- Test Requirements

Depends on your business model, too. We’re using other people’s money to develop software we can sell.

<table>
<thead>
<tr>
<th>SDTA-REQ</th>
<th>NIL</th>
<th>How do you identify requirements? If the customer gives you good requirements, then you analyze them for gaps and conflicts: Are they good? Missing things?...and as you do that, what implied requirements need to be there to build this monster?</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SDTA-REQ</th>
<th>OP</th>
<th>How do you identify gaps and conflicts? We take a schoolbook approach to decomposing the requirements given us by the customer (“Fish out of water...” —??) High level functional descriptions is where we start, via storyboarding or CRC cards (mini use-case; we write a function on each card and group the cards [card sorting]). We also use brainstorming and what-ifs. Use cases are problematic because they are ambiguous with respect to language.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PM-ChM</th>
<th>OP</th>
<th>How do you handle new and changing requirements? Build an open and flexible architecture. It’s always like a crapshoot, you have to do it at least 3 times to get it right.</th>
</tr>
</thead>
</table>
What’s “doing it right?”
Doing it right means it survives the test of time and survives re-use. By about the 3rd time they’ve got it right, and on every iteration it goes faster.

A spiral, evolutionary prototype design approach was used for FCS. 
There’s new software, new hardware, new languages...every few months. Any program has to be adaptable. You have to understand requirements will change and have to insulate yourself from that. Programs, including C3Core, take smaller chunks, e.g., four 1-mth blocks of development, demo once a month, wait for change requests, and then assess new requirements and inform customer regarding costs, tradeoffs, etc.

1. What is the most complex system you’ve been involved in developing?
IE NATOPS.
What about this system made it complex?
team was a relatively large group and was a changing group.

I came in late

It was a tablet-based application. Tablets were a new technology for me. I wasn’t sure what we were trying to do or what could be done

2. What is your most complex system development team involvement?
- What about this team made it complex?
- 2 ½ software engineers; the ‘1/2’ was coordinating with Penn State
- project manager
- subject matter expert

For the chosen development effort:

3. What overarching systems development model/s and philosophy/ies were used?
- I’m usually working on two to three projects at a time. This affects the sequence and how and why I do things.
In retrospect, I’m a proponent of agile development. Don’t over-design upfront. Do just enough design to get implementation going and to meet short term objectives. Get functional specs going, get some software going, and then reconcile the two. The size of the system makes this more doable. Since we were doing research...what’s possible? This model supports exploration of technology and what it allows you to do.

e.g., Word and browser and cell phone: combine them, validate, then design more based on the results. You don’t know upfront if it will work.

Has to be communication at some point between software and cognitive systems engineers so that neither diverges too far from the other or that the application is headed toward the unusable (due to a divergence from what’s possible).

- Hand-in-hand development
- XP (extreme programming); agile development

4. Please describe the organizational structure used to support the effort. For example:
- how large was the organization/team?
- was it a hierarchical or a flat organizational structure?
- how was work divided among groups?
- how large were groups/teams?
- how was group/team membership determined?
- did members of a given group/team have different or shared task responsibilities?

The organizational structure was somewhat hierarchical. There was a functional group and a financial group. Functional was divided into two groups--one focused on what we were going to do (the functional people) and the other group focused on how we were going to do it (the implementation people).

Team membership was defined based on availability, philosophy, and had to have someone who could analyze technology.

The 'How we’re going to do it' team had to include people who could: ID hardware; make the interface build itself dynamically; figure out how to switch from day to night lighting; and architect the overall application.
5. What processes and mechanisms were used to communicate and coordinate within and across teams, e.g., with regard to:
- specific development sub-objectives/milestones?
- the overarching design/vision?
- progress?
- changes in requirements?
- design updates?
- customer/user feedback?
- schedule changes?
- design deficiencies?
- other...

<table>
<thead>
<tr>
<th>Team</th>
<th>Communication</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-OC-DP</td>
<td>NIL</td>
<td>Specific development sub-objectives/milestones? The overarching design/vision? Progress? - We used paired programming: code is developed quickly (although a little less quickly than with one programmer) and it’s better code. The two programmers switch-off between typing and reviewing where code is going from a higher level. The two personalities have to have a common programming style (often a problem). The two people use design patterns to communicate and share concepts of development. - Paired programming + Brainstorming + Design + Knowledge Exchange - Design patterns= similar techniques are always used to do certain things. Patterns characterized in terms of: when to use, when not to use, trade-offs, which patterns work well with others...not canned code...e.g., proxy, façade...</td>
</tr>
<tr>
<td>DT-OS-D</td>
<td>OC</td>
<td>Team needs to work closely and with similar styles</td>
</tr>
<tr>
<td>DT-TC-IE</td>
<td>NIL</td>
<td>Communication at the software level occurred during implementation (via paired programming approach) - Other communication took place at a white board. This was used for working out and communicating higher level constructs</td>
</tr>
<tr>
<td>PM-BS</td>
<td>OP</td>
<td>Development milestones, etc. were established with “functional people”, based on what the functional people wanted and the “implementation people” could deliver. - Milestones get ordered based on which products the functional team has ready and a broader vision of the functional specifications</td>
</tr>
<tr>
<td>OP</td>
<td>Changes in requirements? design updates?</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-----------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| DT-OC-DP | - Agile development says you don’t write to accommodate future desires because it may be wasted effort (this person doesn’t try to accommodate specific future desires, but does try to build flexibility into the software and tries to build in functions that would support a wide range of users and circumstances).
|                      | - You write good code quickly and if the functional specifications change, you re-write the code quickly. |

<table>
<thead>
<tr>
<th>OP</th>
<th>How do you get the big picture early (big picture of the development goals, etc…)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-TC-BCG</td>
<td>- I don’t think there’s a good solution. It doesn’t go very well.</td>
</tr>
<tr>
<td></td>
<td>- Lots of meetings</td>
</tr>
<tr>
<td></td>
<td>- Some documentation. Documentation is difficult—it takes a lot of effort but may not be what you want.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OP</th>
<th>Interface: When will it work? When won’t it work? Software people need to the software to be able to handle 100% of the cases...functional requirements give a solution that addresses 70% of the cases (or less)...and the users get boxed in. To handle this, go back and forth between the software implementers and those developing functional requirements to address the other 30%.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDTA-REQ</td>
<td>SDTA-D</td>
</tr>
</tbody>
</table>
Appendix D: Calculation of Percent of Correspondence between Coders

The table presents five examples of high level codes assigned to data elements by Coders A and B. The third and the fourth columns show the process adopted to determine the correspondence.

Table 11

<table>
<thead>
<tr>
<th>Coder A codes</th>
<th>Coder B codes</th>
<th>Match</th>
<th>Percent of Correspondence assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Development Team – Organizational Structure – Customers, Users, and Developers</td>
<td>System Development Team – Organizational Structure – Development team</td>
<td>1/1 (one match between the high level code used; DT)</td>
<td>1</td>
</tr>
<tr>
<td>System Development Team – Organizational Culture - Internal Politics, Competition, and Conflicts</td>
<td>System Development Team – Organizational Culture - Internal Politics, Competition, and Conflicts</td>
<td>2/3 (two matches between three high level codes; DT, PM, and SDTA)</td>
<td>0.66</td>
</tr>
<tr>
<td>Management - Customer and User Expectations Management</td>
<td>Culture-Internal Politics, Competition, and Conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Development Team - Organizational Culture-Internal Politics, Competition, and Conflicts</td>
<td>Project Management - Customer and User Expectations Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Management - Customer and User Expectations Management</td>
<td>1/2 (only one match between two high level codes used; DT, PM) 0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Development Team - Organizational Culture-Internal Politics, Competition, and Conflicts</td>
<td>System Development Team - Organizational Structure - Development team</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Management - Customer and User Expectations Management</td>
<td>System Development Team - Organizational Culture-Internal Politics, Competition, and Conflicts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Development</td>
<td>1/3 (one match between three high level codes; DT, PM, and SDTA) 0.33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

104
| System Development Team-Organizational Culture/Internal Politics, Competition, and Conflicts | External Pressures-Politics, Competition, and Conflicts external to System Development team | 0/2 (No match between high level codes; DT, EP) | 0 |
**Appendix E: Taxonomy of Team Processes**

Table 12  
**Taxonomy of Team Processes**

<table>
<thead>
<tr>
<th>Process Dimensions</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transition Processes</strong></td>
<td></td>
</tr>
<tr>
<td>Mission Analysis formulation and planning</td>
<td>Interpretation and evaluation of the team’s mission, including identification of its main tasks as well as the operative environmental conditions and team resources available for mission execution</td>
</tr>
<tr>
<td>Goal Specification</td>
<td>Identification and prioritization of goals and sub goals for mission accomplishment</td>
</tr>
<tr>
<td><strong>Strategy Formulation</strong></td>
<td>Development of alternative courses of action for mission accomplishment</td>
</tr>
<tr>
<td><strong>Action Processes</strong></td>
<td></td>
</tr>
<tr>
<td>Monitoring progress toward goals</td>
<td>Tracking task and progress toward mission accomplishment, interpreting system information in terms of what needs to be accomplished for goal attainment, and transmitting progress to team members</td>
</tr>
<tr>
<td>Systems Monitoring</td>
<td>Tracking team resources and environmental conditions as they relate to mission accomplishment, which involves (1) internal systems monitoring (tracking team resources such as personnel, equipment, and other information that is generated or contained within the team), and (2) environmental monitoring (tracking the environmental conditions relevant to the team)</td>
</tr>
<tr>
<td>Team monitoring and backup behavior</td>
<td>Assisting team members to perform their tasks. Assistance may occur by (1) providing a team verbal feedback or coaching, (2) helping a teammate behaviorally in carrying out actions, or (3) assuming and completing a task for a teammate</td>
</tr>
<tr>
<td>Coordination</td>
<td>Orchestrating the sequence and timing of interdependent actions</td>
</tr>
<tr>
<td><strong>Interpersonal Processes</strong></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---</td>
</tr>
<tr>
<td><strong>Conflict Management</strong></td>
<td>Preemptive conflict management involves establishing conditioned to prevent, control, or guide team conflict before it occurs. Reactive conflict management involves working through task and interpersonal disagreements among team members</td>
</tr>
<tr>
<td><strong>Motivation and confidence building</strong></td>
<td>Generating and preserving a sense of collective confidence, motivation, and task-based cohesion with regard to mission accomplishment</td>
</tr>
<tr>
<td><strong>Affect Management</strong></td>
<td>Regulating member emotions during mission accomplishment, including (but not limited to) social cohesion, frustration, and excitement</td>
</tr>
</tbody>
</table>

# Appendix D: Team Dimensional Training Prebrief

## Table 13

**Team Dimensional Training Prebrief**

<table>
<thead>
<tr>
<th>INFORMATION EXCHANGE</th>
<th>COMMUNICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first dimension is information exchange. Effective information exchange allows the team to develop and maintain shared situation awareness.</td>
<td>The second dimension is communication. While information exchange deals with what is passed to whom, this dimension involves how that information is delivered.</td>
</tr>
<tr>
<td>The components of information exchange are:</td>
<td>The components of communication delivery are:</td>
</tr>
<tr>
<td>* Utilizing all available sources of information</td>
<td>* Proper phraseology</td>
</tr>
<tr>
<td>* Passing information to the appropriate persons without having to be asked</td>
<td>* Completeness of standard reports</td>
</tr>
<tr>
<td>* Providing periodic situation updates which summarize the big picture</td>
<td>* Brevity/Avoiding excess chatter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPORTING BEHAVIOR</th>
<th>INITIATIVE/LEADERSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>The third dimension is supporting behavior. This involves compensating for one another in order to achieve team objectives.</td>
<td>The fourth dimension is initiative/leadership. Anyone on the team can demonstrate initiative/leadership.</td>
</tr>
<tr>
<td>The components of supporting behavior are:</td>
<td>The components of initiative/leadership are:</td>
</tr>
<tr>
<td>* Monitoring and correcting team errors</td>
<td>* Providing guidance or suggestions to team members</td>
</tr>
<tr>
<td>* Providing and requesting backup or assistance to balance workload</td>
<td>* Stating clear and appropriate priorities</td>
</tr>
</tbody>
</table>

### Appendix G: Description of the final set of data-driven codes

**Final Set of Data-Driven Codes with Description**

<table>
<thead>
<tr>
<th>Final Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1) System Development Team</strong></td>
<td></td>
</tr>
<tr>
<td><strong>a) Organizational Culture</strong></td>
<td>Deals with work place environment, employee interactions, and the effects of their actions on the environment.</td>
</tr>
<tr>
<td>a. Internal Politics, Competition, and Conflicts</td>
<td>Deals with politics, conflicts, competition (for resources and credibility) within the organization, and issues arising as a consequence.</td>
</tr>
<tr>
<td>b. Attitudes about Management</td>
<td>Attitudes developed about the higher management</td>
</tr>
<tr>
<td>c. Development philosophy/Acquisition strategy</td>
<td>Philosophies and strategies advocated and used for development and acquisition processes.</td>
</tr>
<tr>
<td><strong>b) Organizational Structure (including changes)</strong></td>
<td>Deals with line of authority; their roles, responsibilities, and relationships; and Information flow between them.</td>
</tr>
<tr>
<td>b. Development Team: Roles, Responsibilities, and Relationships</td>
<td>Roles, Responsibilities, and Relationships</td>
</tr>
<tr>
<td>c. Customers, Users and Developers: Roles, Responsibilities, and Relationships</td>
<td>Roles, Responsibilities, and Relationships</td>
</tr>
<tr>
<td>d. Leadership: Roles, Responsibilities, and Relationships</td>
<td>Roles, Responsibilities, and Assignment</td>
</tr>
<tr>
<td>e. Team Member Characteristics</td>
<td>Capabilities of members and how they are grouped into a team</td>
</tr>
<tr>
<td><strong>c) Team Coordination and Communication</strong></td>
<td>Deals with team processes and how team members communicate and resolve various issues</td>
</tr>
<tr>
<td>a. Mission Analysis, Formulation, and Planning</td>
<td>Interpret and evaluate team’s purpose; identify main tasks, detect and respond to changes in operative environment; assess available resources.</td>
</tr>
</tbody>
</table>
b. **Goal Specification**  
Identify and prioritize goals for mission accomplishment

c. **Team Monitoring and Back-Up Behavior**  
Cross-check or monitor teammates for the purpose of assisting proactively; provide/request assistance.

d. **Coordination**  
Coordinate task sequence and timing.

e. **Conflict Management**  
Preemptive- Prevent, control, or guide situation to preempt possible conflict; Reactive- Work through task-related and interpersonal agreements among team members.

f. **Initiative**  
Advocate a specific source of action, team priorities, and task assignments.

g. **Information Exchange**  
Provide information for coordination purposes.

h. **Build Common Ground and Awareness**  
Updating and Educating the team members about every process to increase awareness

2) **System Development Project Management**  
*including management philosophy, attitudes, and practices*

<table>
<thead>
<tr>
<th>a) Planning / Re-planning</th>
<th>Planning the course of the project, projected outcomes, materials and resources needed, planning and being prepared for changes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b) Contracting</td>
<td>Deals with policies and decisions regarding contract agreements.</td>
</tr>
<tr>
<td>c) Budget and Schedule</td>
<td>Estimation of the budget and schedule, related errors and issues faced due to an overshoot of budget and schedule.</td>
</tr>
<tr>
<td>d) Complexity Management</td>
<td>Issues faced coping with the complexity of the system and its infrastructure, strategies adopted for optimization.</td>
</tr>
<tr>
<td>e) Quality Management</td>
<td>Deals with quality planning, control, assurance and improvement methods adopted and issues faced.</td>
</tr>
<tr>
<td>a. Configuration Management</td>
<td>Documenting requirements and changes, design and operational information throughout a system lifecycle, to check</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>b. Risk Management</td>
<td>Identification, assessment, prioritization, and strategies adopted to manage risks, and employ resources effectively during unexpected events.</td>
</tr>
<tr>
<td>f) Customer and User Expectations Management</td>
<td>Managing user expectation and issues faced due to changing customer expectations and demands.</td>
</tr>
<tr>
<td>g) Development Team Selection</td>
<td>Deals with the selection criteria and issues considered while selecting the development team.</td>
</tr>
<tr>
<td>h) Change Management</td>
<td>Managing changes at any part of the system lifecycle and issues faced due to changes.</td>
</tr>
</tbody>
</table>

3) **System Development Technical Activities**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Requirements Development, Allocation, and Management</td>
<td>Method adopted to deal with requirements, communication and allocation of requirements, and managing changing requirements. Also problems faced in these processes.</td>
</tr>
<tr>
<td>b) Design and Development (of architecture, software, etc.)</td>
<td>Deals with the development process used, design criteria, strategy and methods adopted, and issues dealt with during the process.</td>
</tr>
<tr>
<td>c) Testing/Evaluation</td>
<td>Method of testing employed, issues related with testing/evaluation.</td>
</tr>
<tr>
<td>d) Integration/Interfacing (of system components)</td>
<td>Issues related with integration of system components like hardware, software and related processes.</td>
</tr>
<tr>
<td>e) Deployment (i.e., Integration with operational environment and users; includes maintenance)</td>
<td>Issues which arise when the product is being delivered and deployed at the operational environment. Includes issues of maintenance, integration with the operational environment, and ease of usability by users.</td>
</tr>
<tr>
<td>f) Acquire Domain Knowledge</td>
<td>Acquiring adequate knowledge about the project and the product and a thorough understanding of the processes.</td>
</tr>
</tbody>
</table>
4) **External Pressures**

<table>
<thead>
<tr>
<th>a) Politics, Competition, and Conflicts external to System Development Team</th>
<th>Pressures and issues faced due to government Politics, dominance of a superior company, competition from smaller companies.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b) Systems Engineering Industry Trends and Practices (e.g., CMMI and Re-use)</td>
<td>Pressures and issues arising due to meeting with CMMI standards, present market demands, new trends and practices in systems engineering industry.</td>
</tr>
</tbody>
</table>