12th ICCRTS
Adapting C2 to the 21st Century

"Knowledge Sharing Mechanism: Enabling C2 to Adapt to Changing Environments"

David P. Harvie
Major, United States Army

Department of Electrical Engineering and Computer Science
Building 601
United States Military Academy
West Point, New York 10996
(845) 938-3233
david.harvie@usma.edu

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2007		**00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Knowledge Sharing Mechanism: Enabling C2 to Adapt to Changing Environments	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **United States Military Academy,Department of Electrical Engineering and Computer Science,Building 601,West Point,NY,10996**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Twelfth International Command and Control Research and Technology Symposium (12th ICCRTS), 19-21 June 2007, Newport, RI

14. ABSTRACT

The environments of software engineering and command and control (C2) are very similar because they are both instances of complex problem solving. The common nemesis to successfully developing solutions in these environments is change. The challenge of any complex problem solving process is the balance of adapting to multiple changes while keeping focused on the overall desired solution. The Knowledge Sharing Mechanism (KSM) is proposed as framework to achieve this balance. The KSM is an iterative method for understanding a complex problem, developing a framework for solving that problem, developing partial solutions 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 C2. In image theory, there are two perspectives in developing a solution: topsight and insight. These two perspectives must be balanced in order to achieve success. Alexander's unfolding process is the basis for understanding, as an observer, the complex interactions in both software engineering and C2. The KSM uses Alexander's differentiation process, as an actor, achieving the correct balance of topsight and insight.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**56**	

Abstract

The environments of software engineering and command and control (C2) are very similar because they are both instances of complex problem solving. The common nemesis to successfully developing solutions in these environments is change. The challenge of any complex problem solving process is the balance of adapting to multiple changes while keeping focused on the overall desired solution. The Knowledge Sharing Mechanism (KSM) is proposed as framework to achieve this balance. The KSM is an iterative method for understanding a complex problem, developing a framework for solving that problem, developing partial solutions 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 C2. In image theory, there are two perspectives in developing a solution: topsight and insight. These two perspectives must be balanced in order to achieve success. Alexander's unfolding process is the basis for understanding, as an observer, the complex interactions in both software engineering and C2. The KSM uses Alexander's differentiation process, as an actor, achieving the correct balance of topsight and insight.

1 Introduction/Research Motivation

1.1 Research Motivation

The fields of software engineering and military command and control (C2) at first may not seem very similar. Upon closer inspection, however, similarities in the histories and challenges for both software engineering and C2 become very apparent. This leads to an obvious question of what exactly 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[19].

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[9] 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[8] also noted that "not only are changes in the objective inevitable, changes in the development strategy and technique are also inevitable." Based upon their similarities, both software engineering and C2 have a great possibility of learning from each other.

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" [12]. 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.

2 Seeking to Understand Design

2.1 History of Software Development Models

Software engineers sought to codify the software development process through the use of software development models. 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[22]. This methodology closely resembles the Army's Military Decision Making Process (MDMP). 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's goals and implementation were inevitable. Fred Brooks[8] noted that "not only are changes in the objective inevitable, changes in the development strategy and technique are also inevitable." Thus, software developers continued to look for a better model.

Barry Boehm, working at TRW Defense Systems Group during the mid-1908's, 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[7]. 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.

Many in the software engineering community saw both of these software development methodologies as too rigid and incapable to deal with the inevitable changes associated with producing software. They 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[6]. The desire for agility is mirrored in the 2006 Quadrennial Defense Review[10] when it stated "given the dynamics of change over time, we must develop a mix of agile and flexible capabilities to mitigate uncertainty."

One of the most dominant of the Agile Methods is eXtreme Programming (XP) which was developed by Kent Beck in the late 1990s. The classic model of the cost of implementing changes 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. XP seeks to achieve this flattened cost of change curve by adhering to five fundamental principles. Those principles are rapid feedback, assume simplicity, incremental change,embrace change, and quality work. By achieving the simplest solution possible, the developer gives himself or herself more flexibility to add to that code or change it[6].

2.2 Search for a New Paradigm

Not satisfied with the aforementioned software development models, software engineers began to look outside their discipline for inspiration for creating a new paradigm on how to design and develop good software. One of the fields looked at was architecture because it, like software engineering, seeks to create a solution to a complex and changing situation. A pivotal architect of our time is Christopher Alexander. Alexander was invited to give the keynote address to the 1996 Association for Computing Machinery (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[2] 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[5] 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.

2.3 Universe of Centers

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[3] 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[3]. Thus, understanding the power and interaction of centers to create wholeness is the 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[3]. 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.

2.4 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 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 1: Alexander's 15 Properties[3]

These properties of good design are not independent, discrete characteristics. Rather, these properties are interwoven[3]. 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.

2.5 Unfolding Process

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[3].

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[4].

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.

3 Image Theory

3.1 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[23].

Images, obviously, are a powerful tool for communicating thoughts, conditions, and ideas. In fact the U.S. Army *Operations* field manual[11] 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[12]:

- 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 or 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[6]. Even though a metaphor may not be a physical drawing, it still conveys a mental image. In fact, Christopher Alexander[4] 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."

3.2 Topsight Versus Insight

Military commanders must visualize the situation using three different perspectives[12]. 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[23].

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 amount of 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[23].

XP 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, called the driver, is responsible for physically typing in the code or drawing the design. The other programmer, called the navigator, is responsible for observing the work of the driver, looking for tactical or strategic defects[24]. The driver is entirely focused on the tactical design of the code. 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 with the balance of the driver's insight and the navigator's topsight can the pair programming team achieve their goals.

3.3 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 2 outlines this differentiation process.

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 2: Alexander's Differentiation Process[4]

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[4] 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. 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[4].

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.

4 The Knowledge Sharing Mechanism(KSM)

4.1 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[18]. 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[20]. 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 are[17]:

- **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.

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[17].

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[17]. The relationships between these two types of PDCA Cycles are shown in Figure 3.

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 4[17].

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[17]. Incorporating the PDCA Cycle with this 2x2 matrix, we have the KIM shown in Figure 5.

4.2 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 Do, the Sharer corresponds to Check, and the Finder corresponds to Act. The Framer and the Sharer are complementary roles

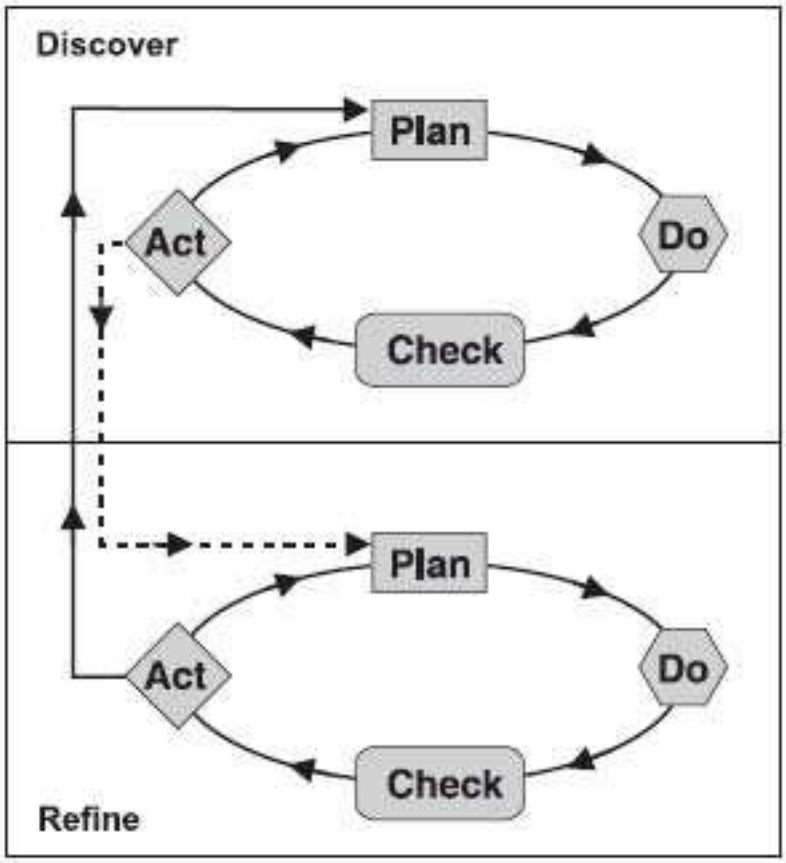


Figure 3: *Hoshin* and *Kaizen* PDCA Cycles[17]

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[16]. The Framer role in the KIM is shown in Figure 6.

The Maker is responsible for creating an innovative solution to the problem using the framework outlined by the Framer[16]. 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[17]. The Maker role in the KIM is shown in Figure 7.

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[16]. 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[17]. The Finder role in the KIM is shown in Figure 8.

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

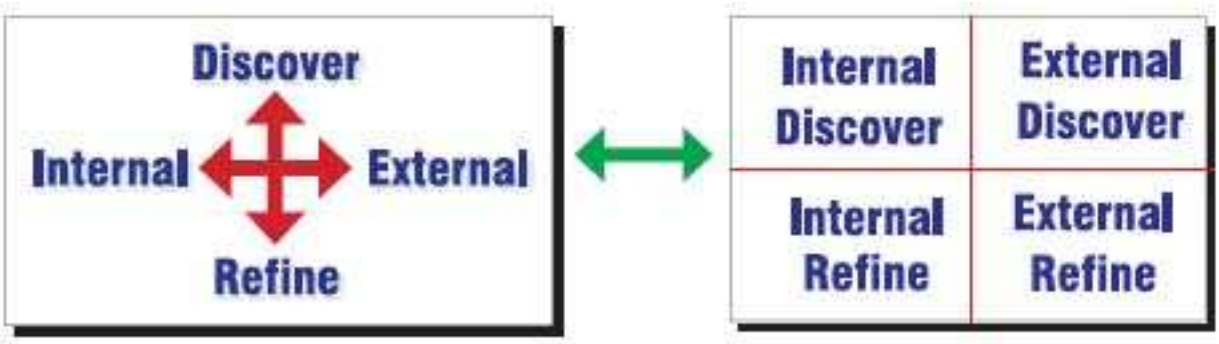


Figure 4: PDCA Cycle Stages of an Organization[17]

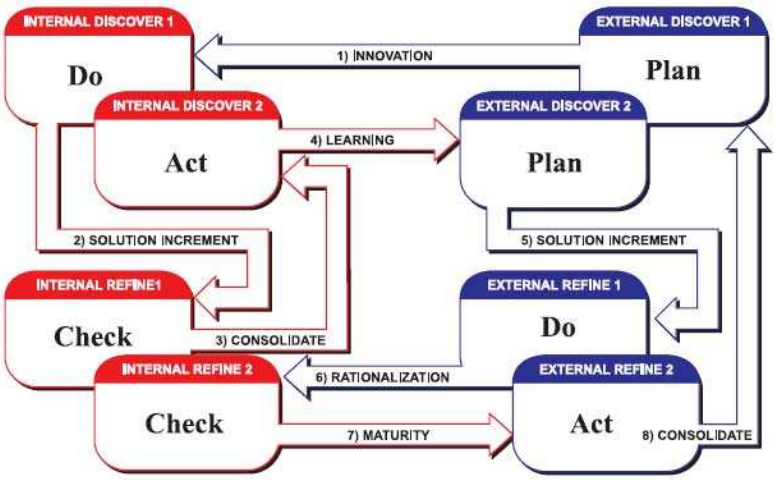


Figure 5: Knowledge Insight Model (KIM)[17]

the specific portion of the solution that is currently being developed or refined[16]. 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 9.

4.3 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 solve its portion of the overall problem.

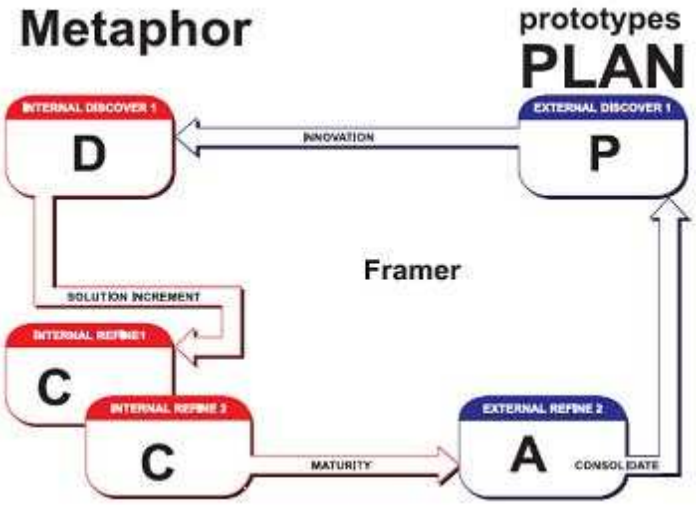


Figure 6: The Framer Role[17]

4.4 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.

4.5 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 the Knowledge

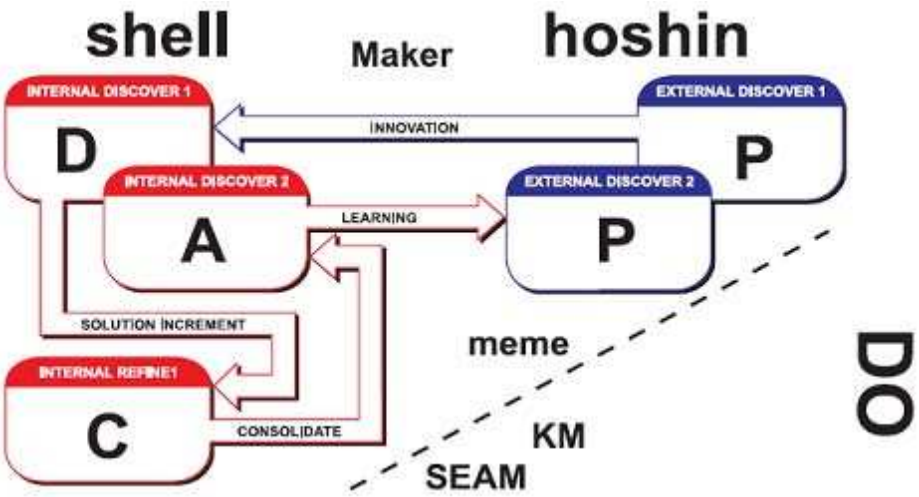


Figure 7: The Maker Role[17]

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 10.

The first step is to clarify what the problem is and what factors will affect the problem and it solution. This analysis of the problem leads to the development of a Solution Vision in the second step. The Solution Vision 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 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

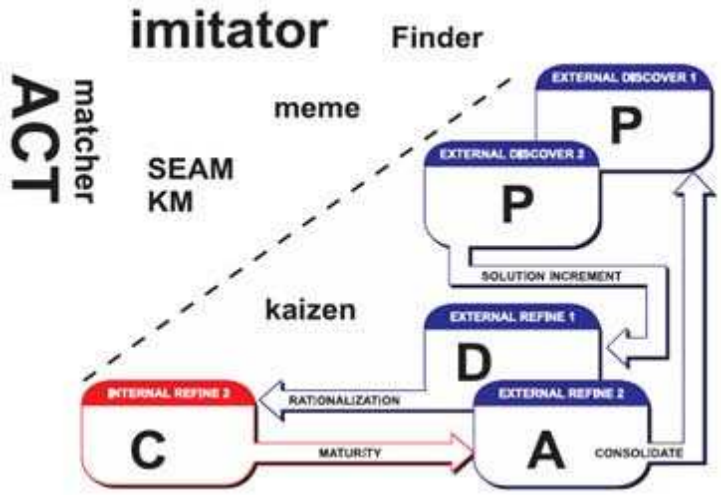


Figure 8: The Finder Role[17]

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.[15]

5 Incorporation of the KSM in C2

5.1 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 11.

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[13]. Those tasks are shown in Figure 12.

Mission analysis Tasks 1 through 11 in Figure 12 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 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

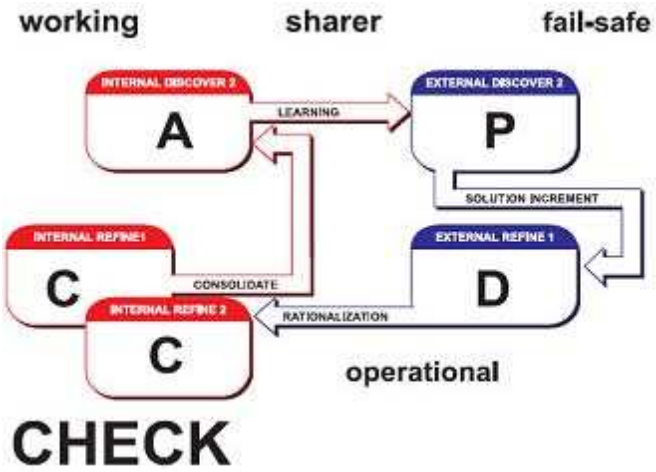


Figure 9: The Sharer Role[17]

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.

5.2 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

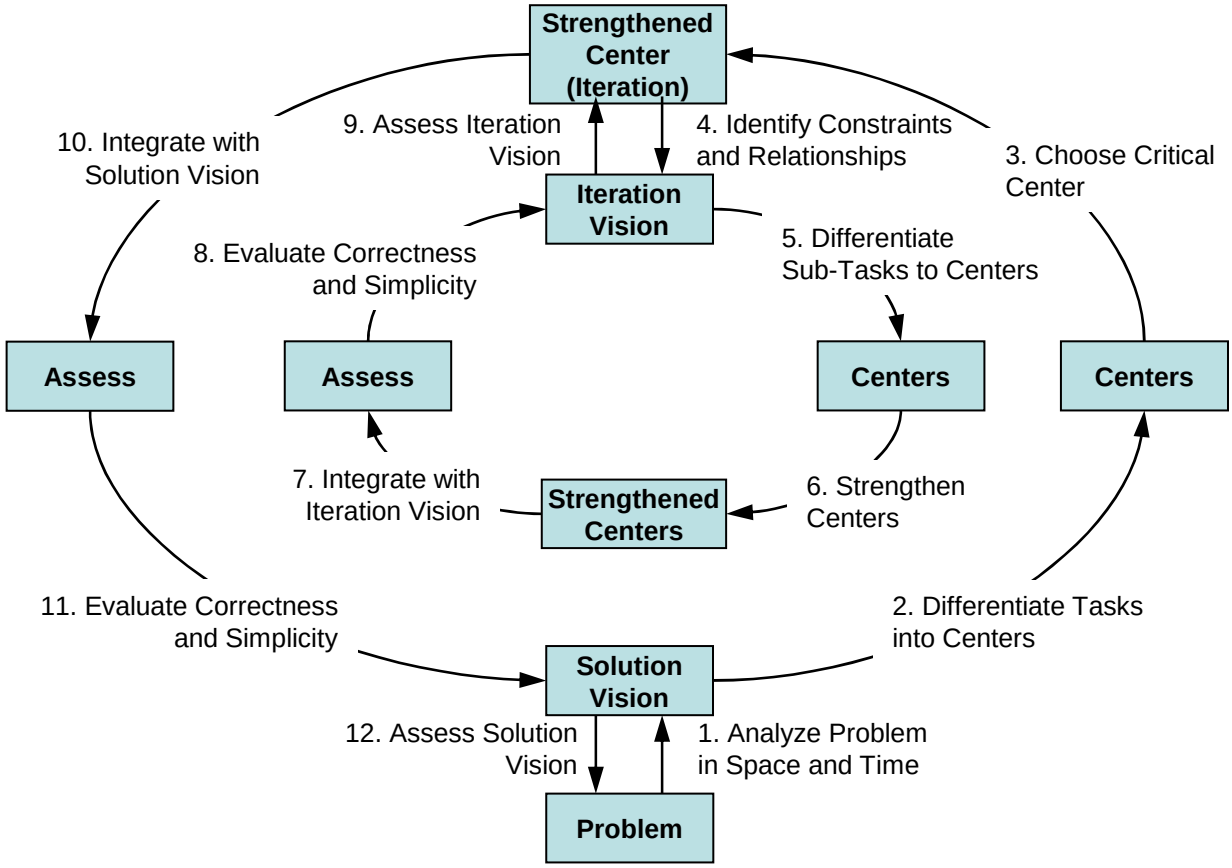


Figure 10: Knowledge Sharing Mechanism (KSM)[15]

United Kingdom. What does it mean for a command and control system to be agile? Dr. David Alberts[1] 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.

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 11: Military Decision Making Process (MDMP)[13]

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. 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[21] 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[14].

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

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 12: Mission Analysis Tasks[13]

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.

6 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.

By tying these two diverse fields to a higher level abstraction of complex problem solving, we gain insight into each environment by the lessons learned from the other environment. We also incorporated other philosophies of dealing with complex problem solving to further our understanding, such as 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).

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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

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

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

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

[10] 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.

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

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

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

[14] 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.

[15] Harvie, D. P. Knowledge sharing mechanism (ksm):a framework for software engineering and command and control. Master's thesis, North Carolina State University, 2006.

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

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

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

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

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

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

[22] 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.

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

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

Knowledge Sharing Mechanism: Enabling C2 to Adapt to Changing Environments

MAJ David P. Harvie

Department of Electrical Engineering & Computer Science

United States Military Academy

Agenda

- Introduction
- History of Software Engineering
- The Unfolding Process
- Image Theory and Differentiation
- Knowledge Insight Model (KIM)
- Knowledge Sharing Mechanism (KSM)
- Analysis of KSM
- Applying KSM to C2
- Conclusion

Introduction

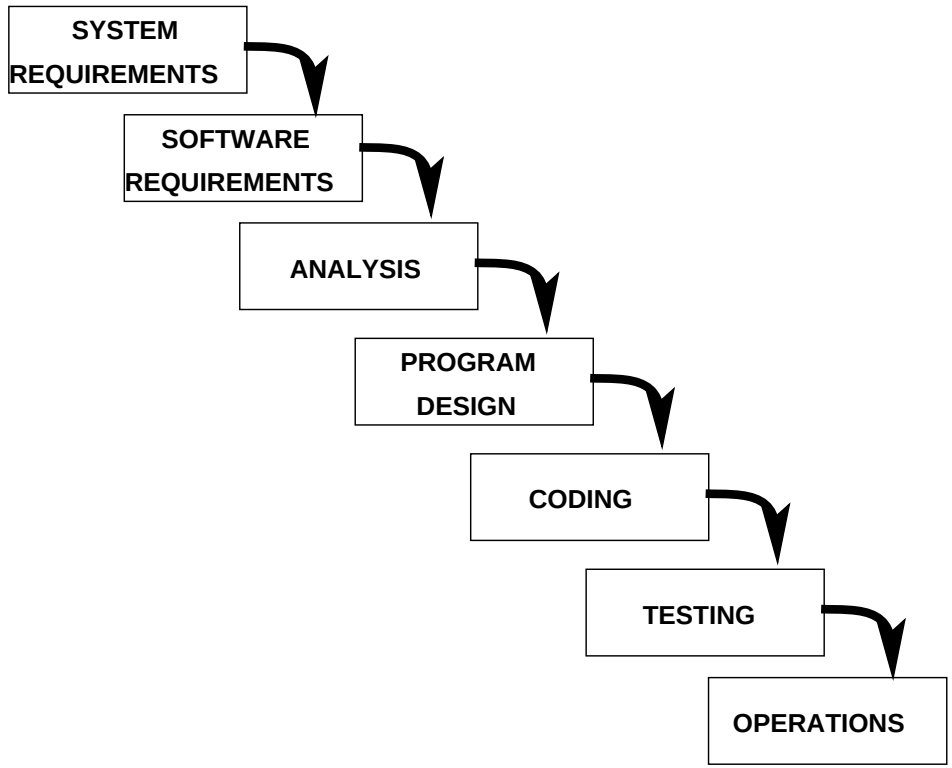
- Research Motivation – Similarity of Software Engineering and Command and Control
  - Instances of Problem Solving
  - Similar histories (Chaos → Hierarchy → Agile)
  - Change and Uncertainty

History of Software Engineering

- Software Engineering
  - Software was an afterthought to hardware, no structured method of development.
  - Growing software complexity resulted in projects being late and over budget (a growing crisis).
  - Phrase coined during 1968 NATO Conference.

History of Software Engineering

- Waterfall Method
  - Winston Royce (1970)
  - Derivative of traditional production methodology
  - Unable to accommodate change



- Agile Methods
  - XP, Scrum, Crystal
  - Agile Manifesto (2001)
  - Embrace change
  - Rapid development and customer feedback

The Unfolding Process

- Christopher Alexander
  - Architect by training, however he has influenced many other disciplines
  - Sought to understand and re-create good design
  - Key note speaker at the 1996 OOPSLA Conference
  - Works include *The Timeless Way of Building*, *A Pattern Language*, and *The Nature of Order* series

The Unfolding Process

- Wholeness and Centers
  - A design that has “life” must have a high degree of “wholeness”
  - Wholeness – “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.”
  - Centers – entities that contribute to the wholeness of a design; “a distinct physical system which occupies a certain volume in space, has a special marked coherence”

The Unfolding Process

- Wholeness and Centers
  - Example: Pond from *The Phenomenon of Life*, *Nature of Order*
  - Wholeness
  - Fuzzy boundaries

Photo from *Nature of Order*

The Unfolding Process

- Nature of Centers
  - Centers themselves have life.
  - Centers help one another; the existence and life of one center can intensify the life of another.
  - Centers are made of centers (recursion).
  - A structure gets its life according to the density and intensity of the centers which have been formed in it.

The Unfolding Process

- 15 Properties
  - Based on Alexander's research into natural and classical design
  - Interdependent, not independent
  - Used to understand and strengthen centers

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

The Unfolding Process

- The Unfolding Process
  - Third party (observer) perspective of design evolution
  - Step-by-step adaptation
  - 15 Properties become 15 Transformations
  - Example: Mouse forelimb development

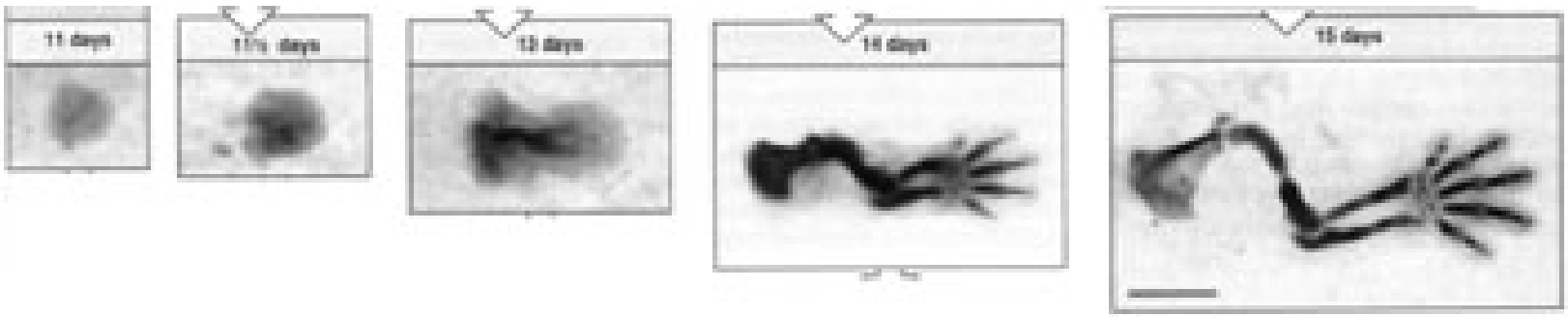


Photo from *Nature of Order*

Image Theory and Differentiation

- Alexander's Approach to Topsight and Insight
  - View of the design's wholeness = topsight
  - View of a particular center in the design = insight
- The Differentiation Process
  - First party (actor) perspective of design evolution
  - 4 Necessary Conditions:
    - Awareness of the whole
    - Step-by-step adaptation
    - Unpredictability
    - Feedback

Image Theory and Differentiation

- The Differentiation Process
  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.*

Image Theory and Differentiation

- The Differentiation Process (continued)
  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.*

Image Theory and Differentiation

- The Differentiation Process (continued)
  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.*

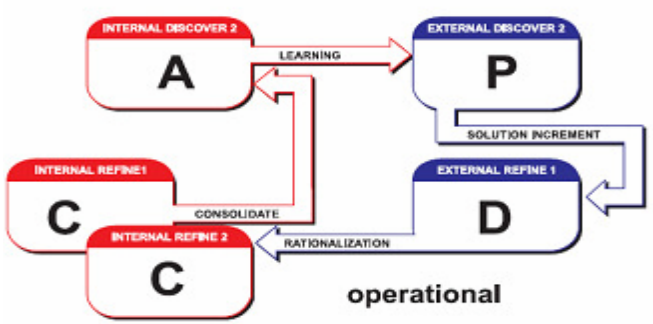
Image Theory and Differentiation

- The Differentiation Process (continued)
  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.*

Knowledge Insight Model (KIM)

- Four Roles (or Patterns) Emerge:
  - Framer: responsible of understanding the problem and designing an overall architecture (or framework) that will solve the problem.
  - Maker: responsible for creating an innovative solution to the problem using the given framework
  - Finder: responsible for finding resources to supplement the Maker's efforts
  - Sharer: responsible for ensuring that the Framer, Maker, and Finder work together by sharing information

working sharer fail-safe



CHECK

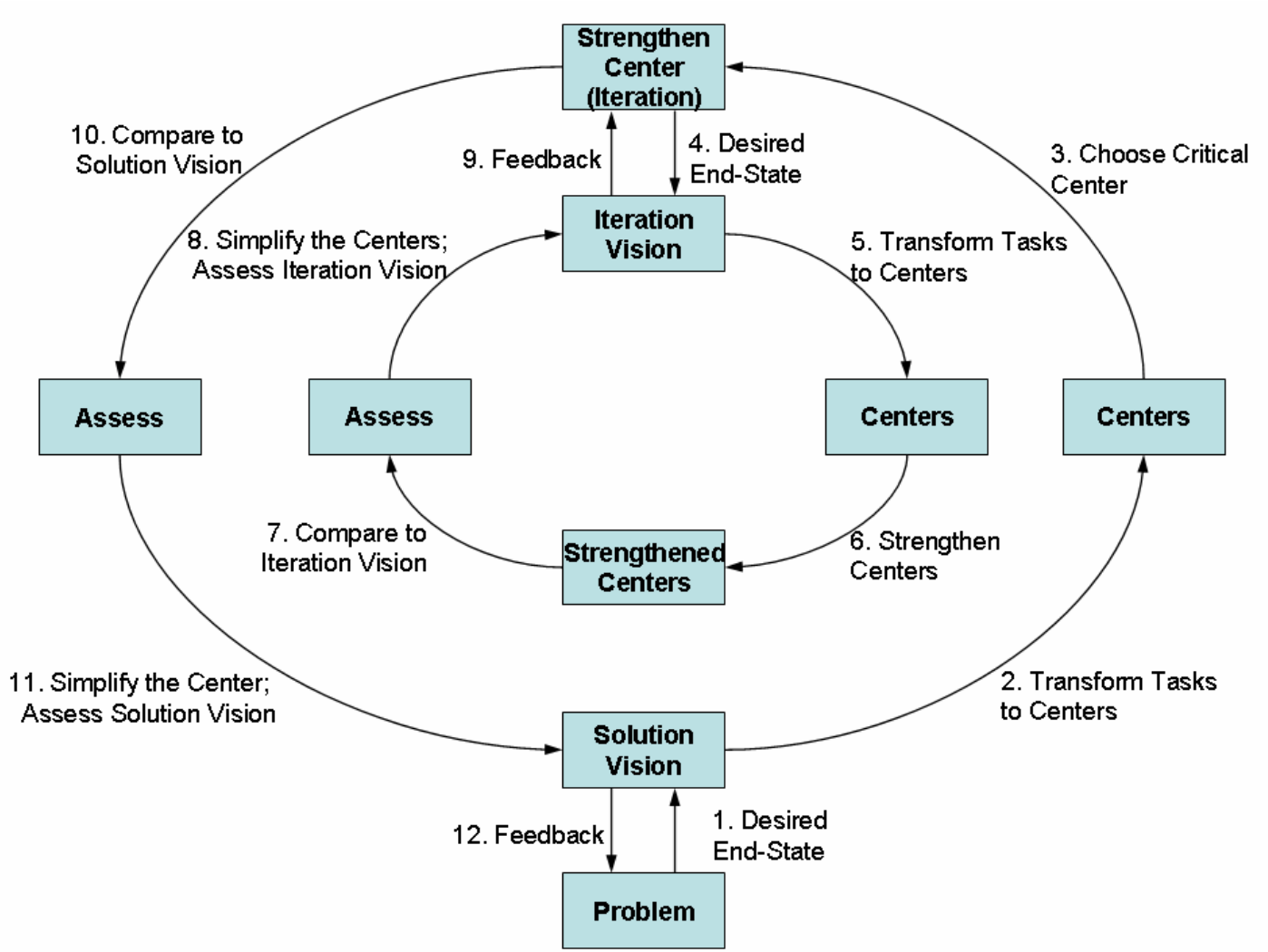
Knowledge Insight Model (KIM)

- KIM and Topsight versus Insight
  - The Framer uses Topsight
  - The Maker and the Finder use Insight
  - The Sharer must balance Topsight and Insight
- The Sharer's Role (or Inner Mechanism) is the most critical to solution's success
- The key must be the development of a knowledge sharing mechanism

Knowledge Sharing Mechanism (KSM)

- Utilizes the power of images
- Balances topsight and insight
- Synthesis of images and the Differentiation Process

Knowledge Sharing Mechanism (KSM)



Knowledge Sharing Mechanism (KSM)

1. Identify the Problem and the Problem's environment.
2. Develop a Solution Vision to solve the problem. The Solution Vision consists of the following: a desired end-state, critical tasks, and a purpose.
3. Begin designing the Solution by transforming each critical task into center
4. Begin a series of Iterations. At the beginning of each Iteration, identify the center which is the most critical at that moment.

Knowledge Sharing Mechanism (KSM)

5. Develop an Iteration Vision to improve/strengthen the chosen center. The Iteration Vision also consists of a desire end-state, critical tasks, and a purpose. It is imperative that the Iteration Vision supports the overall Solution Vision.
6. Improve/strengthen the chosen center.
7. At the end of the Iteration, assess whether the Iteration Vision has been fulfilled with the improvement of the center. Also assess whether the Iteration Vision is still valid and still supports the Solution Vision.
8. Repeat the Iteration process. Throughout the process, assess the Solution Vision to determine if it is still valid or does it need to be modified.

Analysis of KSM

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

Analysis of KSM

- KSM addresses:
  - **Robustness** by focusing on problem solving
  - **Resilience** through feedback mechanisms and simplifying
  - **Responsiveness** through continuous assessment and evolution
  - **Flexibility** by use of vision statements
  - **Innovation** through use of centers
  - **Adaptation** with its inherent step-by-step adaptation

Applying KSM to C2

- Integration of KSM into MDMP
  - MDMP consist of 7 steps:
    1. Receipt of Mission
    2. Mission Analysis
    3. Course of Action Development
    4. Course of Action Analysis
    5. Course of Action Comparison
    6. Course of Action Approval
    7. Orders Production
  - KSM applied to Mission Analysis step

Applying KSM to C2

- Mission Analysis
  1. Analyze the higher HQ's order
  2. Perform Initial Intelligence Preparation of the Battlefield
  3. Identify Specified, Implied, and Critical Tasks
- Integration of KSM
  - Visualize higher HQs' Solution Visions and develop unit's Solution Vision
  - Use IPB to understand the problem environment
  - Transform Critical Tasks into Centers

Applying KSM to C2

- Mission Analysis
  4. Review available assets
  5. Determine constraints
  6. Identify critical facts and assumptions
  7. Perform risk analysis
  8. Determine initial CCIR and EEFI
  9. Determine initial ISR plan
  10. Update the timeline
  11. Write the restated mission
- Integration of KSM
  - Perform successive Iteration Cycles developing each Critical Center of the Solution Vision
  - Compare progress to Iteration and Solution Visions
  - Brief the developed Solution Vision to the commander

Conclusion

- Software Engineering and Command and Control environments will grow more complex
- Successful solutions must accommodate change and unpredictability
- KSM provides a framework to accomplish these tasks using a synthesis of the Differentiation Process and image theory

Questions?