ABSTRACT

HARVIE, DAVID PAUL. Knowledge Sharing Mechanism (KSM):A Framework for Software Engineering and Command and Control. (Under the direction of Dr. Thomas L. Honeycutt).

The Knowledge Sharing Mechanism (KSM) is a framework to develop solutions to the complex problems faced in both software engineering and command and control. The environments of software engineering and military command and control systems are very similar because they are both instances of complex problem solving. The common nemesis to successfully developing solutions in these environments is change. Our understanding of the problem and the requirements needed to solve the problem as well as the problem environment itself undergo change. The challenge for any complex problem solving methodology is the balance of adapting to multiple changes while keeping focused on the overall desired solution.

The KSM is an iterative method for understanding a complex problem, developing a framework for solving that problem, creating, developing, and refining the parts of the solution for the problem, and then reassessing those partial solutions and overall framework until the complete solution has been fully developed. The KSM is based on the integration of Christopher Alexander's unfolding and differentiation processes with the image theory of command and control. In image theory, there are two perspectives in developing a solution. The first is topsight which is an overall general picture of the situation, and the second is insight which is a focused detailed view of a portion of the solution. Use of topsight and insight must be balanced in order to enable the solution's success. Alexander's unfolding process is the basis for understanding the complex interactions of both the software engineering and command and control environments. The KSM uses Alexander's differentiation process to achieve the correct balance of topsight and insight.

The KSM also uses the Knowledge Management discipline as another perspective in learning how to solve complex problems. The KSM uses the Knowledge Insight Model (KIM) in which there are four roles or patterns in Knowledge Management: the Framer, the Maker, the Finder, and the Sharer. The Framer is concerned with establishing the overall architecture for solving the problem, the Maker is responsible for developing innovative solutions for the problem, the Finder searches for resources to assist the Maker in developing

solutions, and the Sharer is responsible for managing the whole process by ensuring that the Framer, the Finder, and the Maker sharer their knowledge. Of the four roles, the Sharer is the most critical to the success of the solution. This knowledge sharing is the basis for the Knowledge Sharing Mechanism.

This paper will then analyze the KSM against evaluation criteria in both software engineering and command and control. The purpose is to demonstrate the validity of KSM as a framework to solving the complex problems in both environments. Finally, the paper will introduce ways that the KSM can be practically implemented in both software engineering and command and control. The KSM is a beneficial framework for an organization to develop software or manage their command and control systems because the KSM has the ability to ably respond to change while keeping the organization focused on achieving its desired goals

Knowledge Sharing Mechanism (KSM):A Framework for Software Engineering and Command and Control

by

David Paul Harvie

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2006

Approved By:

Dr. Matthias Stallmann	Dr. Laurie Williams

Dr. Thomas L. Honeycutt
Chair of Advisory Committee

To my loving and beautiful wife, Stephanie, and to our wonderful daughter, Paula Beth, for all your love, support, and patience.

To my dad and mom for raising my brother, Jonathan, and me in a Christ-honoring house and instilling the importance of character.

Ultimately, to the Lord Jesus Christ who has given me love, life, grace, and mercy that I could never repay.

"The fear of the LORD is the beginning of knowledge." Proverbs 1:7 (NIV)

Biography

David Paul Harvie was born in El Paso, Texas, on May 24, 1974. He attended Eastwood High School in El Paso (1988-1992) and the United States Military Academy at West Point, New York (1992-1996), where he received a Bachelor of Science in Computer Science and was commissioned a Second Lieutenant in the United States Army. After West Point, David successfully completed Airborne School at Fort Benning, Georgia, and graduated on the Commandant's List from the Field Artillery Officer Basic Course at Fort Sill, Oklahoma. He then completed Ranger School at Fort Benning en route to his assignment at Fort Bragg, North Carolina. In 1997, David was assigned to the 3-319th Airborne Field Artillery Regiment, 82nd Airborne Division. During his tour at Fort Bragg, he performed the duties of Company Fire Support Officer, Battery Fire Direction Officer, Battery Executive Officer, and Battalion Fire Direction Officer. David graduated from the Jumpmaster Course in 1998, and he deployed in 1999 to Kosovo for six months as part of Task Force Blue Devil (3-504 PIR). Shortly after returning from Kosovo, David returned to Fort Sill where he graduated as the Distinguished Honor Graduate of the Field Artillery Captains Career Course.

In 2001, was assigned to 1-39 Field Artillery (Multiple Launch Rocket System), 3rd Infantry Division at Fort Stewart, Georgia. David served as the Battalion Maintenance Officer and Battalion Assistant Operations Officer. On January 7, 2003, he deployed with the battalion to Kuwait as part of Operation ENDURING FREEDOM. This operation transitioned to Operation IRAQI FREEDOM on March 19, 2003, and 1-39 FA (MLRS) fired six missiles and over 600 rockets in combat operations from the Kuwaiti border to Baghdad. On May 1, 2003, David assumed command of Battery C, 1-39 FA (MLRS) at the

Baghdad International Airport. The battalion redeployed to Fort Stewart on June 3, 2003, and David continued his duties as Battery Commander until November 24, 2004.

Selected to teach Computer Science at West Point, David is an Active Duty Captain who began pursuing a Master of Science in Computer Science at North Carolina State University at Raleigh, North Carolina, in January 2005. His awards and decorations include the Bronze Star Medal, Meritorious Service Medal, Army Commendation Medal with Oak Leaf Cluster, Army Achievement Medal with 2 Oak Leaf Clusters, Kosovo Campaign Medal, Iraq Campaign Medal, Global War on Terrorism Expeditionary Medal, Global War on Terrorism Service Medal, National Defense Service Medal with Bronze Star Device, NATO Medal, Senior Parachutist Badge, Air Assault Badge, and Ranger Tab. David is married to the former Stephanie Eaton of Fayetteville, North Carolina, and they have one daughter, Paula Beth.

Acknowledgements

I wish to thank Dr. Thomas L. Honeycutt for his inspiration and guidance that sparked my interest in this subject in his Software Engineering class. I have learned so much from conversations with Dr. Honeycutt, both in and outside the classroom. Thank you for guiding me along the way. I also want to thank my committee members, Dr. Matthias Stallmann and Dr. Laurie Williams, for their insight and knowledge.

I wish to also thank David R. Wright for his knowledge and insight as often we discussed various topics with Dr. Honeycutt outside of Daniels Hall and Engineering Building II. I greatly appreciate David's expertise and assistance in building the LaTeXfiles necessary for this thesis.

Finally, and most importantly, I wish to thank my wife, Stephanie, and our daughter, Paula Beth, for being my biggest supporters. Your love, understanding, and acceptance have made this endeavor possible. You two are the biggest blessings in my life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Research Motivation

What is software engineering? One of the first usages of the term 'software engineering' occurred in 1968 during a North Atlantic Treaty Organization (NATO) Science Committee conference. The NATO Science Committee chose the term 'software engineering' as the title of their conference in order to provoke discussion on the need for software development and production to be based on theoretical and practical disciplines similar to other engineering fields. The NATO Science Committee recognized a growing crisis in the ability to develop and produce large scale software for an ever-increasing computerized society. There was an obvious need to formalize the software development process in order to manufacture good software that met the needs of the customer. This, however, would be easier said than done[25].

Why is software engineering hard? Software engineering is hard because it involves developing an abstract solution to a complex and ever-changing problem. Fred Brooks[13] stated that "the hardest part of building software is the specification, design, and testing of abstract concepts that are the essence of software." Compounding the difficulty in this problem solving endeavor is the inevitability of change. Brooks[12] also noted that "not only are changes in the objective inevitable, changes in the development strategy and technique are also inevitable." Thus, software engineering can be viewed as complex problem solving

on a problem that constantly changes.

By viewing problem solving as a higher order abstraction of software engineering, one can look to other fields that are derivatives of problem solving for possible insight. One such field is the military command and control systems. In fact, the problem solving dilemmas faced by software engineering are very similar to the decision making challenges faced by military command and control systems. The Army defines command and control as "the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of a mission. Commanders perform command and control functions through a command and control system." [18] The missions faced by commanders vary widely and greatly from combat operations, civil assistance, and nation building in Iraq and Afghanistan to Hurricane Katrina disaster relief in Louisiana and Mississippi. These are complex problems in which the problem environments and often the problems themselves change very rapidly.

1.2 Statement of the Problem

The common challenge to both software engineering and command and control is how to deal with change. One way to deal with change is to reject it. Once a plan has been made, stick to the plan. This has the benefit of adhering to an overall vision from start to end. Nonetheless, this approach is very prone to failure if the original plan did not take into account all potential problems. The second approach is to constantly react to change whenever it occurs. The most apparent benefit is the ability to be flexible and adjust to unforeseen changes and challenges. However, this approach can also fail if during all the adjustments to change, the overall goal of the initial plan is lost. Obviously, there must be some middle ground between these two extreme approaches that will allow us to adjust to change while simultaneously staying focused on the overall solution. The problem is to develop a methodology that is responsive to the inevitable changes while staying focused on achieving the goals of the overall solution.

1.3 Goals for This Thesis

The purpose of this paper is to introduce the Knowledge Sharing Mechanism (KSM) as a framework for adapting to change in both the software engineering and military command and control environments. First, the paper will demonstrate the need for the KSM in both software engineering and command and control. The characteristics of the desired KSM will be revealed using Christopher Alexander's Unfolding Process[4]. Then, the paper will describe the KSM in detail which uses as a foundation Alexander's Differentiation Process[5]. Finally, the paper will show how the KSM can and should be integrated in the software engineering and military command and control environments.

1.4 Objectives of This Thesis

The objectives for this paper are as follows:

1. Review relevant literature regarding the historical problems of dealing with change in the software engineering and command and control environments.
2. Demonstrate the need for and the characteristics of a knowledge sharing mechanism in both environments.
3. Introduce the Knowledge Sharing Mechanism (KSM) in detail.
4. Analyze the effectiveness of the KSM.
5. Demonstrate the integration of KSM in both the software engineering and command and control environments.

1.5 Thesis Layout

This paper will begin with historical perspectives of both software engineering and command and control. These two histories parallel each other in that hierarchical, specialized systems were developed to bring order out of chaos. However, these systems failed to adjust to the increasingly complex and uncertain environments of the software engineering and command and control worlds. The paper will then introduce Christopher Alexander's

unfolding process, a discussion about image theory, and Alexander's differentiation process as the foundations for the KSM. Then, the KSM will be introduced in detail. Finally, the paper will analyze the KSM using appropriate performance evaluations and propose suggested implementation of the KSM in both the software engineering and military command and control communities.

Chapter 2

Problems of Complexity in SE and C2

2.1 SE Environment

2.1.1 In The Beginning

As computing was in its infancy during the 1950's and early 1960's, the effort was almost entirely focused on hardware. Software was treated as an afterthought that was only necessary to make the hardware work. In fact, most software was coded, maintained, and fixed by one individual for a specific piece of hardware[29]. There was no underlying strategy or development concept that guided these early programmers. Robert Graham[25] compared the practices of software developers in 1968 to that of the Wright brothers trying to build a flying machine. Programmers would "build the whole thing, push it off the cliff, let it crash, and start all over again." Obviously, these haphazard and random means of developing software would soon prove inadequate.

It was in this state of affairs, that the North Atlantic Treaty Organization (NATO) Science Committee held an international conference in 1968 to address their growing concerns with software development. In this conference, NATO coined the phrase "software

engineering" in order to prescribe an engineering methodology and mind set to the world of software engineering[25]. This conference was important because the experts in the computer and software development fields recognized a growing crisis with developing future software reliably and within budget. However, they did not offer any solutions to this crisis.

2.1.2 Along Comes the Waterfall

The first software development model was developed by Winston Royce in 1970. He first detailed that the requirements for a piece of software must be methodically analyzed before actual code could be written. Royce further refined this process into seven steps. The process would begin with the system requirements which would translate to software requirements. These requirements would be analyzed, and from this analysis the program design would take shape. The program design would then be translated into code which would be tested until finally it was operational. These seven steps flowed down from top to bottom like a waterfall[30]. Thus, Royce's methodology of software development became forever known as the "Waterfall Model."

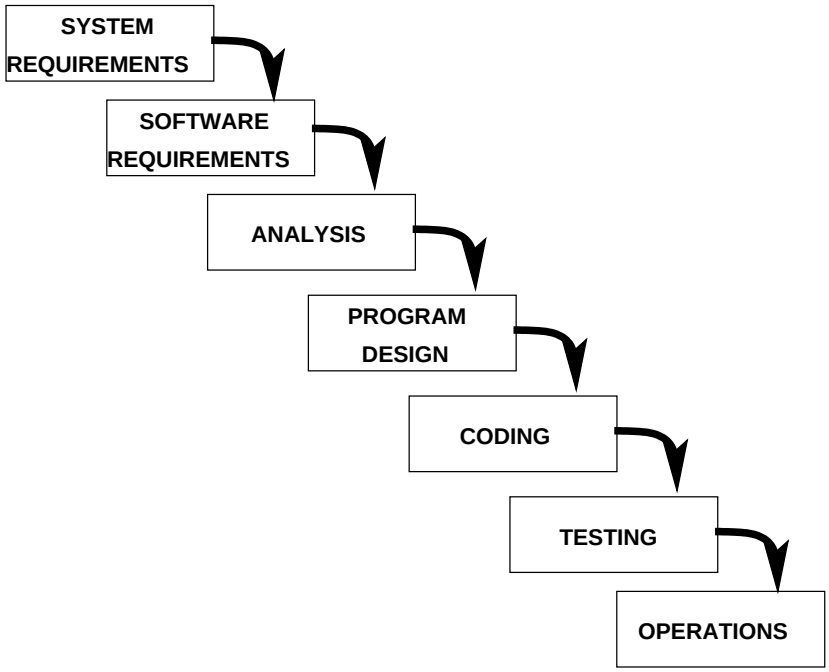

Figure 2.1: Royce's Waterfall Model[30]

The Waterfall Model was essentially a derivative of the traditional production method approach. The requirements for a product would be analyzed, translated into design, built along an assembly line, tested, and then marketed. Nevertheless, Royce in his seminal article recognized that discoveries made in successive steps in model may force developers to revisit previous steps in order to make changes to the design of the software. Hopefully, this revisiting of previous steps could be limited to immediately preceding steps. Royce also recommended that a first product, or prototype, software product should be developed in order to learn about an original product before developing the second product for the customer[30].

The fundamental flaw with the Waterfall Model is its inability to accommodate change. As early as 1975, it became obvious that certain changes in a software system goals and implementation were inevitable. Fred Brooks[12] noted that "not only are changes in the objective inevitable, changes in the development strategy and technique are also inevitable." In fact, he advocated that a software developer plan to throw away the first design or prototype because the software developer would have to anyway due to changing requirements and implementations. Thus, software developers continued to look for a better model.

2.1.3 The Spiral Model - It's All About the Risk

The Waterfall Model was the dominant software development model until the mid-1980's. Then, Barry Boehm, working at TRW Defense Systems Group, proposed an alternative, risk-based software development model known as the Spiral Model.

The Spiral Model is an iterative software development model consisting of multiple concentric cycles that spanned four quadrants. This was an evolutionary model that produced more complex prototypes upon the completion of each circuit of the four quadrants until the final product is delivered to the customer. In the first quadrant, the software developer in conjunction with the customer determined the objectives for the upcoming release. The software developer also determines alternative methods of accomplishing these objectives and what constraints are placed on those methods. In the second quadrant, the software developer evaluates the alternative methods by identifying and resolving their inherent risks. The developer then enters the third quadrant using the method with the least residual risk remaining and implements that method. At the conclusion of the third

quadrant, the software developer conducts a review with the customer to determine how well that release accomplished its stated objectives. The fourth quadrant is then used to plan for the next iteration and release (unless you have just delivered the final release)[11]. A later version of the Spiral Model added three more activities (or sectors) to the beginning of each cycle. Those three activities were identifying the key stakeholders in the customer organization, identifying those stakeholders win conditions for the system, and then reconciling any conflicting win conditions among the stakeholders that results in a final set of negotiated win conditions before identifying objectives[9].

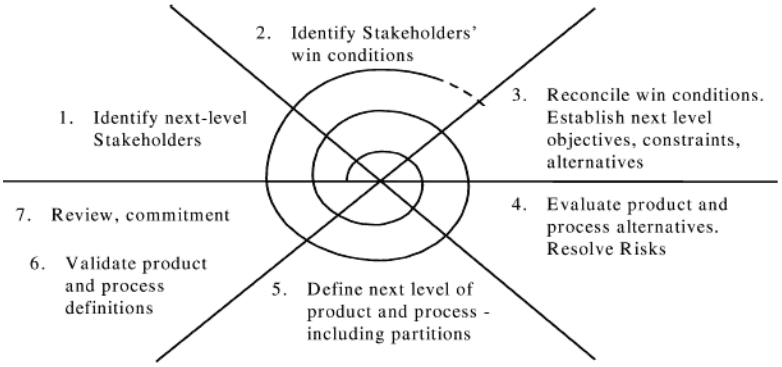

Figure 2.2: Boehm's Spiral Model [9]

The weakness of this model is the ability of the software developer to accurately identify, assess, and mitigate risks during each iteration. This ability to manage risk is entirely hinged upon the developer's knowledge of the product and the environment into which it is being developed for. It is apparent that such complete knowledge will not be available at the onset of the project. There will be new knowledge gained not only by the software developer, but by the customer during the course of the project.

2.1.4 The Agile Revolution

Many in the software engineering community saw the current software development methodologies as too rigid and incapable to deal with the inevitable changes associated with producing software. These people began to develop what were later called Agile Methods. The motivation behind the Agile Methods is to embrace the inevitability of change and to

then adeptly use the change to produce successful software[7]. Two of the most dominant of the Agile Methods are eXtreme Programming (XP) and Scrum.

XP was developed by Kent Beck in the late 1990s. The classic model of the cost of implementing changes is that it follows an exponential path. A change implemented later in software development will cost much (perhaps exponentially) more in resources than if the change was implemented sooner. The driving principle behind XP is to flatten the cost of change so that a change implemented late in the development of the software system is little more or even the same as if the change was implemented at the very beginning in of the development cycle[7].

XP seeks to achieve this flattened cost of change curve by adhering to five fundamental principles. The first principle is rapid feedback. In the ideal XP environment, a customer representative is positioned on-site with the XP development team in order to answer questions and give immediate feedback on the progress of the system. The second principle is to assume simplicity. XP developers solve the current problem as simply as possible instead of devising a much more complex solution to handle potential future problems. The third principle is incremental change. The idea is that any problem can be solved in a series of little steps as opposed to trying one giant step. The fourth principle is to embrace change since it is inevitable. The fifth and final principle is to do quality work. Developed code is evaluated to see if there is a better or more efficient way to implement the same solution. By achieving the simplest solution possible, the developer gives himself or herself more flexibility to add to that code or change it[7].

Another agile software development methodology is Scrum which was developed by Ken Schwaber in the early 2000's. Scrum attempts to control the software development process using an empirical approach. This empirical approach expects the unexpected and uses frequent inspection and adaptation in order to respond to the uncertainties that develop. This is very similar to the XP approach except that Scrum deals more with the managing of the software development process whereas XP deals more with the mechanisms of coding and testing the software[32].

The Scrum process begins with the Product Owner developing and prioritizing the Product Backlog for the desire software product. The Product Backlog is a prioritized list of all the product requirements. The Scrum Team, which is a self-organizing and autonomous group of about seven programmers led by a Scrum Master, takes as many requirements from the Product Backlog that can be met in 30 days and creates a Sprint Backlog. The

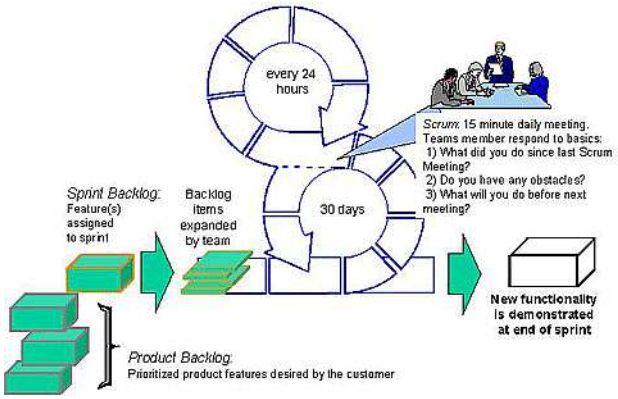

Figure 2.3: Scrum Lifecycle [15]

Scrum Team then has a 30 day Sprint in which to write and test the code to fulfill all the requirements outlined in the current Sprint Backlog. During this 30 day Sprint, the Scrum Master facilitates a daily 15 minute meetings called the Daily Scrum. At the Daily Scrum, each member of the Scrum team brief each other on what he or she has done since the last Scrum meeting, what he or she plans to do between now and the next Scrum meeting, and what issues he or she encountered that prevented him or her from accomplishing the desired goals. A key concept in Scrum is that the Scrum team has autonomy and be left alone during this Sprint process. At the end of the Sprint, the Product Owner and the Scrum team conduct a Sprint Review to cover how the Scrum Team fulfilled the requirements outlined in the Sprint Backlog and to help develop the next Sprint Backlog[32].

2.1.5 Attempting to Balance Agile and Plan-Driven Methods

Some have attempted to create a hybrid of the agile and plan-driven software development models. One such hybrid is the model proposed by Barry Boehm and Richard Turner. Three of their top six conclusions are that:

1. Agile and plan-driven methods have home grounds where one clearly dominates the other.

2. Future trends are towards application developments that need both agility and discipline.

3. Some balanced methods are emerging.

Boehm and Turner thus attempt to create a balance model using a five axis polar plot using risk analysis as the basis of determining the mix of agile and plan-driven methodologies to use given a particular software project[10].

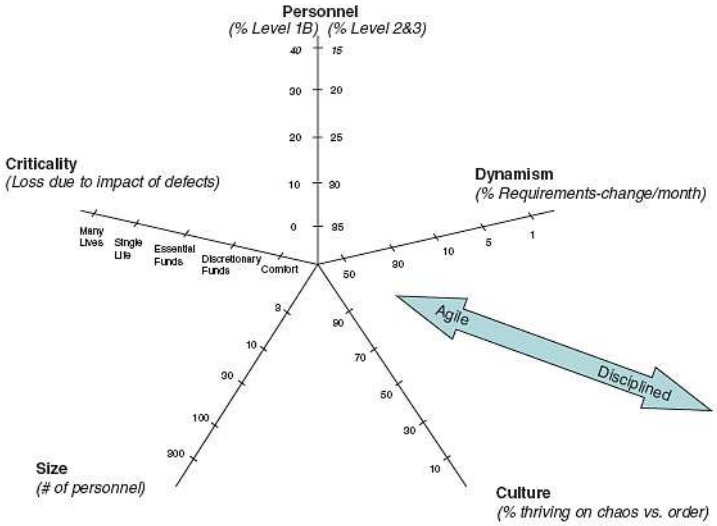

Figure 2.4: Boehm-Turner's Model for Balancing Agile and Plan-Driven Methodologies[10]

2.2 C2 Environment

2.2.1 What is Command and Control

As previously mentioned, the Army[18] defines command and control (C2) as "the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of a mission. Commanders perform command and control functions through a command and control system." Similarly, the Marine Corps[34] defines C2 as "the means by which a commander recognizes what needs to be done and sees to it that appropriate actions are taken." In other words, C2 is a systemic framework by which an organization, such as the military, identifies problems, develop solutions to those problems, and then implement those solutions. It is important to note that while the remaining discussion with regards to C2 will have a military flavor, C2 is not limited to

the armed forces. Businesses, organizations, sports teams, and other groups of people that strive to achieve goals by working together use some form of C2. Otherwise, only individual accomplishments would be possible.

2.2.2 The Rise of Industrial Age C2

Command and control has existed in military forces since the earliest recordings of history. However, command and control used to be referred as only command with the obvious emphasis on the ability of the military commander. Prior to the advent of the Industrial Revolution in the 1800's, the predominant method of C2 was detailed command. The commander would position himself in a strategic location where he could see the entire battlefield, the majority of his forces, and the most enemy forces as possible. The commander would then formulate a battle plan to defeat the enemy and send instructions to his subordinate commanders on what their assigned tasks were. This centralized problem solving approach to C2 generally worked because the battlefield and the military forces involved were small enough to be effectively seen by one person. In addition, most battles lasted only one day[18].

Detailed command is a plan-driven approach that stems from the belief that success on the battlefield comes from imposing order and certainty. The commander gave detailed and explicit orders to his subordinates who were expected to carry them out with strict obedience. In this model, information flows up from subordinate to superior levels all the way to the commander. The commander develops a grand plan and then transmits the necessary orders to accomplish that plan back down to his subordinates to carry out.[18] The commander was the pivotal, and many times the only, decision-maker on the battlefield. Success or failure rested with his ability to comprehend the situation and develop an effective battle plan.

As armies and battlefields grew larger and more complex, it became impossible for one commander to be able to visualize the entire battlefield and thereby develop an effective battle plan. During the Industrial Age, military theorists began to see armies as simply machines of the nation-state. Armies could then be optimized for success on the battlefield based on elaborate mobilization and deployment algorithms and schedules, similar to an industrial production plant[27]. The military, as a result, began to adopt some Industrial Age production principles. These principles were decomposition, specialization, hierarchical

organizations, centralized planning, decentralized execution, and optimization[2].

Decomposition is taking a "divide and conquer" approach to a problem. A military staff responsible for assisting the commander in developing the battle plan would be decomposed into various functional areas. Different staff officers were assigned to analyzing intelligence, keeping track of personnel data, monitoring logistics, developing a scheme of maneuvering units, and developing a plan of using artillery and mortar fires. This decomposition into specific functional areas led to the second principle, specialization. Since these staff officers and their soldiers were assigned to perform a specific task, such as monitoring logistics, they became specialists in their fields. We still see this specialization today as soldiers are assigned to different functional organizations called branches (e.g. Infantry, Field Artillery, and Transportation)[2].

Militaries are by their nature hierarchical organizations. The importance of hierarchy during the Industrial Age deals with span of control, which is the amount of people that one person can adequately supervise. A general rule of thumb is that one person can have a span of control between three and seven people. If there are only seven people in an organization then it is no problem for the leader to supervise. However, if the organization has more people, for example 20 people, then the leader can divide the organization into four groups of five people each and appoint a leader within each of those groups. Now the organization leader only has to supervise four group leaders who each supervise four people under them. Thus, you begin to grow a bureaucracy in the organization, civilian and military[2].

Hierarchical organizations by nature gravitate toward centralized planning and decentralized execution. Sensory information flows upward from the worker at the lowest rung up through a bureaucracy of middle management to the leader (or his staff) who analyze it and develop a plan based on that information. The final plan is then transmitted back down through the middle management until it is disseminated to the lowest level.[34] One key weakness in this approach is that there is little to no cross communication between elements in the organization if they do not share a common hierarchy. The other key weakness is the assumption that the higher up a person is in the organization, the more informed and the better able he or she is to develop an appropriate solution given the current situation. Unfortunately, many people have learned that "higher does not mean smarter[8]."

The purpose behind decomposition, specialization, hierarchical organizations, cen-

tralized planning, and decentralized execution was optimization. Military planners sought to deal with the complexity of warfare by iteratively dividing it down into multiple small problems that could be easily solved. They believed that not only did solutions exist to these small problems; there existed an optimal solution for each problem. The thought was the optimal solutions of these small problems would then result in an aggregate optimal solution for the large scale problem (i.e. the battle at hand)[2].

2.2.3 The Failure of Industrial Age C2

Centralized planning has two nemeses: change and uncertainty. In the late 1800's, Helmuth von Moltke, Chief of the Prussian General Staff recognized the inability of the C2 philosophy developed during the Industrial Age to handle change. Von Moltke stated that "no plan of operations extends with any degree of certainty beyond the first encounter with the enemy main force." As a result, von Moltke developed a concept called *Auftragstaktik* (which is translated "mission tactics"). *Auftragstaktik* emphasized decentralized initiative within the overall strategic design[18]. This was the start of the move away from Industrial Age C2 to Information Age C2.

The German Army, which directly descended from the Prussian Army, did immediately heed von Moltke's warning. The German General Staff, under the leadership of Alfred von Schlieffen, developed a detailed battle plan to wage a two-pronged war against France and Russia. This plan, later named the Schlieffen plan, became the blueprint for Germany's actions during World War I. Likewise, the Schlieffen plan is one of the clearest examples of the failure of strictly adhering to a plan-driven approach. The Schlieffen plan called for the mobilization and deployment of German forces to attack France first. The Germans planned to conduct a flanking attack through neutral Belgium. The German General Staff made the critical assumption that it would take six weeks to defeat France. This was pivotal because those German forces attacking France would need to be quickly deployed to the east to take on the Russians. The General Staff assumed that it would take Russia much longer than France to mobilize its forces, and the General Staff planned to quickly defeat France first in order to prevent fighting a war against France and Russia simultaneously on two fronts[27]. The General Staff calculation of victory in France in six weeks did not materialize during World War I, and the German Army had to fight the entire war on two fronts.

The inability of the German Army's Schlieffen plan to deal with change (i.e. it would take longer than six weeks to defeat France) clearly demonstrated the failure of Industrial Age C2. The goal of Industrial Age C2 was to discover the globally optimal plan to achieve victory through decomposition and specialization. This focus on optimization created two major weaknesses: lack of communication and lack of agility[2].

The first major weakness was the lack of communication. The hierarchical structures of Industrial Age militaries created "stovepipes" of information. Information acquired at the lowest level would travel up a specific chain-of-command until it reached its destination at the highest level of the organization. Any individuals not in this particular chain would not be privy to that information. Thus, stovepipes of information transported information vertically between higher and lower levels of the organization, but never horizontally between adjacent levels of the organization. David Alberts[2] states that "Industrial Age organizations create fixed seams through which information is lost." This placed an enormous amount of reliance on the centralized planning of the highest level in the organization because this was the only entity in the organization privy to all the information being transmitted.

The second major weakness was the lack of agility. In order to achieve the optimal solution, there have to be tradeoffs. The Industrial Age military chose the "best" solution that presented the best results only if very specific conditions were met over a "good" solution that presented slightly less, but still acceptable, results given a much wider range of conditions. The "best" solution worked tremendously if all the conditions could be met, however the "best" solution also failed miserably if one just of those conditions could not be satisfied. This "best" solution was rigid and could not successfully adapt to change[2].

This lack of communication and agility meant that the Industrial Age C2 model could neither handle complex situations nor adapt to changes. This led the military to look for a new paradigm for command and control.

2.2.4 C2 in the Information Age

Today's military environment is far more complex and ever-changing than in von Moltke's era. General Charles Krulak, former Commandant of the United States Marine Corps, best summarized today's complex military environment when he stated in 1999 that,

> in one moment in time, our service members will be feeding and clothing dis-

> placed refugees - providing humanitarian assistance. In the next moment, they will be holding two warring tribes apart - conducting peacekeeping operations. Finally, they will be fighting a highly lethal mid-intensity battle. All on the same day, all within three city blocks. It will be what we call the three block war[23].

In 2006, we clearly see our military operating in this environment in Iraq and Afghanistan in addition to disaster relief missions following Hurricanes Katrina and Rita.

Command and control in the Information Age must be able to quickly respond to fluid, rapidly changing military situations. This means that the C2 system must be able to rapidly obtain and assess information about the environment and then quickly determine and communicate the appropriate response. This C2 system must accomplish this while dealing with some level of uncertainty and to fulfill the mission under time constraints[34]. These demands of the Information Age C2 model can be articulated in the following essential capabilities[2]:

1. The ability to make sense of the situation.
2. The ability to work in a coalition environment including nonmilitary partners.
3. The ability to orchestrate the means to respond in a time manner

While it is clear what tasks a successful command and control system in the Information Age must accomplish, there is currently not an obvious scheme on how to accomplish those tasks.

2.3 Agility is the Winner

The histories and experiences of software engineering and military command and control bear an uncanny resemblance of one another. The genesis of software engineering and C2 came from the desire to bring some type of order to a previously chaotic process. This was the rise of the plan-driven approaches to both areas. In the beginning, this plan-driven approach had a measure of success because the programs and battles were small enough and simple enough for one person to visualize and plan. Nonetheless, both the software environment and the military environment grew too large in scale and complexity for one person to visualize and plan, which led to the birth of organizations and bureaucracies

whose goal was to gain control of these increasingly difficult processes. These plan-driven organizations likewise failed because they could not handle uncertainty and change.

The desired goals of both software engineering and C2 are the abilities to deal with uncertainty and change. Both the software engineering and the military C2 communities have independently concluded that the key to dealing with uncertainty and change is through agility. The software engineering community codified this importance of "responding to change over following a plan" in the Agile Manifesto which was signed by numerous developers and practitioners of agile software development methods[1].

Likewise, the United States Department of Defense has recognized the imperative for it to be more agile. Secretary of Defense Donald Rumsfeld expressed this need for change in a speech at the Pentagon. Rumsfeld[31] said,

> The topic today is an adversary that poses a threat, a serious threat, to the security of the United States of America. This adversary is one of the world's last bastions of central planning. It governs by dictating five-year plans. From a single capital, it attempts to impose its demands across time zones, continents, oceans and beyond. With brutal consistency, it stifles free thought and crushes new ideas. It disrupts the defense of the United States and places the lives of men and women in uniform at risk. ... The adversary's closer to home. It's the Pentagon bureaucracy. Not the people, but the processes. Not the civilians, but the systems. Not the men and women in uniform, but the uniformity of thought and action that we too often impose on them. ... It demands agility – more than today's bureaucracy allows. And that means we must recognize another transformation: the revolution in management, technology and business practices. Successful modern businesses are leaner and less hierarchical than ever before. They reward innovation and they share information. They have to be nimble in the face of rapid change or they die.

Ironically, these words were uttered the day before the events of September 11, 2001. The United States Department of Defense has now formalized this need for agility in the 2006 Quadrennial Defense Review[16] when it stated "given the dynamics of change over time, we must develop a mix of agile and flexible capabilities to mitigate uncertainty."

Chapter 3

Complexity and the Unfolding Process

3.1 The Universe of Centers

Software engineering and military command and control are not the only disciplines that deal with complex problem solving. Structural architecture is another field that also seeks to create a solution to a complex and changing situation. A pivotal architect of our time is Christopher Alexander. Alexander is not a stranger to the software engineering community. In fact, he gave the keynote address to the 1996 ACM Conference on Object-Oriented Programs, Systems, Languages, and Applications (OOPSLA). Alexander tied in the similarities of architecture and software engineering by identifying both as creative disciplines. Alexander[3] said to the OOPSLA attendees that "the idea of generative process is natural to you. It forms the core of the computer science field."

Alexander[6] states that, "architecture presents a new kind of insight into complexity because it is one of human endeavors where we most explicitly deal with complexity and have to create it." Alexander's comments are based upon 35 years of experience in architecture that culminated in his *Nature of Order* series. The *Nature of Order* deals with understanding the characteristics of good designs, natural and man-made, with the intent

of using this knowledge to produce future good designs. It is therefore insightful to apply this discovery about design to both the software engineering and military command and control environments since they, like architecture, are creative endeavors of complexity.

A building that has "life", according to Alexander, must have a high degree of "wholeness." The building is not an isolated structure that exists; it is part of a larger world consisting of trees, grass, streets, and other buildings that are part of the building's environment. Within the building itself, there is a sense of wholeness consisting of the shape of the walls, the windows, and the doors. To the degree that there is a harmonious blend of these entities, there is the corresponding amount of wholeness. Alexander[4] states that "*wholeness* is the important thing: the local parts exist chiefly in relation to the whole, and their behavior and character and structure are determined by the larger whole in which they exist and which they create."

In defining "wholeness" we do so in terms of entities that contribute to the wholeness of a design or a structure. These entities are called "centers." The defining mark of each center is that each "*appears to exist a local center within a larger whole.*" Alexander further refines his definition of a center as not a specific point, but "*a physical set, a distinct physical system which occupies a certain volume in space, and has a special marked coherence.*" These centers create the wholeness of a design, and paradoxically the wholeness of the design creates these centers[4]. Thus, understanding the power and interaction of centers to create wholeness is key to good design. Even a whole design can be thought of in the context as just a center of an even larger scale design.

There is a clear benefit in viewing entities as centers because it gives us greater insight into the true nature of those entities. It is also less problematic to view an entity as a center rather than a whole. Alexander uses the example of describing a fishpond as a whole or as a center. If we describe the pond as a whole then we have the unenviable and very difficult task of establishing distinct boundaries that encapsulate all the components of a pond and exclude those that are not components of the pond. For example, do we include the air above the water or the ground below the water as part of the pond? Both of the air and the ground can be considered critical components of the pond. If we then include them, how much air and ground should we encapsulate in our boundary? A better way is to view the pond as a center. This perspective recognizes the "existence of the pond as a coherent entity" while allowing it to have "fuzzy" boundaries. We can now deal with the existence and the effect of the pond instead of trying to establish artificial boundaries[4].

The final benefit of viewing an entity as center rather than a whole is the perspective that no one thing exists in isolation. We must understand not only the entity but its interactions with its environment and vice versa.

3.2 The Nature of Centers

With the existence of centers firmly established, it becomes important to understand the nature of centers. Alexander[4] proposes the following four ideas regarding centers. It is important to note that according to Alexander the better a design or structure is, the more that it has "life."

1. Centers themselves have life.
2. Centers help one another; the existence and life of one center can intensify the life of another.
3. Centers are made of centers (this is the only way of describing their composition).
4. A structure gets its life according to the density and intensity of centers which have been formed in it.

There are two critical characteristics of centers found in these ideas. First, is the idea that "Centers are made of centers." This gives the center a recursive characteristic in which we can further identify supporting centers. Alexander uses the example of a tree as a center. A tree consists of a trunk, branches, and leaves, each of which can be viewed as centers themselves. You could then view an individual leaf as a center and identify its supporting centers such as the stem and the minor ribs of the leaf. The recursive nature of centers is not only in the decomposition of an entity but in its composition. When we see the centers of a trunk, branches, and leaves positioned in a particular way, it creates the center that we call a tree[4].

The second critical characteristic comes from the ideas that "Centers help one another; the existence and life of one center can intensify the life of another," and "A structure gets its life according to the density and intensity of centers which have been formed in it." These ideas give centers the characteristic of a field-like structure. There is a force emanating from the center toward other centers that help strengthen those centers. Likewise, there is a corresponding force pulling from other centers towards the one center

to help strengthen it[4]. Similar to gravitational forces, each center exerts an influence on other centers and itself is influenced by other centers.

3.3 The Fifteen Properties

What makes a design a good design? Through his years of studying various designs, both natural and man-made, from the ancient era through the modern era, Alexander has recorded 15 properties present in good designs. These 15 properties are listed in Figure 3.1.

1. Levels of Scale	9. Contrast
2. Strong Centers	10. Gradients
3. Boundaries	11. Roughness
4. Alternating Repetition	12. Echoes
5. Positive Space	13. The Void
6. Good Shape	14. Simplicity and Inner Calm
7. Local Symmetries	15. Not-Separateness
8. Deep Interlock and Ambiguity	

Figure 3.1: Alexander's 15 Properties[4]

These properties of good design are not independent, discrete characteristics. Rather, these properties are interwoven[4]. For example, Alternating Repetition and Gradients work together to define Strong Centers. It is important not only to understand the 15 properties individually, but also how these properties interrelate.

According to Alexander, the reason that these 15 properties are present in good design is because these are the 15 principal ways in which centers can be strengthened by other centers. Since we can view complex environments as a system of centers, we can use our understanding of these 15 properties to understand the interaction of these centers. More importantly, we can use the understanding of these interactions in order to improve (or strengthen) these centers in order to improve the system as a whole. The following are the interactions of the 15 properties with regards to centers:

1. Levels of Scale – "the way that a strong center is made stronger partly by smaller centers contained in it, and partly by its larger strong centers which contain it."

2. Strong Centers – "requires a special field-like effect, created by other centers, as the primary source of its strength."
3. Boundaries – "strengthens the field-like effect of a center by the creation of a ring-like center, made of smaller centers which surround and intensify the first."
4. Alternating Repetition – "centers are strengthened when they repeat, by the insertion of other centers between the repeating ones."
5. Positive Space – "a given center must draw its strength, in part, from the strength of centers immediately adjacent to in in space."
6. Good Shape – "the strength of a given center depends on its actual shape, and the way this effect requires that even the shape, its boundary, and the space around it are made up of strong centers."
7. Local Symmetries – "the intensity of a given center is increased by the extent to which other smaller centers which it contains are arranged in locally symmetrical groups."
8. Deep Interlock and Ambiguity – "the intensity of a given center can be increased when it is attached to nearby centers through a third set of centers that ambiguously belong to both."
9. Contrast – "a center is strengthened by the sharpness of a distinction between its character and the character of surrounding centers."
10. Gradients – "a center is strengthened by a graded series of different-sized centers which then point to the new center and intensify its field effect."
11. Roughness – "the field effect of a given center draws its strength, necessarily, from irregularities in the sizes, shapes, and arrangements of other nearby centers."
12. Echoes – "the strength of a given center depends on similarities of angle and orientation and systems of centers forming characteristic angles thus forming larger centers, among the centers it contains."
13. The Void – "the intensity of every center depends on the existence of a still place - an empty center - somewhere in its field."
14. Simplicity and Inner Calm – "the strength of a center depends on its simplicity - on the process of reducing the number of centers which exist in it, while increasing the strength of these centers to weigh more."
15. Not-Separateness – "the life and strength of a center depends on the extent to which that center is merged smoothly - sometimes even indistinguishably - with the centers that form its surroundings."[4]

3.4 Unfolding

The 15 properties occur not only in human designed structures, but more importantly in nature. One can see the importance of Boundaries in the structure of a plant cell, and the relevance of local symmetries found in the wings of a bee. There are so many more examples of the 15 properties in nature, that Alexander devotes much of The Phenomenon of Life to categorizing and analyzing the emergence of these properties[4].

The 15 properties however represent much more than a description of a design. They are also the means by which one design evolves, or unfolds, into a stronger design. We see this especially in nature. An example that Alexander gives is in the development of an embryonic mouse forelimb. Eleven days after conception, the mouse's forelimb is nothing more than a circular blob of cells. Within the next four days, however, that circular blob of cells will begin to differentiate in both shape and substance. At the end of 15 days, the forelimb and foot of the mouse have evolved to the point that the individual digits, forelimb bones, and joints are clearly present. This radical transformation does not happen instantaneously. The unfolding of the mouse forelimb from cell blob to well-defined structure occurs as a step-by-step transformation of the previous structure. We see the development of Strong Centers as the forelimb bones and digits begin to emerge. We also see Alternating Repetition in the space between the digits. In this unfolding process, we see one or more of the 15 properties being used as structure-preserving transformation. The reason that this is structure-preserving is that at no time in the evolution of the forelimb is there only part of a forelimb. The forelimb exists as a whole at day 11 just as it does on day 15. The difference between these two days is that the whole forelimb of day 11 has undergone numerous transformations that result in a much stronger and more clearly defined whole forelimb of day 15[5].

In observing this unfolding process in nature, we see a step-by-step change in the whole of a design. The vehicle for this change is the employment of one or more of these 15 properties as a structure-preserving transformation. It is from this insight into nature's unfolding process, that Alexander will later develop his method for replicating this process.

Chapter 4

A Tale of Two Images

4.1 The Value of Images

Images provide an invaluable means to communicate. It is often said that "a picture is worth a thousand words." People normally do not think in terms of data or words. They naturally think in terms of mental pictures. Also, people are more adept at assimilating information in the form of a picture as opposed to information in the form of text or numbers. Images can be created for the purpose of more accurately describing a problem or situation. Also, images can be developed to communicate the solution to the problem[34].

Images, obviously, are a powerful tool for communicating thoughts, conditions, and ideas. In fact the U.S. Army *Operations* field manual[17] clearly dictates that commanders, assisted by their staffs, must first visualize the situation before determining and directing a course of action. It is in creating this mental image (or visualizing) that a commander can more adeptly understand his or her environment and the dynamics of that environment. Words and data alone can not create such a rich context as an image can.

There is a methodology to visualizing. Commanders, with the support and input of their staffs, must focus on the following three factors when visualizing[18]:

- Foreseeing an end state.
- Understanding the current state of friendly and enemy forces.

- Visualizing the dynamics of operations leading to the end state.

In other words, the commander assesses the unit's current situation, envisions a desired goal for the unit to obtain, and then formulates a way to move from the current situation to the desired goal. While this may seem rather simple, it is in fact a profound cycle that does not end until the commander and his/her unit achieve the desired end state. This visualization must be continuously updated and validated because the current states of friendly and enemy forces are dynamic, not static. As a result, the dynamics of operations leading to the end state must also be fluid enough to respond to changing friendly and enemy situations. Finally, it is very possible that the desired end state may also change based upon the changing environment.

The value of images is not limited to the battlefield. Software developers also attempt to harness the power of an image. One of the 12 practices in XP is Metaphor. In this case, the XP team is using a metaphor, or a word picture, in order to guide the overall project. The metaphor becomes a tool that communicates the basic elements of the project. The reasoning behind developing and using a metaphor is to create a simple design architecture that is easy to communicate and elaborate[7]. Even though a metaphor may not be a physical drawing, it still conveys a mental image. In fact, Christopher Alexander[5] contends that "word pictures are better than actual pictures to give a sense of the whole because actual pictures contain too much (and usually arbitrary) information."

4.2 Topsight Versus Insight

Military commanders must visualize the situation using three different perspectives[18]. Those three perspectives are:

- A close-up view of the situation gained through personal observation and experience.
- An overview of the situation and overall progress of the operation.
- A view of the situation from the enemy's perspective.

In a general problem-solving context, we use just the first two perspectives for our visualization.

The first perspective is "insight." This insight provides the most detailed picture of a situation. The individual using insight selects a part of the situation and focuses

on the specific part. Then, the individual is able to exploit his or her own observations, experiences, and tacit knowledge to gain understanding as to what progress or action is taking place. It is often said that the person using insight gets a "feel" for the situation since he or she is relying so heavily on his or her sensory and thinking abilities. Insight is similar to using a magnifying glass on a picture. The benefit of the magnification is the ability to see the details of the picture more clearly and perhaps understand how they work together. Magnification, though, comes at a price. Insight, by its definition, can only look at a portion of the whole picture. The viewer is restricted to a narrow scope, and similar to the laws of optics, the greater the level of magnification the smaller the field of view. Thus, one who uses too much insight is at risk of losing sight of the big picture[34].

The second perspective is "topsight." Topsight provides a view of the entire situation similar to a bird's eye view of a piece of land. The individual using topsight looks at the entire situation and tries to understand the overall unfolding situation. The viewer sees what the current state of the environment is and compares it to the desired end-state. Using topsight, one can constantly comprehend the entire situation. However, there is a danger too with relying just on topsight. In order to see the entire picture, the viewer must settle on the perspective with the least detail. The patterns we see at this large-scale level may give a false impression of the true situation. Critical details to the success of the situation may fall well below the resolution of the overall picture. Thus, one who uses too much topsight is at risk of losing touch with reality[34].

The concepts of insight and topsight can be found in software engineering too, though they are not specifically mentioned as such. In Scrum, the team works with a Product Backlog and multiple Sprint Backlogs. The Product Backlog is a prioritized list of features, functions, technologies, and enhancements that are to be created during the entire project. While the Product Backlog is not a true image in the sense of a picture or metaphor, it does convey the sense of an image. The Product Backlog is an attempt to capture an overview of the entire project. In that sense, the Product Backlog is akin to topsight. The Sprint Backlog is a specifically chosen subset of the Product Backlog. The Scrum team in conjunction with the Product Owner determines which features, functions, technologies, and enhancements from the Product Backlog can be implemented in the upcoming 30 day Sprint. The Sprint Backlog, like the Product Backlog, is not a true image. Nonetheless, the Sprint Backlog provides a detailed, magnified view of a portion of the entire project. The Spring Backlog is therefore similar to insight[32].

Scrum is not unique in conveying the ideas of insight and topsight; XP also suggests these two perspectives in the practice of Pair Programming. Pair Programming is defined as "a style of programming in which two programmers work side by side at one computer, continually collaborating on the same design, algorithm, code, or test." One programmer is called the driver. The driver is responsible for physically typing in the code or drawing the design. The other programmer is called the navigator, and the navigator is responsible for observing the work of the driver, looking for tactical or strategic defects. A tactical defect could be a typo, invoking the wrong procedure, or syntax error. On the other hand, a strategic defect is when the driver begins to deviate away from the system's design[35]. The driver is entirely focused on the tactical design of the code. Once again, this is a detailed, magnified view of a portion of the entire project which relates to insight. The navigator has to maintain a strategic perspective with the motivation of keeping the driver from heading down the wrong path. In order to have this strategic perspective, the navigator must be able to have the topsight necessary to view the project as whole. Only then can the navigator ensure that the driver is following the right path.

4.3 Achieving Balance Through Differentiation

In both command and control systems and software development methodologies, there definitely exist the concepts of insight and topsight. One must use both perspectives in order to evaluate and determine solutions to a particular situation. However, it is impossible to simultaneously view a situation from the tactical and strategic perspectives. As previously cited, a person who spends too much time using insight runs the risk of being so detailed and action oriented that he or she loses sight of the grand design. Likewise, one who spends too much time assessing the overall scheme can easily overlook critical details that will have a tremendous impact on the entire situation. Thus, the problem is the balance between these two perspectives.

In Christopher Alexander's unfolding process, two concepts are at work in any structure: wholeness and centers. The wholeness and strength of a structure are dependent upon the strength and interaction of the centers that are present in the structure. The wholeness of the structure can be viewed as a topsight perspective, and the view of a specific center within that structure is equivalent to an insight perspective. The interaction

of these centers with the wholeness of a structure led Alexander to discover the 15 properties and the unfolding process in nature.

Alexander did not stop with observation of the unfolding process in nature. He wanted to replicate it. This led him to develop the differentiation process by which a structure or design is consciously strengthened and improved. The differentiation process uses knowledge of the 15 properties of the unfolding process to derive 15 structure-preserving transformations. Figure 4.1 outlines this differentiation process.

This differentiation process gives us a framework for balancing the topsight and insight perspectives of a given design or structure. Differentiation begins with the topsight view in steps 1 through 4 by recognizing the complex interaction of the centers that comprise the whole and analyzing the strength of the whole in terms of these centers. The purpose is to identify a latent (or weak) center that needs to be strengthened in order to strengthen the entire design. Then, there is a transition to the insight perspective in steps 5 through 9 as the differentiation process focuses on the chosen latent center and performs structure-preserving transformations to strengthen that latent center. Finally, the process returns to the topsight view in steps 10 and 11 to verify that the strengthening of the center is the simplest possible and reexamines the whole structure now that this once latent center has been strengthened. It is important to note that this differentiation process is a continuous cycle that switches back and forth between the topsight and insight perspectives.

The differentiation process does not exist in a vacuum. There are four necessary conditions that must be present in order to successfully differentiate a design. Alexander[5] identifies those four conditions as:

- Awareness of the whole
- Step by step adaptation
- Unpredictability
- Feedback

Awareness of the whole is best described in step 1 of the differentiation process. It involves understanding the entire design in terms of a system of partially evolved centers that are nested relative to one other. Without awareness of the whole, the differentiation process can easily veer off course from the desired end state. Step by step adaptation is a

gradual, continuous process that allows assessments, corrections, and improvements. This evolutionary process is necessary because the interactions and relationships of the centers in a structure are too complex to be understood and changed simultaneously. The step by step adaptation can be seen in the development of the embryonic mouse forelimb from a clump of cells to a clearly differentiated structure. The process obviously followed a step by step transformation as the cells took shape; the change did not happen instantaneously. Unpredictability recognizes that the differentiation process must be open-ended. Changes will occur, and there is a clear danger in fixing a design too early in the process before recognizing and dealing with those changes. Finally, feedback is necessary throughout the design process. Feedback enables us to check and correct, if necessary, our progress. Feedback only after the design process is complete is useless because we are now committed to that design, whether it is good or bad[5].

Alexander has developed an amazing model in his differentiation process. This differentiation process recognizes the two perspectives of topsight and insight and the importance of balancing the two throughout the design process. Military command and control and software engineering also have recognized the existence of topsight and insight and the importance of their balance. Therefore, it is logical to use the differentiation process to improve or enhance military command and control systems and software development methodologies.

1. *At any given moment in a process, we have a certain partially evolved state of a structure. This state is described by the wholeness: the system of centers, and their relative nesting and degrees of life.*
2. *We pay attention as profoundly as possible to this wholeness - its global, large-scale order, both actual and latent.*
3. *We try to identify the sense in which this structure is weakest as a whole, weakest in its coherence as a whole, most deeply lacking in feeling.*
4. *We look for the latent centers in the whole. These are not those centers which are robust and exist strongly already; rather, they are centers which are dimly present in a weak form, but which seem to us to contribute to or cause the absence of life in the whole.*
5. *We then choose one of these latent centers to work on. It may be a large center, or middle-sized, or small.*
6. *We use one or more of the fifteen structure-preserving transformations, singly or in combination, to differentiate and strengthen the structure in its wholeness.*
7. *As a result of the differentiation which occurs, new centers are born. The extent of the fifteen properties which accompany creation of new centers will also take place.*
8. *In particular we shall have increase the strength of certain larger centers; we shall also have increased the strength of parallel centers; and we shall also have increased the strength of smaller centers. As a whole, the structure will now, as a result of this differentiation, be stronger and have more coherence and definition as a living structure.*
9. *We test to make sure that this is actually so, and that the presumed increase of life has actually taken place.*
10. *We also test that what we have done is the simplest differentiation possible, to accomplish this goal in respect of the center that is under development.*
11. *When complete, we go back to the beginning of the cycle, and apply the same process over.*

Figure 4.1: Alexander's Differentiation Process[5]

Chapter 5

The Knowledge Sharing Mechanism

5.1 The Knowledge Insight Model (KIM)

Another discipline that deals with complex problem solving is knowledge management. Knowledge management is defined as "helping people create knowledge and share and act upon information in ways that will measurably improve the performance" of the organization[24]. Knowledge takes two forms: explicit and tacit. Explicit knowledge can be easily communicated and shared because it can be expressed in words and numbers. Tacit knowledge, on the other hand, is highly personal and very difficult to express and share[26]. The difficulty of knowledge management is finding an effective means of not only managing the voluminous amount of explicit knowledge in a organization, but more importantly capturing and sharing the tacit knowledge in the organization.

One method of creating, sharing, and managing knowledge is the Knowledge Insight Model (KIM). The KIM begins with the Plan-Do-Check-Act (PDCA) Cycle as its approach to problem solving. The four stages of the PDCA Cycle shown in Figure 5.1 are[22]:

- **Plan** to improve your operations first by finding out what things are going

wrong (that is identify the problems faced), and come up with ideas for solving these problems.

- **Do** changes designed to solve the problems on a small or experimental scale first.
- **Check** whether the small scale or experimental changes are achieving the desired result or not.
- **Act** to implement changes on a larger scale if the experiment is successful.

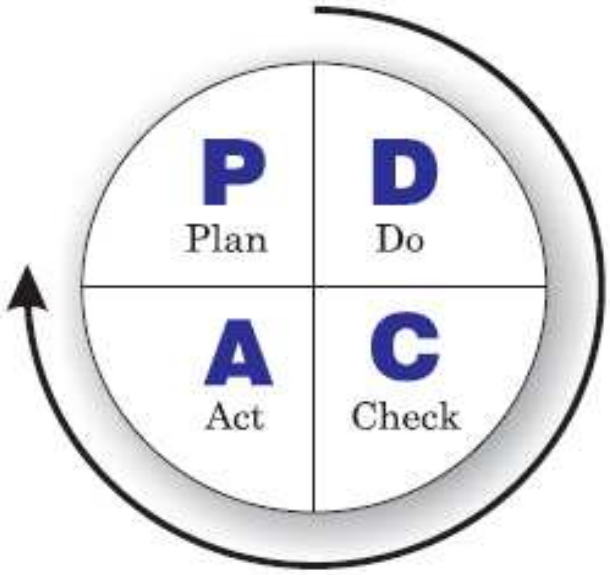

Figure 5.1: PDCA Cycle[22]

There are numerous iterations of the PDCA Cycle with each cycle having two potential outcomes. First, if the small scale changes implemented during the Do stage are verified as successful during the Check stage then the cycle progresses naturally to the Act stage for large scale implementation and finally back to the Plan stage for the next problem to be solved. However, if the small scale changes are determined to be unsuccessful during the Check stage then the cycle skips the Act stage and moves directly to the Plan stage in order to make modifications to the plan based upon the unsuccessful results and begin the cycle again[22].

The PDCA Cycle can be used in one of two purposes: discovery and refinement. In discovery, we are planning to be innovative and create something new. This could be the creation of new knowledge, a new process, or a new invention. This innovation is also

called *hoshin*, the Japanese word for innovation or breakthrough. In refinement, we are taking an existing piece of knowledge, process, or artifact and seeking to improve upon it. This refinement is called *kaizen*, the Japanese word for continuous improvement process. With each PDCA Cycle, we can either be in a *hoshin* or a *kaizen* mode depending on our purpose. Additionally, a successful innovation achieved during a *hoshin* PDCA Cycle can directly lead to a new *kaizen* PDCA Cycle because the innovation has allowed us to improve an existing process or artifact. A *kaizen* PDCA Cycle, likewise can directly lead to a *hoshin* PDCA Cycle[22]. The relationships between these two types of PDCA Cycles are shown in Figure 5.2.

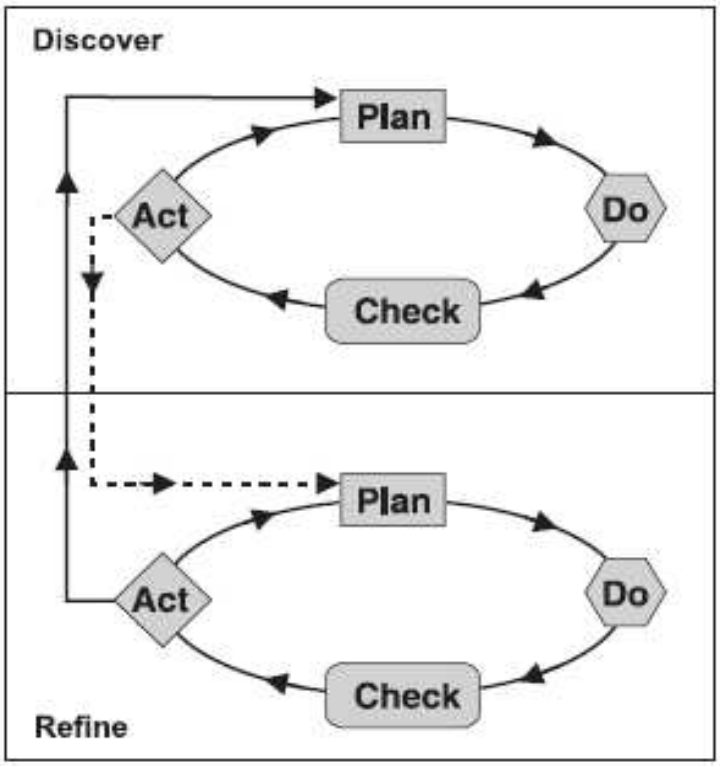

Figure 5.2: *Hoshin* and *Kaizen* PDCA Cycles[22]

An organization can use PDCA Cycles to either discover or refine its systems. Another dimension of these PDCA Cycles is where the systems operate. A system is external if it interfaces between the organization and its environment. An example could be a product that a business markets to its customers. Similarly, the system could be

internal if the system is encapsulated within the organization. An example would be the process that produces the item marketed by the organization. These opposite dimensions of discover and refine versus external and internal lead to the development of a 2 x 2 matrix showing the stages that an organization can be in when executing a PDCA Cycle, shown in Figure 5.3[22].

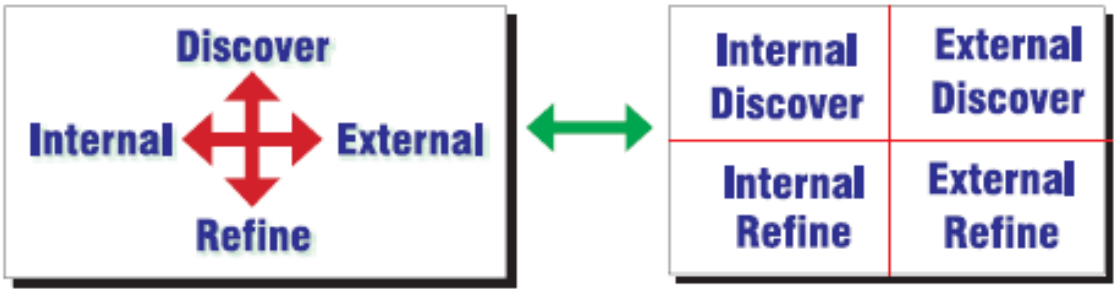

Figure 5.3: PDCA Cycle Stages of an Organization[22]

Transitions from discovery to refinement occur in solution increments since we can only refine a portion of a system at a time. Transitions from refinement to discovery represent the consolidation and synthesis of what has been refined and learned back into the creative process. There can be numerous iterations between the discovery and refinement modes. Transitions between the external and the internal depend on the state of creative knowledge. External forces dictate the need for internal innovation. During this internal innovation, we learn more about the system. A result of this learning is the need for additional external input. This new external information is then rationalized and allowed to refine the creative internal innovation. As a result, a mature product or process is returned to the external environment[22]. Incorporating the PDCA Cycle with this 2x2 matrix, we have the KIM shown in Figure 5.4.

5.2 The Four Roles

There are four roles that emerge from the KIM. These four roles are the Framer, the Maker, the Finder, and the Sharer, and they represent the four stages of the PDCA Cycle in the following manner: the Framer corresponds to Plan, the Maker corresponds to

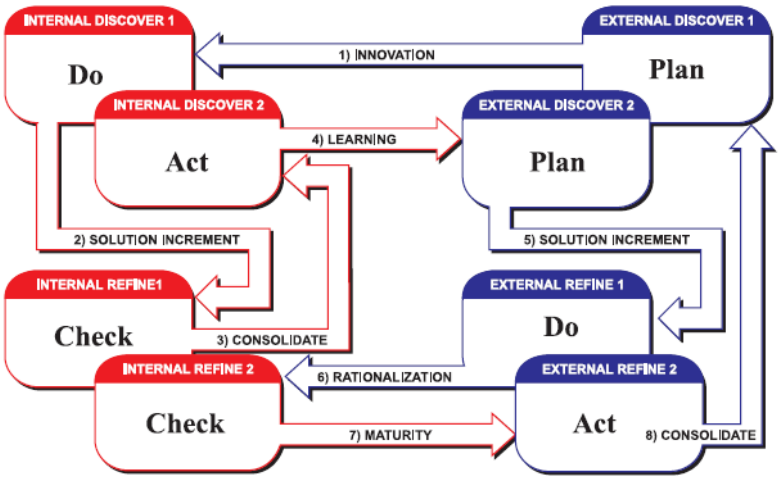

Figure 5.4: Knowledge Insight Model (KIM)[22]

Do, the Sharer corresponds to Check, and the Finder corresponds to Act. The Framer and the Sharer are complementary roles in that the combination of the two roles generates the entire KIM. Likewise, the Maker and the Finder are complementary roles. It is imperative to understand these four roles to gain further understanding from the KIM.

The Framer is responsible for understanding the problem and designing an overall architecture (or framework) that will solve the problem. The Framer sees the problem as a whole, determines the requirements needed for a solution to solve the problem, and then guides the process for developing the solution[21]. The Framer role in the KIM is shown in Figure 5.5.

The Maker is responsible for creating an innovative solution to the problem using the framework outlined by the Framer[21]. The Maker corresponds to the *Hoshin* (or breakthrough) process because the Maker is creating (or making) a new product or process that did not exist before[22]. The Maker role in the KIM is shown in Figure 5.6.

The Finder is responsible for finding resources to supplement the Maker's efforts. If the Finder is able to "find" the solution to a problem that the Maker is attempting to solve, then the Finder has saved the Maker from needlessly using his or her resources to create the solution from scratch[21]. The Finder also corresponds to the *Kaizen* (or continuous improvement process) by taking an existing product or process and refining it into a better product or process[22]. The Finder role in the KIM is shown in Figure 5.7.

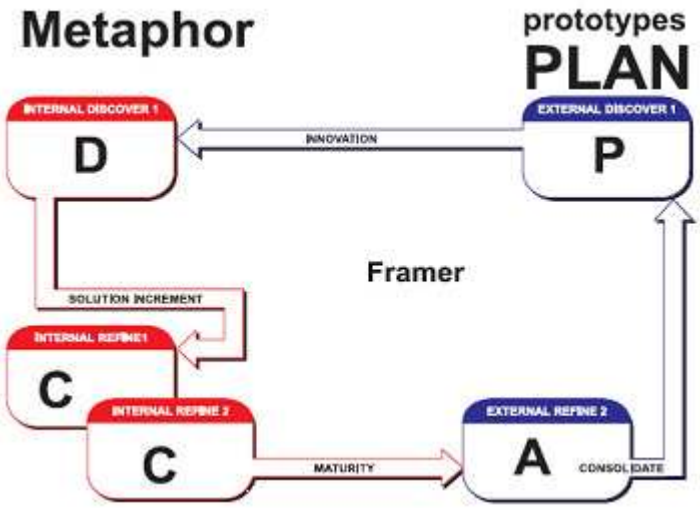

Figure 5.5: The Framer Role[22]

The Sharer is responsible for ensuring that the Framer, Maker, and Finder roles work together by sharing information. The Sharer provides feedback on the development of the overall framework and the specific portion of the solution that is currently being developed or refined[21]. The knowledge gained by the other three roles needs to be disseminated among them in order to enable a success process. The Sharer role in the KIM is shown in Figure 5.8.

5.3 Topsight and Insight Using KIM

The perspectives of topsight and insight are also present within the KIM. The Framer must use topsight in order to design an overall architecture for solving the problem. Without the ability to see the problem in its entirety, the Framer could only construct a partial framework around the problem and would be doomed to failure. The Maker and the Finder both use insight, because they are working with subsets of the entire problem. However, the Maker and the Finder use insight for different purposes. The Maker focuses on a subset of the entire problem in order to manufacture a new product or process that will assist in the solving of the overall problem. The Finder focuses on an already constructed portion of the solution with the intent of strengthening and improving it in order to better

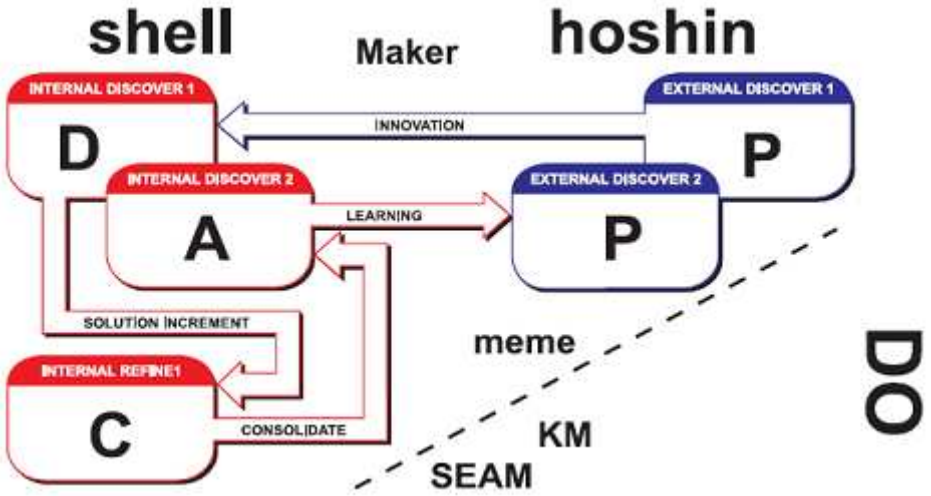

Figure 5.6: The Maker Role[22]

solve its portion of the overall problem.

5.4 The Importance of the Sharer

The Sharer is unique among the roles in that it is not confined to topsight or insight. Instead, the Sharer ensures that knowledge gained by the Framer, the Maker, and the Finder is shared among everyone. This means that the knowledge gained by topsight perspective of the Framer flows into the Maker and the Finder. Similarly, the insight perspectives of the Maker and the Finder are pushed back to the Framer. The Sharer is then involved with both topsight and insight perspectives, and the successful Sharer will balance these two perspectives. The Sharer is thus the most important role in the KIM because the Sharer keeps the Framer, the Maker, and the Finder working together. A Sharer that can not balance the topsight and the insight perspectives will inevitably result in a system dominated by the one perspective or the other. As previously mentioned, an imbalance of topsight and insight seriously jeopardizes the success of a system.

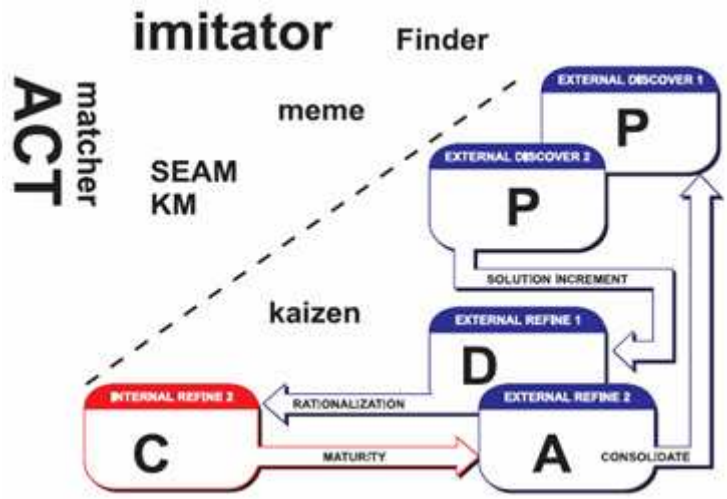

Figure 5.7: The Finder Role[22]

5.5 The Knowledge Sharing Mechanism (KSM)

The sharing of knowledge between a topsight perspective and insight perspective is critical to the designing of a successful system. The question then becomes how do we share the knowledge. As previously stated, there is tremendous power in the value of images to communicate effectively and efficiently. A vision can more succinctly convey a complete idea than merely listing facts and data regarding that idea. That is why a system metaphor is so important in XP, and why military commanders must visualize the situation and communicate that visualization using the commander's intent. Therefore, knowledge sharing must involve the use of image.

Nonetheless, the sharing of images is not sufficient to develop a solution to a complex problem. Action must be taken to craft and refine a good design. This action, however, is not haphazard or without thought. It must be undertaken with the understanding of how complex problems can be solved. This is where the power of Christopher Alexander's differentiation process comes into place. The differentiation process is based off the unfolding process of good design that Alexander observed over and over in both nature and historical architecture. The viewing of the problem and the problem's domain through the lens of wholeness and centers enables the development of a good solution.

It is this synthesis of images and Alexander's differentiation process that produces

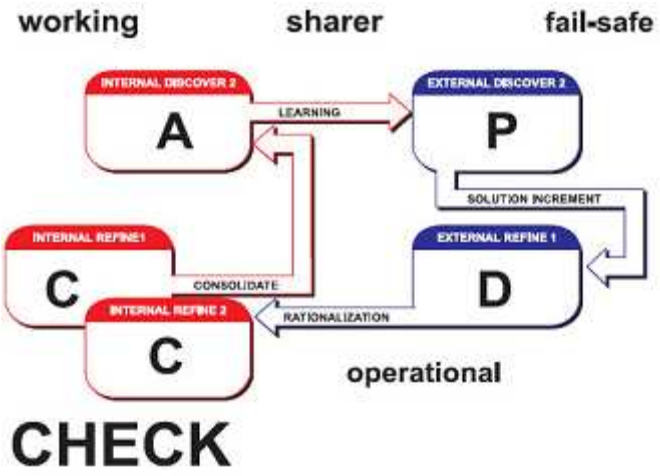

Figure 5.8: The Sharer Role[22]

the Knowledge Sharing Mechanism (KSM). The KSM is an iterative method for understanding a complex problem, developing a framework for solving that problem, creating, developing, and refining the parts of the solution for the problem, and then reassessing those partial solutions and overall framework until the complete solution has been fully developed. The KSM is outlined in Figure 5.9.

The first step is to clarify what the problem is and what factors will affect the problem and it solution. The Solution Vision in the second step is used to communicate the overall architecture of the solution. The desired end-state is the vision of the environment upon the successful implementation of the solution. The critical tasks are those tasks that must be accomplished in order to achieve success. The number of tasks should be limited to no more than seven tasks. Too many critical tasks will clutter the overall vision. The purpose is the reason why we are implementing the solution, and it gives motivation to the team developing the solution.

Viewing each critical task in step three as a center enables the solution to be more holistic. Most likely each critical task will impact or be related to the other critical tasks. Using typical decomposition techniques to tackle these tasks does not allow for the solution design to account for these complex interactions. Conversely, viewing the tasks as centers allows us to explore and anticipate these interactions.

Step four begins the iteration process. At the beginning of each iteration, we select

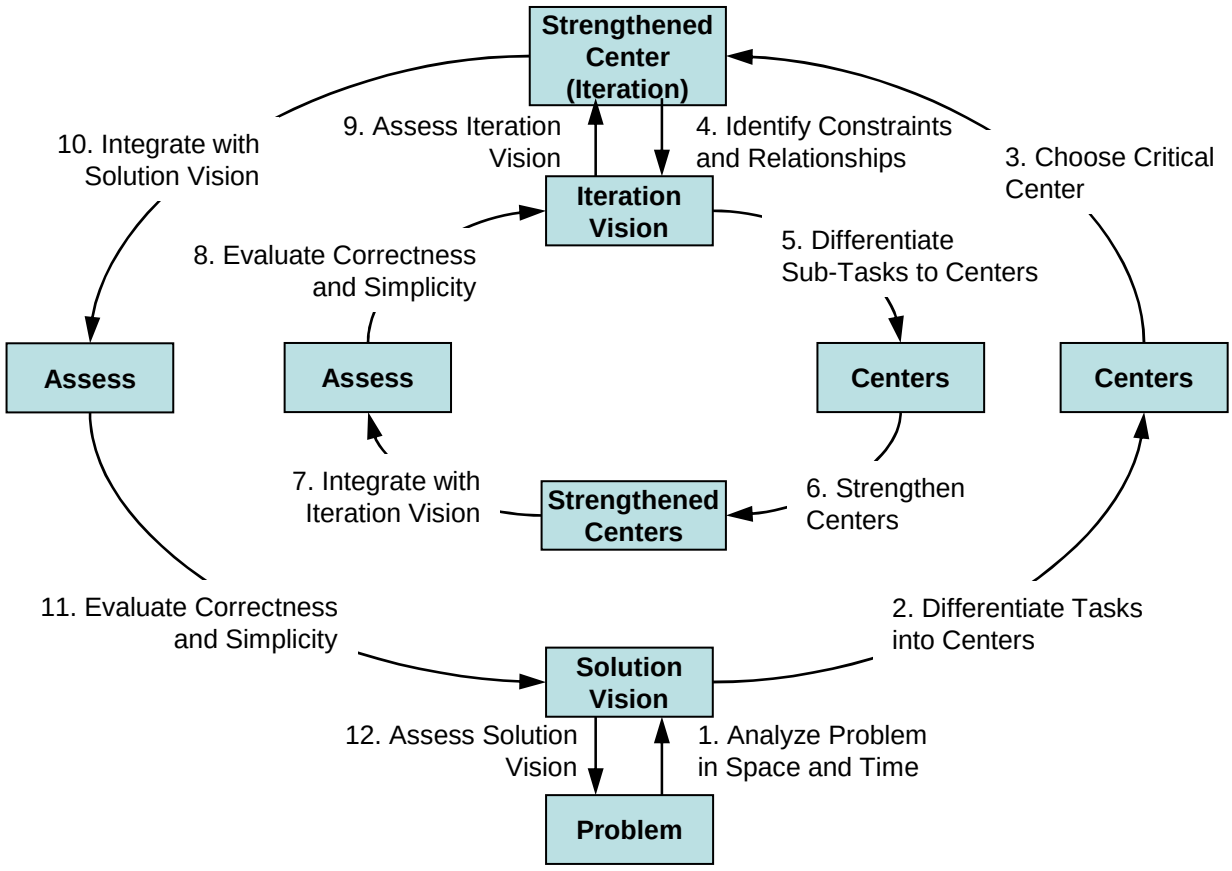

Figure 5.9: Knowledge Sharing Mechanism (KSM)

the most critical center to strengthen. The basis of the selecting the most critical center depends on each particular situation. This center could be the weakest center, it could be the center that must developed first, or both conditions could apply. The Iteration Vision is developed and has the same components as the Solution Vision. Since the Iteration Vision is patterned exactly like the Solution Vision, the KSM ensures that both the topsight and the insight view of the solution can be easily communicated and understood.

Once the chosen center has been improved or strengthened, we conduct an assessment on whether our improvements have achieved the Iteration Vision. In addition, we check to make sure that the improvements we have implemented are the simplest improvements as possible. This requires removing any extraneous and unnecessary steps and modifications. This assessment gives us feedback to the iteration process that corresponds

to the Check stage of the PDCA Cycle. We also assess the Iteration Vision as to whether it supports the Solution Vision and if the Iteration Vision is still valid. It is possible that vision formulated at the beginning of the iteration is no longer valid due to either changing requirements, changing knowledge about the problem, or both. This feedback is critical to ensuring the proper growth of the solution.

In addition to assessing each Iteration Vision at the end of each iteration cycle, we also assess the Solution Vision. The Solution Vision can change based off knowledge gained during the solution development process. The desired end-state envisioned at the beginning of the process may differ from the final desired end-state of the solution. This feedback builds the necessary flexibility into the system in order to deal with uncertainty.

5.6 The Existence of KSM in Software Engineering

This knowledge occurs in successful software development, even though it may not be explicitly identified. One such successful software project is the U.S. Army's Maneuver Control System (MCS) Light program. MCS Light is a command and control program that operates on a notebook or desktop Windows platform. MCS Light is used for monitoring the movements and operations of subordinate and adjacent units within a command, as well a planning tool for future operations[36].

The development of MCS Light began with an initial, but flexible, architecture based on an analysis of the systems goals and requirements. The program then went into a series of iteration cycles. The MCS Light development team would issue a release every three months. At the beginning of the three month cycle, the team would determine which objectives of the overall architecture were to be implemented. These objective were then translated into user stories that started a four-week iteration cycle of design, coding, testing, and system integration. After three iteration cycles, the team had a deliverable release. It is important to note that the deliverable release provided feedback to the overall mission analysis with the goals of adapting and clarifying those goals and the overall architecture as shown in Figure 5.10[36].

The KSM is present, even though it is not identified in the MCS Light project. The overall mission analysis and resulting mission goals and architecture correspond beautifully to the Solution Vision of the KSM. A subset of the mission goals, called the release objec-

tives, is then chosen to implement in the three month release window. This corresponds to selecting of a critical center and the development of an Iteration Vision which supports the Solution Vision. The feedback of the knowledge gained during the three month release process correlates well to the idea of assessing the validity of the Iteration Vision and the Solution Vision. The knowledge sharing occurs with the mission goals translating into the release objectives, and the feedback of knowledge back into the mission goals.

The MCS Light project development team gained feedback regarding each release through interactions with the customers (i.e. soldiers) who use the system. The project team participated in training exercises in both garrison and field environments, and some even deployed with combat units to Afghanistan and Iraq in order to get feedback on the current release and what was needed for the next release (and ultimately the system as a whole). The MCS Light is considered a success by the fact that as of 2004, nine of the ten active duty U.S. Army divisions had adopted the MCS Light program as their chief planning and operations command and control software system. Lieutenant General John Vines, former commander of the 82nd Airborne Division and 18th Airborne Corps, credited the MCS Light program as the best command and control tool available at the present time[36].

This example shows how knowledge sharing enables successful software development. The KSM is a framework for instituting and enabling this knowledge sharing.

5.7 The Existence of KSM in Command and Control

Knowledge sharing is also a critical component to the success of military command and control. One such successful example of knowledge sharing took place during the Allied invasion of Normandy, France, on D-Day, June 6, 1944.

The Normandy invasion involved amphibious landings on five beaches code-named Omaha, Utah, Gold, Sword, and Juno. The American 4th Infantry Division was the lead division under VII Corps to land on Utah beach. The 4th Infantry Division's mission was to conduct an amphibious landing at Utah beach, secure the beachhead, and push inward along the causeways in order to expand the beachhead. The control mechanism for coordinating this complex movement and landing of troops, tanks, and other vehicles consisted four control vessels (two primary vessels and two secondary vessels). During the conduct of the

landings, the two primary vessels were destroyed and one of the secondary vessels had to be diverted away from the beach in order to guide in the amphibious tanks. In addition, the troop landing crafts missed their intended locations, and soldiers from the 4th Infantry Division ended up 2,000 yards south of their objectives[14].

This great error in the location of the actual landings could have created tremendous confusion. Each assault unit had detailed orders that they were to execute immediately upon landing. Those orders were no longer valid because the units were in the wrong place. It was at this time that Brigadier General Theodore Roosevelt, Jr., the assistant division commander, recognized the error and took action. General Roosevelt conducted an individual reconnaissance of a route from where the division landed at to the causeways, the division's objectives. General Roosevelt then coordinated with and directed two battalion commanders to neutralize the enemy immediately facing the division and lead the division north toward the causeways. General Roosevelt's recognition of the problem and quick adaptation of the battle plan resulted in the rapid landing of the division and its subsequent movement inward toward its objectives. As a result, the entire 4th Infantry Division (minus one battalion) successfully landed at Utah beach in the first 15 hours[14].

In this example, the 4th Infantry Division had developed a Solution Vision with an end-state of a secure and expanded beachhead. The critical tasks were to conduct an amphibious landing, secure the beachhead, and secure the causeways. Each of those critical tasks can be viewed as sequential Iteration Visions. Bad fortune through the loss of the control vessels and errant landings invalidated the current Iteration Visions. Without feedback about the changing environment, these plans were doomed to fail. General Roosevelt acted as knowledge sharing mechanism. Knowing the desired end-state and their current orders, General Roosevelt recognized the disparity, adapted the orders based on the division's current state, and provided feedback to both his subordinates and his superiors regarding the change in plans. The overall end-state of the Solution Vision did not change, but the supporting Iteration Visions had to adapt to the challenging circumstances. This successful knowledge sharing enabled the rapid movement of the division through the beachhead and onto the designated causeways.

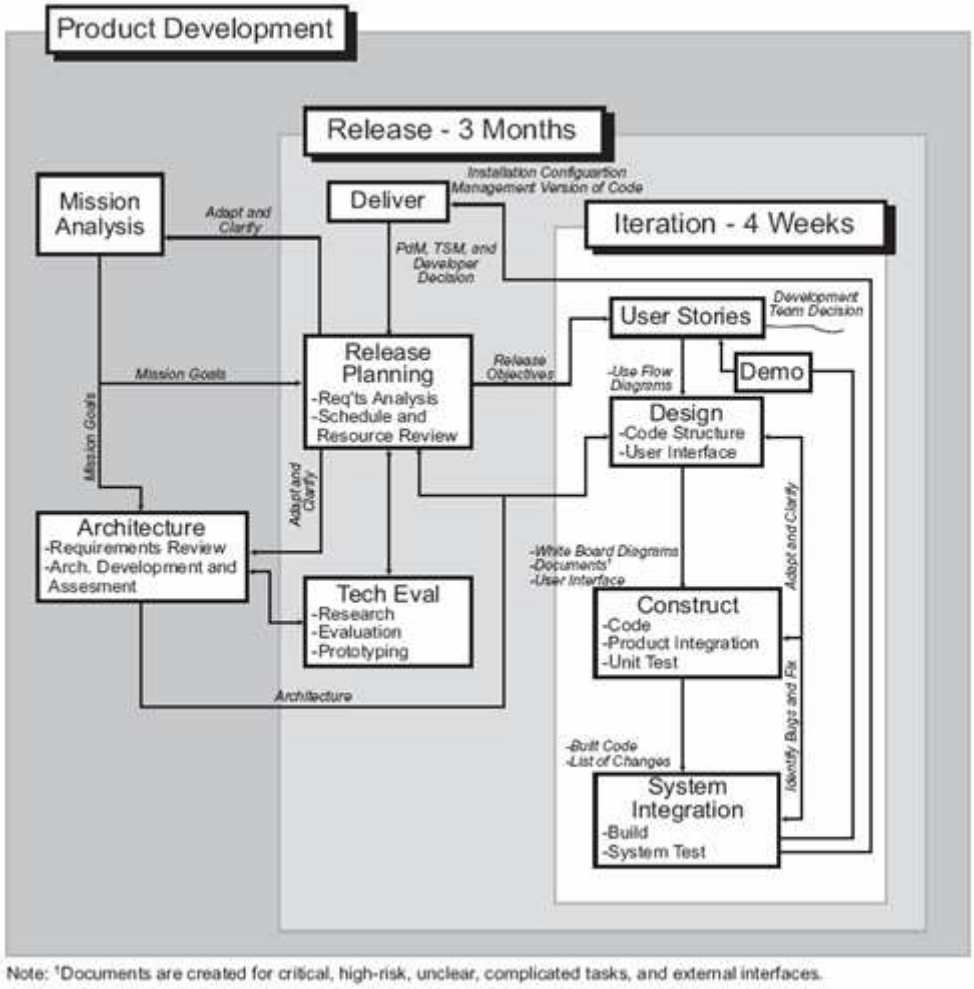

Figure 5.10: MCS Light Project Development Process[36]

Chapter 6

Analysis of the KSM

6.1 Purpose for the Analysis

The preceding chapters established the case for and outlined the proposed KSM as a means for solving complex problems in software engineering and command and control. In addition, previous text detailed how knowledge sharing contributed to the success of a software project and pivotal battle during World War II. The purpose of this chapter is to analyze the KSM in regards to both software engineering and command and control. Proposing a solution is not sufficient to ensure that problem of balancing topsight and insight has been adequately addressed. The proposed solution must be evaluated within the environments that it must operate in order to determine the worth of the proposed solution.

6.2 Analysis of the KSM in Software Engineering

Software engineering methodologies can be evaluated against two standards: validation and verification. IEEE[33] defines validation as "the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the specified requirements." Another way to phrase validation is "Did we build the right product?" Validation is ultimately in the eyes of the customer. Does the delivered

software product meet the expectations of the customer? The customer's expectations are formalized as requirements, but these requirements will inevitably undergo changes as the software naturally develops and evolves. IEEE[33] defines verification as "the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed as the start of that phase." Verification can also expressed as, "Did we build the product (or component) correctly?" Verification is based upon mathematical proof and facts. If the purpose of the software component is to sort a group of records alphabetically, then verification asks, "Does the software component correctly sort the records?"

The KSM addresses validation in the Solution Vision. The Solution Vision communicates the customer's expectations by constructing a word image. This image consists of the customer's purpose for the software product, the most critical tasks for the software product, and the desired end-state of the customer's environment as a result of the software product. These three components of the Solution Vision provide a concise image that can be readily understood, negotiated, and communicated between the developer and the customer. This type of vision and understanding is just not possible with an enormous laundry list of detailed technical requirements.

KSM also allows the validation process to be, in fact, valid. The IEEE definition of validation referred to the satisfaction of specified requirements. If the requirements for the product do not change throughout the software development process then this definition is sufficient. Changing requirements, however, are a fact of life in software development. One reason, according to Fred Brooks[13] is that customer "usually does not know what questions must be answered, and he has almost never thought of the problem in detail for specification." The feedback mechanism in the KSM enables the Solution Vision to grow and evolve as both the customer and the developer acquire knowledge about the software's requirements and how the software will achieve the desired end-state for the customer's environment.

The KSM addresses verification in the Iteration Vision. The Iteration Vision, like the Solution Vision, consists of purpose for the chose software component, critical tasks for that component, and a desired end-state of the software product as a result of the development of the specific component. Verification occurs during the assessment of the Iteration Vision. The feedback mechanism forces the developer to verify the correctness of the developed component in regards to achieving the specified critical tasks. The KSM has an

additional benefit in that during the assessment phase the developer not only checks for correctness, but also to simplify the code. This simplification of the code is akin the XP practice of refactoring which Kent Beck[7] defines as "a change to the system that leaves its behavior unchanged, but enhances some nonfunctional quality–simplicity, flexibility, understandability, performance." The KSM not only satisfies the traditional definition of verification, but takes it to the next level by improving the design of code through simplification.

The Solution Vision and Iteration Vision of the KSM correspond very well to the standards of validation and verification. The KSM not only achieves these two standards in the traditional sense, but it also enhances them. The KSM enables flexibility and growth in customer expectation in regards to validation, and it incorporates a form of refactoring into the area of verification.

6.3 Analysis of the KSM in Command and Control

One of the most important characteristics in today's military command and control system is agility. This is not only true of the United States military, but in other countries' militaries such as the United Kingdom. What does it mean for a command and control system to be agile? Dr. David Alberts[2] defines the six attributes of an agile command and control system as the following:

1. Robustness: the ability to maintain effectiveness across a range of tasks, situations, and conditions.
2. Resilience: the ability to recover from or adjust to misfortune, damage, or a destabilizing perturbation in the environment.
3. Responsiveness: the ability to react to a change in the environment in a timely manner.
4. Flexibility: the ability to employ multiple ways to succeed and the capacity to move seamlessly between them.
5. Innovation: the ability to do new things and the ability to do old things in new ways.
6. Adaptation: the ability to change work processes and the ability to change the organization.

In order to be relevant for today's command and control environment, the KSM must display those six attributes of an agile command and control system.

The first attribute to evaluate KSM against is robustness. KSM is a robust system because it built around solving complex problems, not just specific instances of those problems. Traditional military command and control systems focused on engaging an enemy force in combat operations. Today's environment is dramatically different. A military unit today is expected to decisively engage and destroy an enemy in combat operation, provide civil assistance in rebuilding a country's infrastructure, and assisting in disaster relief. Furthermore, these tasks can take place simultaneously or in very rapid succession. From my personal experience, the 1st Battalion, 39th Field Artillery (MLRS) had to transition quickly from providing rocket and missile fires to the 3rd Infantry Division during Operation IRAQI FREEDOM to providing checkpoint security for the Baghdad International Airport and assisting the local population within the airport complex with meeting their basic needs. By viewing military command and control as a subset of complex problem solving, the KSM can be quickly and ably applied to these varied missions.

The second attribute is resilience. Plans will not always proceed as expected and bad fortune will occur. The KSM is resilient because of the feedback mechanism built into both the Solution Vision and the Iteration Vision. The feedback mechanism allows the KSM to adjust, if necessary, the Iteration Vision and/or the Solution Vision based off the current situation. The KSM is also resilient in the fact that the assessment of the Iteration Vision looks to see if the vision could have been implemented better (i.e. simpler). If there is a better way, then that better way is implemented. This enables the command to learn from the experience of that Iteration and apply the lessons learned toward the future.

The third attribute is responsiveness. The uncertainty and unpredictability of today's environment demand that a military be agile, and an organization can not be agile if it can not respond to its environment. The continuous assessment and evolution, if necessary, of the Solution Vision enables the KSM to be responsive to changes in its environment and to changes in the organization's understanding of the environment. The KSM is not locked into a detailed plan that becomes invalid shortly after the implementation of that plan begins.

The fourth attribute is flexibility. The use of vision statements for the overall solution and for each iteration of a solution component instills a great deal of flexibility. The destination of each iteration is established, but not the journey to reach that destination. People are then able to use their initiative and tacit knowledge to develop innovative solutions to the problems. This idea is in-line with General George S. Patton, Jr.'s[28] directive

to "never tell people how to do things. Tell them what to do and they will surprise you with their ingenuity."

The fifth attribute is innovation. As just mentioned, the KSM's flexibility encourages innovation on the part of individuals. KSM also encourages innovation by viewing critical tasks as centers and not discrete objects. A center is not isolated from its environment; a center interacts and is a part of that environment. The use of centers enables people from various backgrounds and expertise to analyze and develop solutions for a problem instead of assigning to the problem to a specific staff element. This idea of using an inter-disciplinary approach to solving problems has already taken root in the U.S. Army. Lieutenant General David McKiernan, Commander of the Combined Forces Land Component Command during Operation IRAQI FREEDOM, restructured his staff around operational functions instead of the traditional vertical staff stovepipes. Other Army units also organized their staffs into multi-discipline functional cells[20].

The sixth and final attribute is adaptation. Step-by-step adaptation is at the heart of the KSM. This adaptation along with awareness of the whole, feedback, and uncertainty constitute the enabling conditions for Alexander's differentiation process and form the foundation of the KSM.

The KSM possesses the desired attributes of an agile command and control system. Therefore, the KSM can be utilized as framework for command and control for today and tomorrow's military.

Chapter 7

Integration of the KSM

7.1 Purpose for Integration

The preceding chapter analyzed the KSM in regards to both software engineering and command and control. As a result of the analysis, KSM was shown to satisfy the software engineering components of validation and verification, as well as satisfy the attributes of an agile command and control system. Based on that knowledge, the purpose of this chapter is to demonstrate how to integrate the KSM in both environments. It goes beyond a theoretical discussion and addresses practical implementations of the KSM.

7.2 Integrating KSM into Software Engineering

Integration of the KSM into software development does not mean creating a new process from scratch. The KSM can be adapted into an already existing agile software development methodology such as Scrum[32]. A key attribute of the KSM is agility, and thus it is logical that the KSM can be more readily adapted to an agile methodology rather than a plan-driven methodology.

There are two key deficiencies with the current implementation of Scrum. First, the Product Backlog does not address design. The Product Backlog represents a laundry list of items to be accomplished. There is, however, not a method to the development of

the Product Backlog. Each derived Sprint Goal and corresponding Sprint Backlog is simply based on what the Scrum Team believes that it can accomplish in the next 30-day Sprint cycle. At the completion of the Sprint, there is only a four hour Sprint Review meeting, immediately preceding the next Sprint cycle, allocated to capture the lessons learned from the completed Sprint. There is no mechanism to capture the feedback learned through the conduct of the Sprint to shape the overall Product design.

The second deficiency is that there is no feedback between the customer (also called the Product Owner) and the Scrum team for 30 days at a time. The customer can attend the Daily Scrums, but he/she does not have a part in them. The following is a description of how the KSM can be incorporated into Scrum.

The Scrum Master works in conjunction with the customer to create a Product Vision. The Product Vision entails the customer's purpose for the product, the key functionalities (or tasks) of the product, and the desired customer's environment as a result of the product. The Product Backlog of requirements must support this Product Vision. Prior to each Sprint cycle, the customer and the Scrum Master identify the product tasks that are the most critical at the moment. The Scrum Master and the Scrum Team then develop a Sprint Vision that will support the Product Vision. The Sprint Vision entails the Scrum Team's purpose for the current Sprint, the critical tasks that must and can be accomplished during the Sprint, and how the resulting Product end-state as a result of the Sprint. The Scrum Team will then derive the Sprint Backlog from the Sprint Vision.

Each Scrum Team member will brief how their work supports the Sprint Vision during the Daily Scrums. This is in addition to the already required briefing items of what has been accomplished since the last Scrum, what will be accomplished prior to the next Scrum, and what obstacles they are facing in accomplishing their stated goals. Also during the Daily Scrum, each member will have the opportunity to address whether the Sprint Vision or even the Product Vision needs to be changed or refined. It is critical that everyone has a clear understanding of the Product Vision and the current Sprint that is supporting it. On a weekly basis, the Scrum Master will meet with the customer to note the progress, as well as the hindrances, of the Scrum Team in relation to the Sprint Vision and the Product Vision. During this weekly meeting, the Scrum Master can address proposed changes to the Sprint Vision and/or Product Vision with the customer prior to the end of the Sprint.

At the end of the Sprint, the delivered increment is compared to the Sprint Vision

and the Product Vision. This is where the customer will most likely alter the Product Vision based on knowledge gained from the delivered increment and knowledge gained through the development process. Another critical component of the Sprint Review process is to identify if the solutions implemented during the Sprint can be done in a simpler way.

7.3 Integrating KSM into Command and Control

Army command and control is currently implemented using a process called the Military Decision Making Process (MDMP). The MDMP is a seven-step process that prescribes the manner in which Army commanders with their staffs develop operational orders. The seven steps are listed in Figure 7.1.

Step 1: Receipt of Mission
Step 2: Mission Analysis
Step 3: Course of Action Development
Step 4: Course of Action Analysis(War Game)
Step 5: Course of Action Comparison
Step 6: Course of Action Approval
Step 7: Orders Production

Figure 7.1: Military Decision Making Process (MDMP)[19]

Arguably, the most important step in the MDMP is Step 2: Mission Analysis. Mission analysis is important because "both the process and the products of mission analysis help commanders refine their situational understanding and determine their vision." In addition, the mission analysis will be the foundation for developing, analyzing, and comparing the Courses of Action that will ultimately be implemented as result of this MDMP. Mission analysis consists of 17 tasks which are generally, though not necessarily, done in sequential order[19]. Those tasks are shown in Figure 7.2.

Mission analysis Tasks 1 through 11 in Figure 7.2 must be completed before Task 12, Deliver a Mission Analysis Briefing. Following the conclusion of the staff's mission analysis briefing to the commander, the commander personally completes Tasks 13 through 15 and the staff, in turn, then completes Tasks 16 and 17.

All of the mission analysis tasks are necessary, but they are organized in a focused

1. Analyze the Higher Headquarters Order
2. Perform Initial Intelligence Preparation of the Battlefield
3. Determine the Specified, Implied, and Essential Tasks
4. Review Available Assets
5. Determine Constraints
6. Identify Critical Facts and Assumptions
7. Perform Risk Assessment
8. Determine Initial Commander's Critical Information Requirements and Essential Elements of Friendly Information
9. Determine the Initial Intelligence, Surveillance, and Reconnaissance Plan
10. Update the Operational Timeline
11. Write the Restated Mission
12. Deliver a Mission Analysis Briefing
13. Approve the Restated Mission
14. Develop the Initial Commander's Intent
15. Issue the Commander's Planning Guidance
16. Issue a Warning Order
17. Review Facts and Assumptions

Figure 7.2: Mission Analysis Tasks[19]

fashion. The most important part of communicating the vision of the operation, developing the commander's intent, is not performed until the end of the mission analysis brief. In addition, the commander alone formulates this intent instead of allowing input from the staff which then the commander can approve, disapprove, or modify.

The KSM can help guide the mission analysis process by focusing the staff to visualize the mission through an overall Solution Vision. During Task 1, Analyze the Higher Headquarters Order, the staff should visualize the Solution Visions developed by their superior unit and their superior's superior unit. This will help the staff visualize how the Solution Vision they are developing supports the Solution Visions of their higher headquarters. Specifically, it is critical for the staff to develop the End State for the Solution Vision. Task 2, Perform Initial Intelligence Preparation of the Battlefield, helps to define the problem environment.

The essential tasks identified in Task 3 become the Critical Tasks for the Solution Vision. The staff then transforms these Critical Tasks into centers and performs successive iterations developing an Iteration Vision for each and subsequently developing and strengthening each. The staff can either perform the iteration cycles in the chronological order that each Solution Vision Critical Task must be executed or they can start on the most Critical Task in regards to the overall success of the mission.

Tasks 4 through 11 naturally will be developed as the staff performs the successive iteration cycles and assessing them against their respective Iteration Visions and the overall Solution Vision. During the mission analysis briefing, the staff would present the developed Solution Vision along with its strengthened centers to the commander. The commander can then approve, modify, or even disapprove the Solution Vision at the conclusion of the mission analysis briefing. It is very unlikely the commander will flat out disapprove the Solution Vision because this would signify a major miscommunication between the commander and his/her staff. Most likely, the commander will have small adjustments to the Solution Vision based on his/her experience.

The benefit of the KSM in the mission analysis is that now the knowledge of the problem, in this case the unit's mission, is being developed and communicated in a high level picture between the staff and commander instead of the staff presenting facts to the commander for the commander to visualize on his/her own. Thus, there is more knowledge sharing taking place between the commander and staff which help facilitate unit success.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Software engineering and command and control will continue to grow more and more complex. As such the only constants that we can count on are change and unpredictability. We can not eliminate the change and unpredictability. Instead, we must accept these realities and deal with them.

In this thesis, we have illustrated the parallel histories and complexities that both software engineering and command and control systems face. Both can be viewed as instances of complex problem solving. By tying these two diverse fields to a higher level abstraction, we could then gain insight into each environment by the lessons learned from the other environment. We were also able to incorporate other philosophies of dealing with complex problem solving to further our understanding. These philosophies were Alexander's unfolding and differentiation processes, military image theory, and the Knowledge Insight Model. The result of bringing all of these perspectives together was the Knowledge Sharing Mechanism (KSM).

After laying the foundation for the KSM, this thesis articulates what the KSM consists of. The next step was to take a critical look of the KSM and analyze it using evaluation criteria from both software engineering and command and control. The KSM passed both evaluations. Finally, this paper outlined two practical ways to incorporate

the KSM into the software engineering and command and control communities. These are not the only ways to integrate KSM, but ultimate the KSM must be translated from the theoretical to the practical in order to be of any benefit.

The KSM provides a framework for acknowledging and addressing the change and uncertainty that are inevitable in any complex problem solving. By using this framework, one can better mitigate the risks and challenges of change and uncertainty and increase the chances of a successful solution in whatever field it may be.

8.2 Future Work

The next logical step would be to collect empirical data on a software engineering project and a military command and control problem using the KSM. A possible way to test the KSM in software engineering is to introduce it in an advanced undergraduate or graduate computer science course and have the experimental group use the KSM on a semester project and the control group use an agile method such as XP or Scrum. Through the conduct of the experiment, survey both groups on their understanding of the software problem, their proposed solutions, and how their specific methodology (KSM or Scrum) helps or hinders their understanding. Finally, the delivered product and its results can provide feedback to both methodologies.

In order to test the KSM in a military environment, it would be best to incorporate it into a maneuver (infantry or armor) battalion. Train the staff to use the KSM in order to modify the battalion's particular MDMP. Use and evaluate the KSM as the battalion conducts unit level training, brigade field exercises, and ultimately during a Combat Training Center(CTC) rotation such as the National Training Center at Fort Irwin, California, or the Joint Readiness Training Center at Fort Polk, Louisiana. The battalion staff's feedback and the feedback of the observer/controllers at both home station and the CTC would be invaluable to compare to the typical results of using MDMP at both home station and at the CTC.

The Department of Defense is currently in the midst of transforming itself. It is pushing forward an idea called Net-Centric Warfare that acknowledges the importance of networks and information sharing. These networks are not only technological networks that we would think of in computer science, but they are also social networks were people

communicate with one another. In today's environment, the actions of a sergeant and private at a checkpoint could have more ramifications on the success of a mission than decisions made by generals and colonels. Being a member of the Department of Defense, I will be actively involved in this transformation and hopefully can be of some benefit to it. I believe that the KSM is one of those tools that can benefit this transformation.

Bibliography

[1] Agile Alliance. Manifesto for agile software development, 2001. http://www.agilemanifesto.org, accessed February 17, 2006.

[2] Alberts, D. S., and Hayes, R. E. *Power to the Edge, Command Control in the Information Age*. Information Age Transformation Series. Department of Defense Command and Control Research Program, 2003.

[3] Alexander, C. The origins of pattern theory: The future of the theory, and the generation of a living world. *IEEE Software 16*, 5 (Sep–Oct 1999), 71–82. Keynote speech given by Christopher Alexander in October 1996 at OOPSLA '96 in San Jose, CA.

[4] Alexander, C. *The Phenomenon of Life: An Essay on the Art of Building and the Nature of the Universe*, vol. 1 of *Nature of Order*. Center for Environmental Structure, Berkeley, CA, 2002.

[5] Alexander, C. *The Process of Creating Life: An Essay on the Art of Building and the Nature of the Universe*, vol. 2 of *Nature of Order*. Center for Environmental Structure, Berkeley, CA, 2002.

[6] Alexander, C. New concepts in complexity theory, May 2003. http://www.natureoforder.com/library/scientific-introduction.pdf, accessed February 14, 2006.

[7] Beck, K. *eXtreme Programming Explained: Embrace Change*. Addison-Wesley Publishing Company, Boston, MA, 1999.

[8] Benigno, K. W., 2003. Quote by MAJ Kenneth W. Benigno, Operations Officer for 1-39 FA (MLRS), 3d Infantry Division, during Operation Iraqi Freedom.

[9] Boehm, B., Egyed, A., Port, D., Shah, A., Kwan, J., and Madachy, R. A stakeholder win-win approach to software engineering education. *Annals of Software Engineering 6*, 1 (Mar 1998), 295–321.

[10] Boehm, B., and Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Publishing Company, Boston, MA, 2003.

[11] Boehm, B. W. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes 11*, 4 (Aug 1986), 21–42.

[12] Brooks, Jr., F. P. *The Mythical Man-Month*. Addison-Wesley Publishing Company, Reading, MA, 1982.

[13] Brooks, Jr., F. P. No silver bullet: Essence and accidents of software engineering. *Computer 20*, 4 (April 1987), 10–19.

[14] Center of Military History. *Utah Beach to Cherbourg (6 June - 27 June 1944)*. Armed Forces in Action. United States Army, 1990. http://www.army.mil/cmh-pg/books/wwii/utah/utah3.html, accessed March 3, 2006.

[15] Control Chaos. What is scrum?, January 2001. http://www.controlchaos.com/about/, accessed January 31, 2006.

[16] Department of Defense. Quadrennial defense review report. Tech. rep., Department of Defense, Washington, DC, February 2006. http://www.defenselink.mil/pubs/pdfs/QDR20060203.pdf, accessed February 27, 2006.

[17] Department of the Army. *Field Manual 3-0, Operations*. Department of the Army, Washington, DC, June 2001.

[18] Department of the Army. *Field Manual 6-0, Mission Command: Command and Control of Army Forces*. Department of the Army, Washington, DC, August 2003.

[19] Department of the Army. *Field Manual 5-0, Army Planning and Orders Production*. Department of the Army, Washington, DC, January 2005.

[20] Fontenot, G., Degen, E. J., and Tohn, D. *On Point: US Army in Operation IRAQI FREEDOM.* Combat Studies Institute Press, Fort Leavenworth, KS, 2004. Operation IRAQI FREEDOM Study Group mandated by GEN Erik K. Shinseki, US Army Chief of Staff, on April 30, 2003.

[21] Honeycut, T. L., and Kocherlakota, S. M. A knowledge insight framework for knowledge discovery and data mining. North Carolina State University, 2002.

[22] Honeycutt, T. L. Knowledge enabling organon: Knowledge executive officer, 2001.

[23] Krulak, C. The strategic corporal: Leadership in the three block war. *Marine Corps Gazette 83*, 1 (January 1999), 18–22.

[24] National Aeronautics and Space Administration. What is knowledge management?, 2004. http://km.nasa.gov/whatis/index.html, accessed February 28, 2006.

[25] Naur, P., and Randell, B., Eds. *Software Engineering* (Garmisch, Germany, October 1968), NATO Science Committee, North Atlantic Treaty Organization.

[26] Nonaka, I., and Takeuchi, H. *The Knowledge-Creating Company.* Oxford University Press, New York, NY, 1995.

[27] Paparone, C. R. U.S. Army decisionmaking: Past, present, and future. *Military Review 81*, 4 (July–August 2001), 45–53.

[28] Patton, Jr., G. S. *War As I Knew It.* Houghton Mifflin Company, Boston, MA, 1947.

[29] Pressman, R. S. *Software Engineering: A Practioner's Approach*, 4th ed. The McGraw-Hill Companies, Inc., New York, NY, 1997.

[30] Royce, W. W. Managing the development of large software systems. In *Proceedings of IEEE WESCON* (New York, NY, August 1970), IEEE Computer Society Press, pp. 1 – 9.

[31] Rumsfeld, D. DOD acquisitions and logistics excellence week kickoff - bureacracy to battlefield, September 2001. http://www.defenselink.mil/speeches/2001/s20010910-secdef.html, accessed February 27, 2006.

[32] Schwaber, K., and Beedle, M. *Agile Software Development with Scrum.* Prentice Hall, Upper Saddle River, NJ, 2001.

[33] Standards Coordinating Committee of the Computer Society of the IEEE. IEEE Standard Glossary of Software Engineering Terminology, December 1990. IEEE Std 610.12-1990 (Revision and redisignation of IEEE Std 792-1983).

[34] United States Marine Corps. *Marine Corps Doctrinal Publication 6, Command and Control.* Department of the Navy, Washington, DC, October 1996.

[35] Williams, L., and Kessler, R. *Pair Programming Illuminated.* Addison-Wesley Publishing Company, Boston, MA, 2002.

[36] Willison, J. S. Agile software development for an agile force. *Crosstalk* (April 2004), 16–19.