Understanding the Implementation of System Architectures in the Context of Distributed Cognition

Christopher B Watkins
Missouri University of Science and Technology
4570 Hidden Ridge Dr
Hudsonville, MI 49426 USA
chris@watkinsplace.com

Cihan H Dagli
Department of Engineering Management and Systems Engineering
Missouri University of Science and Technology
Rolla, MO 65409 USA
dagli@mst.edu

Abstract. Current system architectures are typically attributed to the technical leader that holds the formal title of "System Architect" and for larger projects there may be a team of system architects. There is no question that this person (or team) is responsible for the development and management of the system architecture. However, a theory in cognitive science known as "Distributed Cognition" challenges the notion that a single person or a single team can entirely understand the implemented architecture. This theory can aid in our understanding of system architectures and thus improve the probability of a successful implementation of an architecture. When applied to the domain of systems engineering, distributed cognition theory suggests that a system architecture can only be fully understood through the collective minds of the group that develops and implements the architecture. In fact, the group's understanding likely diverges from the understanding of any single individual. This helps explain why architectures implemented by a large group commonly deviate from the vision held by the lead system architect(s). After system architectures are understood as a distributed "system of knowledge", a systems engineer can actively manage the interfaces between distributed units of knowledge. The goal is to ensure that the collective system of understanding is a healthy system, and thereby increase the probability of success that the implemented architecture will meet the needs in the problem space. This paper summarizes current research in distributed cognition and applies that research to form heuristics that can be leveraged when developing and implementing an architecture.

The Theory of Distributed Cognition

The concepts of distributed cognition must be understood in order to understand how a system architecture moves from conception to implementation, and how to best manage that transition. Distributed cognition is a theory developed by psychologists studying cognitive science. Edwin Hutchins is credited with developing the theory in the mid-1980's. His study matured in 1995 and was published in a book entitled "Cognition in the Wild" (Hutchins 1995). He provided a detailed study of the cognition process embodied by an uncoordinated system of Navy ships entering and leaving a port. The captains of the ships acted independently but collectively developed an efficient system for port operations. This paper seeks to apply recent research in distributed cognition to the domain of systems engineering.

Psychology and systems engineering are not commonly studied together. However system

architecture and system design are a product of human thought, so the psychology of the human mind is relevant. In fact, cognitive science quickly converges with concepts in systems engineering. If the isolated knowledge maintained by an individual is understood in terms of its interfaces to knowledge held by another individual, then a "system of knowledge" is formed. As a "system" this becomes a problem that can be addressed using familiar methods of systems engineering.

Knowledge System Interfaces Rely on Mental and Physical Artifacts. A person's understanding is affected by both mental and physical artifacts related to knowledge. In other words, an individual's understanding is affected by their own internal thoughts, and by their physical environment that shape and filter their thoughts. For example, consider a person talking on the phone while driving a car filled with three small children that are fighting and screaming. The phone provides an interface to share information (mental artifacts). The driver's environment (physical artifacts) demands multiple cognitive tasks to be performed at the same time, and thus reduces the cognitive ability to process the information being presented on the phone. The driver is much less likely to comprehend complex stock market information being discussed by their financial advisor on the phone than if they were in a face-to-face meeting at the office that did not include the kids. Physical artifacts can significantly impact the cognition of information.

The same is true at the group level. A group's understanding is affected by the quantity and quality of individual thought shared between members of the group as well as by the environment that affects the perception of the shared information.

A generic representation of a knowledge system is shown in Figure 1.

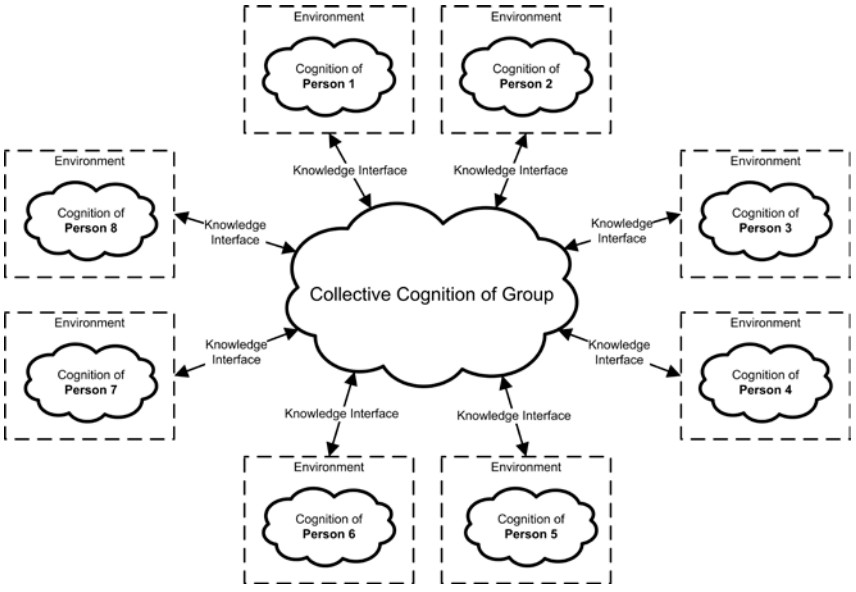

Figure 1. Generic Knowledge System Diagram

The distributed elements of cognition together form the collective cognition of the group. The main elements of a knowledge system include:

- Individual's local knowledge
- Individual's environment that affects (shapes) their cognitive process
- Interfaces between the distributed knowledge

Recent Study of Distributed Cognition. Recent work in the field of distributed cognition has been published by Gureckis and Goldstone at Indiana University (Gureckis and Goldstone 2006). The authors do not address system architectures directly, but they embrace distributed cognition theory and draw conclusions about conditions under which the behavior of individual agents self-organize into adaptive problem-solving group structures. Through a number of case study experiments, these researchers develop a set of lessons learned regarding the behavior of group-induced thought. The experiments addressed three topics of human cognition: Group Path Formation, Propagation of Innovations, and Human Foraging Behavior (Gureckis and Goldstone 2006). The experiments presented problems to a group of individuals to solve on their own computer, but allowed controlled communication with other individuals. The problem-solving performance of the group was measured while each person worked individually. In each of the experiments, the interfaces between individuals were varied in order to observe the effect on the group's collective performance.

Based on these experiments, the researchers develop four lessons learned (a.k.a. heuristics) regarding distributed cognition (Gureckis and Goldstone 2006):

1. **People are a large part of people's environment**

   People communicate indirectly by modifying their local environment. Within the study experiments, people followed the behavior of the group even though they were free to act on their own. People naturally took advantage of the behavior of others when devising their own actions. Therefore it can be concluded that people will think more as a group when they share more of their environment.

2. **People divide and conquer. As a group they explore the unknown while exploiting the known.**

   People take advantage of distributed knowledge to explore and exploit the problem domain. While traditional project management relies on delegation of tasks to subordinates, this study demonstrated conditions in which a group would spontaneously organize into an effective problem solving structure without centralized control. Therefore it can be concluded that there are factors outside of a group's formal organizational structure that contribute to results in the solution domain.

3. **More information isn't always better**

   The group's productivity is affected by the amount of information an individual has access to. Perhaps counter intuitively, the study demonstrated that *less* information encouraged independence between individual agents and resulted in greatly enhanced group performance when addressing a complex problem. This is because the scarcity of information encouraged more individuals to explore the problem space rather than exploit existing information. Therefore it can be concluded that there is a balance between (a) a lack of knowledge interfaces between individuals such that their collaboration as a group is lacking, and (b) providing too many knowledge interfaces between individuals such that the group's exploratory creativity is stifled.

4. **Groups are best influenced by bottom-up pressures rather than top-down rules**

   The study demonstrated that minor changes to the incentives of a few individuals can cause radical changes in the group's problem solving performance. This leads to the surprising

conclusion that collective behavior is potentially more controllable than isolated individual behavior. The most common form of crowd control is through direct orders or enforced laws of behavior. However it may be more effective to focus on the incentives for a few strategically-selected individuals in order to shape the behavior of the group with which they interact. This concept is not new to Chinese history. The Chinese "bao-jia" governance system relies on bottom-up pressures where every 10 households were led by a leader known as a "jia zhang" (Zhong 2003). These local leaders maintained personal relationships with their 10 households, and therefore could act as the government's hand in influencing politics at a grass-roots level.

A group that is implementing a system architecture can be characterized as a problem-solving group structure. Therefore the lessons learned from these studies form a set of heuristics that can be applied to the domain of systems engineering to help us understand how system architectures are implemented.

Applying Distributed Cognition within the Domain of Systems Engineering

If an implemented architecture is understood in the context of distributed cognition, then it can be concluded that the architecture is actually a result of the *collective understanding* of everyone that can affect the architecture. This begs the question, “Who can affect a system architecture?” Certainly the system architect, or architecture team, is the immediate conclusion. A secondary conclusion is that any individual that contributes to the design and implementation can affect the architecture. Therefore, according to this proposal, a system architecture that has been implemented is a product of the distributed cognition of the collective architects, system designers and implementers.

To highlight how an architecture can be passively impacted during implementation, consider an example electronic system architecture that relies on a shared data bus. The architect connected all systems to this common bus so that data is globally available to all systems, and new or modified systems can be achieved without impacting wiring. However an engineer implementing the architecture may not understand this concept and proceed to connect systems to separate busses in order to simplify data timing analyses. The architect is not the only person that affected the architecture. The engineer implementing the architecture can negatively impact the architecture by independently making a design decision with good intentions.

The knowledge systems that surround product architectures can occur through unintentional (wild) interfaces between knowledge units, or through intentional interfaces (Hutchins 1995). Unintentional knowledge systems can lead to unintentional product architectures that do not sufficiently address all customer needs. Therefore it is important to explicitly form the knowledge system.

Is Distributed Cognition Beneficial or Destructive? An interesting conclusion of distributed cognition research is that the collective understanding of the architecture is likely to be different from the system architect’s local cognition. Thus the implemented architecture is likely to be different than the architecture conceived by the system architects. This gap in understanding can be considered as beneficial or destructive. The individuals that work to implement the architecture

may have unique expertise and are able to apply that expertise to affect architectural change for its benefit. Conversely, not all individuals may understand the architecture and thereby destructively cause the implementation to deviate and lose qualities that are necessary to the solution space. It is a mistake for a system architect to respond to the potentially negative effects of distributed cognition by attempting to *force* everyone into a common understanding through micro-management. This approach limits creativity and original thought through authoritative mandates. Instead the system architect needs to embrace a balanced approach that leverages the beneficial contributions that result from collective brain power, while supplementing architectural authority as required to facilitate the decision process.

Requirements Decomposition is NOT a Sufficient Knowledge System. A focus on knowledge systems is lacking within current systems engineering practices. The knowledge system that tends to receive the greatest focus is the requirements database. Multiple levels (or "tiers") of requirements are created and decomposed into lower level detail. Requirements traceability between tiers forms a primary knowledge interface between requirement teams at different requirement levels. The system architects create the top tier(s) of requirements that document the customer need. Then they rely on the engineers implementing the architecture to create lower level requirements through requirements decomposition and traceability. These are good practices, but requirements decomposition is not a sufficient knowledge system. Engineers are able to change the architecture by implementing lower level requirements that passively impact characteristics of the intended architecture that could not be sufficiently defined in the high level requirements. Requirements databases should be a part of a knowledge system, but additional knowledge interfaces should also be employed to enhance the collective architectural understanding of the architects and implementation teams.

Recognizing the Cost of a Knowledge System. Heuristics that apply to product systems are also applicable to knowledge systems. A fifth heuristic that we can add to the four distributed cognition heuristics is:

5. **System cost increases proportionally to the increase in system complexity**

Complex systems are simply expensive, and this is no exception for knowledge systems. The minimal amount of knowledge interfaces required to do a job is the most efficient architecture. Each additional person (and thus knowledge interface) above the optimal number of interfaces in a knowledge system adds cost. Tausworthe published a productivity model that demonstrated that adding more workers to a project already containing an optimum number of workers will slow progress. Tausworthe's model also showed that "A team producing at the fastest rate humanly possible will spend half of its time coordinating and interfacing." (Tausworthe 1976). Fred Brooks created a concept known as "Brooks' Law" that states "Adding manpower to a late software project makes it later" (Brooks 1995). Both Tausworthe and Brooks were modeling the cost of the knowledge system interfaces.

This begs the question of how to identify the optimum number of interfaces for a knowledge system. It is not an exact science, but the real point is that it should be carefully and explicitly considered during architecture development. More is not always better. The optimal knowledge system in a solution space contains the minimum amount of people and artifacts necessary to

create the solution.

Two Knowledge Systems: Problem Space and Solution Space. A system architecture can be characterized by two systems of knowledge: the knowledge system in the problem space and in the solution space. Neither the problem nor the solution are rarely understood by one individual. Both the problem space and solution space are further studied in this paper to identify how architectures can be understood and implemented in the context of distributed cognition.

Distributed Cognition in the Problem Space

First the problem space is analyzed. Many proposals have been published for better characterizing the problem, or the customer's need. The theory of distributed cognition and the lessons learned from related studies help answer why some of those proposals are effective in better defining the problem space.

Recent work provided by Steven Saunders highlights the fact that the problem space is not well characterized within current systems engineering processes (Saunders 2007). Saunders proposes that too much emphasis is placed on requirements management, and the biggest value in systems engineering is to sufficiently characterize the problem space so that an appropriate architecture can be developed to successfully meet all needs. To this end Saunders proposes a SCARIT process model that places emphasis on the identification of all Stakeholders, their Concerns, and the related Architecture in addition to the traditional Requirements, Implementation and Test. The theory of distributed cognition is not part of Saunders' paper, but it certainly validates the focus on the stakeholders and their concerns in the problem space as impacted by their environment. A stakeholder that is a contracts manager is likely to understand most issues as a cost issue. Likewise, a product user is likely to understand most issues in the context of how they personally use the product, which may not be how others use the product. A holistic problem space analysis can be completed when all stakeholders and their environments are considered together.

Related work was provided by Trainor and Parnell in a paper entitled "Using Stakeholder Analysis to Define the Problem in Systems Engineering" (Trainor and Parnell 2007). The paper does not recognize distributed cognition, but instead provides practical techniques for providing a diverse perspective of the problem space. They claim that the initial problem statement is rarely the full statement of the problem since the problem space is not normally explored thoroughly. They proceed to recommend methods for exploring the problem space that include one-on-one interviews, focus groups, and mass surveys. In terms of distributed cognition, these represent different knowledge interfaces and physical environments that impact the level of comprehension during an information exchange. One-on-one interviews provide the greatest opportunity for comprehension, while mass surveys provide the least opportunity for comprehension but provide feedback from a large group in a minimal amount of time. Two knowledge interfaces that were not mentioned by Trainor-Parnell are the "documented problem space" and "co-located work environment". This represents two extremes of comprehension. A "documented problem space", such as that contained in a Request for Proposal (RFP), provides the least potential for comprehension. A "co-located work environment" provides the greatest potential for comprehension. The investigators of the problem space co-locate with the stakeholders and work together to (a) explore and understand the problem and (b) to characterize the environment that shapes the stakeholder's perception.

An example of a knowledge system for a problem space is shown in Figure 2. Knowledge interfaces discussed by Trainor and Parnell are used in this example. Example environments that can shape an individual's cognition process are also depicted.

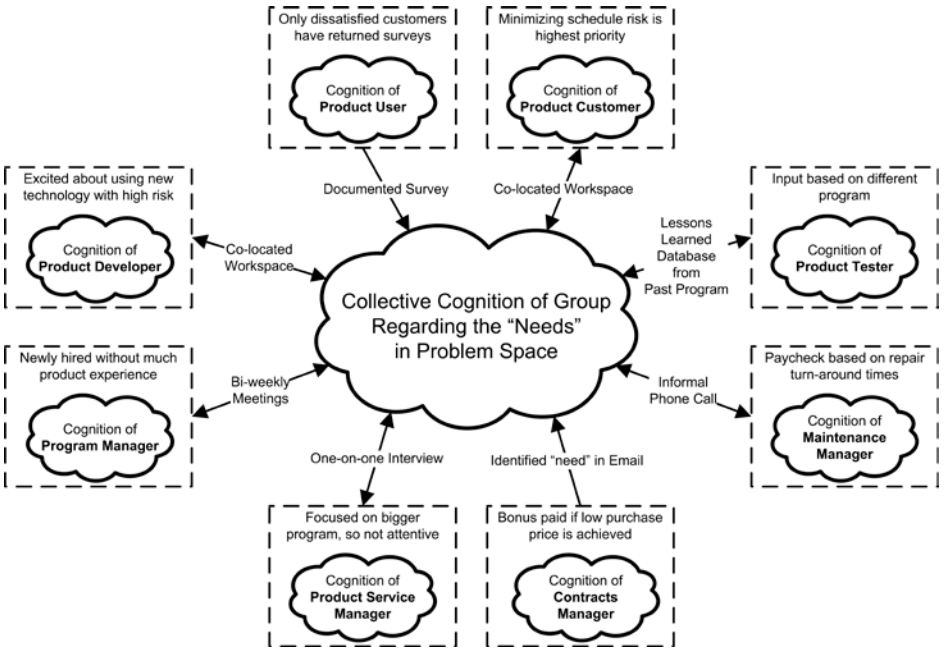

Figure 2. Knowledge System Diagram for Example Problem Space

Real-world examples of the knowledge interface extremes help to highlight how the interface and environment can impact the comprehension of the problem space. These two examples involve a reference company called "RefCo" that was proposing architectures during the formal proposal/bidding process. However the interfaces in the knowledge systems were very different for each example and that made the difference between an architectural failure and a success.

Example 1: ForeignCo was a company located in a country with a different language and culture than RefCo. ForeignCo sent a request for proposal (RFP) to company RefCo, but time zone and cultural differences made communication challenging, so there was virtually no personal communication to allow RefCo to explore and understand the problem space. This is an example of a "documented problem space". RefCo proposed an architecture to meet the needs that they understood from the written RFP, but ForeignCo quickly rejected the proposal since it did not actually meet the needs in their problem space. The interfaces between distributed sources of knowledge were neglected so the knowledge system in the problem space was unhealthy and led to architectural failure.

Example 2: ShareCo invited the same company, RefCo, to temporarily co-locate some of their system architects in the office space of ShareCo. This allowed for day-to-day interactions between the two companies so that they could learn from each other. Neither RefCo or ShareCo were fully cognizant of all ShareCo's needs in the problem space, but the two companies worked together to interface their distributed knowledge and increase their collective understanding of the problem space. Following this interaction, both companies worked together to co-author a Request for

Information (RFI) that RefCo formally responded to. Following the RFI, ShareCo authored an RFP for RefCo to respond to. By this time the problem space was very well understood, and RefCo was able to provide a proposed architecture to successfully meet all the needs of ShareCo.

These two examples exemplify the first heuristic "People are a large part of people's environments". ShareCo and RefCo shared their working environment to build effective interfaces between the distributed knowledge held by each company. ForeignCo and RefCo didn't share any of their environments, and it resulted in a failed architecture.

Proposal for Architecting a Knowledge System in the Problem Space

The five heuristics derived from the study of distributed cognition are helpful in understanding knowledge systems in the problem space. Each heuristic is applied below:

Heuristic #1: People are a large part of people's environment. When exploring a problem space, the architect will interrogate the known stakeholders using some sort of information gathering method (ie. surveys, focus groups, interviews). If the stakeholders are independent and do not share an environment, then the knowledge system may be disjoined or incomplete. One way to address this is to create a shared environment by bringing stakeholders together in a focus group session. This environment fosters knowledge interfaces between the stakeholders who are complete strangers. These interfaces will enhance communication and understanding about a problem. For example, it is more productive to put a driver and the mechanic in the same room to assess a problem with a broken car than it is to interview the driver and the mechanic separately. When they share an interface, they can more effectively share information that affects what questions are asked and answered (forms of information flow across an interface).

According to this heuristic, the cloud in the center of the knowledge system represented in Figure 2 actually includes a transfer of information between the outlying clouds of interconnected knowledge.

Heuristic #2: People divide and conquer. As a group they explore the unknown while exploiting the known. This heuristic provides insight into how interfaces are made between stakeholders in the problem space. According to this heuristic, if interfaces are provided between people, such as by bringing the stakeholders together in a focus group, then the participants will naturally divide and conquer the problem space. People that are motivated to contribute will not repeat what someone else already said about the problem. They exploit the known by indicating agreement, but then explore the non-communicated space in search of how they can contribute to the problem definition. If you do not put the stakeholders together, then individually they are more likely to state a similar set of obvious facts about the problem since they don't know what other people have contributed. When separated, the stakeholders are unable to explore and exploit as a group.

Heuristic #3: More information isn't always better. Consider the case when you interview each stakeholder individually and then distribute the transcripts of the interviews between everyone before bringing them together in a focus group. It will provide a lot of information to each individual stakeholder and thus create a false sense that the problem space is already fully explored. They are more likely to exploit the vast amount of information cited in the one-on-one

interviews rather than explore the unknown. Drowning people in information about a problem is a sure way to stifle the exploration of new information regarding the problem. A balance is needed to foster effective exploration and exploitation.

Heuristic #4: Groups are best influenced by bottom-up pressures rather than top-down rules. The stakeholders in the problem space are much more likely to exchange useful information if they understand the incentive for doing so. Bringing stakeholders together as part of a process without clear benefit to the stakeholder will likely result in a minimalist summary of the problem. The stakeholder declares "success" because they completed the process, not because they completed their exploration and exploitation of the problem space. If the stakeholder recognizes their personal benefit to solving the problem, then they are much more likely to exhaust all knowledge that they can contribute in defining the problem. For example, ask a teenager to describe the problem space surrounding their parent's choice of music on the car radio and they may simply utter "it is horrible". This does not describe the problem very well. One way to incentivize the teenager is to tell them you will change the music to their liking if they describe what's wrong with the current music. This may lead the teenager to divulge more information describing what they like, such as electric guitars and example bands that they prefer. It is important for the stakeholder to understand their incentive when defining the problem space.

Heuristic #5: System cost increases proportionally to the increase in system complexity. When defining the problem space, the architect should avoid interfacing with as many people as possible that are stakeholders in the problem. Rather, the goal should be to interface with one person from each stakeholder group. If a system architect believes that two people from one stakeholder group have unique perspectives, then technically those two people represent two different stakeholders even if they belong to the same organization. Recognizing the subtle differences between stakeholder groups will help the architect better understand their interface with the overall knowledge system. The management of redundant knowledge interfaces (two people from same stakeholder group) represents unnecessary cost. The architect must expend the time to collect and correlate the redundant input for a stakeholder perspective that is already known. In addition, an unbalanced number of representatives from a stakeholder group can misleadingly weigh the problem space.

PROPOSAL: Formally document the knowledge system in the problem space. The knowledge system is the foundation for cognitive understanding in the problem space and needs to be documented. It should include stakeholder identification, their environment, the knowledge interface used to gather their input, and a recording of their input. Many architects document a summary of the problem space. An analysis of the knowledge system validates that the problem space was thoroughly searched. The knowledge system can be reviewed to determine (a) if all stakeholders are represented, (b) if a stakeholder is overly weighted by redundant inputs, and (c) if the knowledge interface provided a sufficient transfer of information. Even if a system architect is working alone, this methodical approach to documenting the knowledge system may reveal an inadequacy that needs to be addressed before exploring the solution space.

A good place to document the knowledge system is in an "Architecture Description Document (ADD)". If a project doesn't already have such a document, then it can be created. This differs from a System Description Document since the ADD is limited to only describing the architecture

and does not describe the system implementation details. The knowledge system should be maintained within the ADD during the project lifecycle. History of changes to the knowledge system should also be recorded. If an architecture is modified to meet a differing stakeholder need, then the Architecture Description Document should record those changes and how the knowledge system defining the problem space was changed. For instance, a new stakeholder could have been identified in the project, or an existing stakeholder could have changed their mind, or the architect's organization could have directed a change to the architecture to ensure it remains aligned with their product roadmap (thereby instantiating themselves as a new stakeholder).

Distributed Cognition in the Solution Space

Once the problem space is well understood, the system architect and product implementation teams begin working in the solution space. This represents another knowledge system that needs to be managed. Much work has been published on the need to foster increased collaboration between project teams working in the solution space, especially teams that are geographically separated. A system architecture is defined by its interfaces, but the architecture is usually broken down at its interfaces when it is divided into organizational teams or outsourced. This begs the question of how the system architecture is understood and managed by the collective set of divided teams. Recent work was provided by Dagli, Miller and Abbott in a paper entitled "An Approach to a Network Centric Product Development System" (Dagli, Miller and Abbott 2007). The paper does not recognize distributed cognition, but instead provides practical techniques for managing knowledge interfaces in a collaborative development environment. Their proposal can be understood as a three faceted approach:

1. Provide common group environment through instant, continuous audio/visual communication via Two-way video over internet protocol (TVIP)
2. Provide effective interfaces between sources of knowledge through use of shared information databases
3. Affect positive group behaviors through use of a joint venture business model. This model integrates teams/organizations that are separated by utilizing integrated accounting methods, schedules, project management methods, tools and information technology.

The Dagli-Miller-Abbott proposal is analyzed in the context of distributed cognition to understand why it can be effective:

1. The TVIP initiative seeks to provide a shared working environment. This is directly related to the first heuristic: "People are a large part of people's environments."
2. The shared information databases are related to the second heuristic: "People divide and conquer. As a group they explore the unknown while exploiting the known." The shared databases provide a record of the "known" which can be exploited rather than re-explored. "Wiki" sites are an excellent example of a shared database that leverages the "divide and conquer" mentality (Leuf 2008).
3. The joint venture business model is related to the fourth heuristic: "Groups are best influenced by bottom-up pressures rather than top-down rules." A joint venture business model measures the collective performance of the integrated team through shared performance metrics (bottom up pressures). The businesses are accountable for each other.

An example of a knowledge system for a solution space is shown in Figure 3. Typical knowledge interfaces are shown along with example environments that can shape an individual's cognition process.

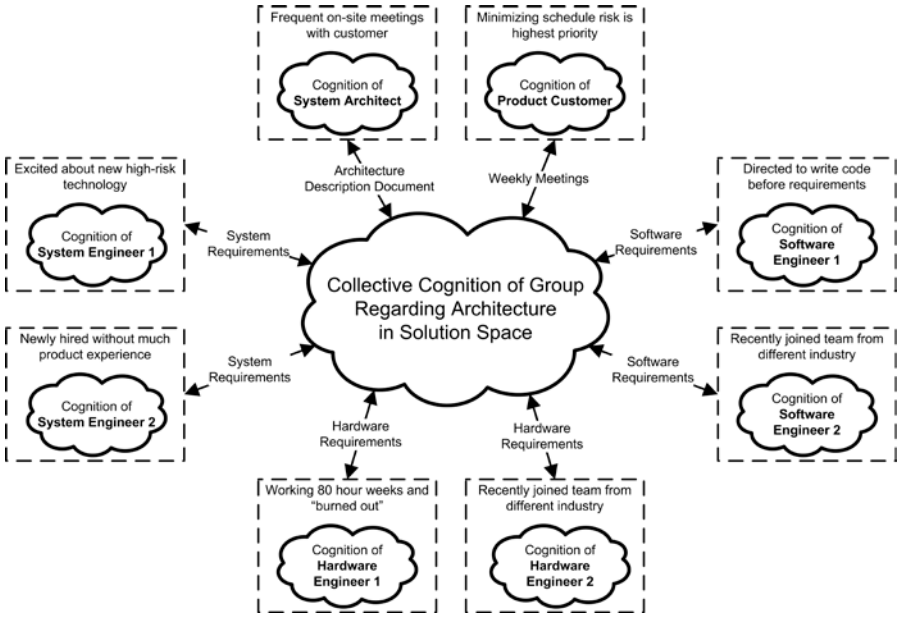

Figure 3. Knowledge System Diagram for Example Solution Space

Real-world examples of the knowledge system in the solution space are provided below to help highlight how the knowledge interface and environment can impact the comprehension in the solution space.

Example 1: RefCo created an architecture for a software verification tool that was supposed to locate two elements in a source model and then verify that all model elements between those two defined points existed in a reference model (strategy of minimal specification). The verification tool was contracted out to ForeignCo, which delivered a verification tool with a different architecture than intended. The tool simply verified that the two start and end model elements existed in the reference model instead of verifying that all elements existed *between* the two elements. The architecture had changed during implementation because the interfaces in the knowledge system of the solution space were insufficient. The architecture was verbalized by RefCo over the phone and wasn't formally documented anywhere. This was a poor knowledge interface, and the working environment was not shared between geographically separated RefCo and ForeignCo. In order to fix the architecture, RefCo formally documented the architecture and opened new knowledge interfaces through regular, frequent phone calls and emails. In addition, RefCo brought people from ForeignCo on-site in order to share a working environment. By fixing the knowledge system in the solution space, the architecture for the verification tool was successfully implemented.

Example 2: Toyota employs the "Big Room" concept as part of their Toyota Product Development System (TPDS). The "Big Room" concept leverages communication interfaces through the worker's environment. Closed offices and cubicles are removed and replaced with

large open spaces (a.k.a. The Big Room). The team leaders are located in the center of the room and teams are clustered around the leaders. The walls in the room are used to host visuals that support the development process. The open environment encourages frequent communication and fosters an environment of sharing ideas. By focusing on the knowledge interfaces between people and artifacts, the TPDS has achieved a highly efficient development process that is about four times shorter than that of a typical North American auto company (Cleveland 2006).

Proposal for Architecting a Knowledge System in the Solution Space

The five heuristics derived from the study of distributed cognition are helpful in understanding knowledge systems in the solution space. Each heuristic is applied below:

Heuristic #1: People are a large part of people's environment. When working in a solution space, the implementation of an architecture is more likely to succeed if the architects share a working environment with the team(s) implementing the architecture. The architects need to stay involved. By sharing a working environment, the architect is more likely to be exposed to the same pressures that can derail an architecture. In addition, a shared environment provides an interface for architects to be exposed to unique knowledge from experts on the system implementation team that could possibly reshape the architecture for the better.

Heuristic #2: People divide and conquer. As a group they explore the unknown while exploiting the known. Traditional project management relies on delegation of tasks to subordinates, but the referenced studies demonstrated that a group would spontaneously organize into an effective problem solving structure without centralized control. In order for this to occur, the group needs to know what others are working on so that they can exploit the "known". In contrast, the more that project teams are segregated from each other, the more that they will duplicate work by exploring the same solution space. In order to facilitate the "divide and conquer" mentality, it is crucial to foster healthy communication across teams. It is a mistake to purposefully segregate teams working in the same solution space.

Heuristic #3: More information isn't always better. System architects should avoid micro-managing the implementation team by providing comprehensive detail on how to implement an architecture. This serves to stifle creativity. There are many ways a given architecture can be implemented, and the team responsible for the implementation needs to be an owner of the solution. The architect(s) become the owner if they dictate an implementation. Since the architect may be held responsible for a failed architecture, it is in their best interest to remain involved without suffocating exploration in the solution space. The architect will benefit if they recognize that their implementation team brings unique experience to the table.

Heuristic #4: Groups are best influenced by bottom-up pressures rather than top-down rules. A healthy knowledge system requires good communication interfaces between all people working together. If the performance of a person (or team) is evaluated based solely on their own progress, then there are no incentives to build strong communication channels with other people (or teams) that depend upon their work. A top-down pressure model where management directs each team to increase communications through regularly scheduled meetings is likely to result in the typical meeting where everyone reports their status (perhaps round-robin) while rarely acting upon anyone's status. According to this heuristic, a bottom-up pressure approach would be much

more effective. For example, each individual member (or team) could also be evaluated on how well their dependents (people or teams) are able to interface with and/or use their work product. If a hardware engineer's personal evaluation was dependent upon whether the software team could successfully integrate with their hardware, then they would be much more likely to build healthy communications with the software team.

Heuristic #5: System cost increases proportionally to the increase in system complexity. When working in the solution space, one should avoid throwing as many people at the solution as possible. The productivity models created by (Tausworthe 1976) and (Brooks 1995) prove that a knowledge system that contains more elements than optimal will actually slow down the progress of the group due to the burden of coordination and interfacing. Therefore the goal should be to minimize complexity of the knowledge system by minimizing the number of people necessary to implement the architecture in the solution space.

PROPOSAL: Formally document the knowledge system in the solution space. The knowledge system in the solution space directly affects the implementation of the architecture and therefore needs to be documented. The documentation of the knowledge system in the solution space follows a similar structure as with the problem space. It must identify the knowledge stakeholders, interfaces (requirements documents, weekly telecoms, etc), and the environment that affects architectural perception. The knowledge system in the solution space should be reviewed to determine (a) if all stakeholders are represented, (b) if a sufficient knowledge interface is provided for each stakeholder, and (c) if the environment needs to be modified to improve the efficiency of the knowledge interface. For instance, if a team is working 80-hour weeks and suffers from burn-out, then their comprehension and communication of information will be reduced. If 80-hour work-weeks cannot be avoided, then the architect will likely need to modify the knowledge interfaces to ensure the team remains on-track.

The architectural knowledge system in the solution space should be documented together with the problem space in the proposed document referred to as the "Architecture Description Document (ADD)". History of architectural changes attributed to the knowledge system should also be recorded. For example, a change could be attributed to expertise of a system architect or other individual on the implementation team that affected positive change during the architecture's implementation. Alternatively, a negative change could be attributed to a misunderstanding by a team member. Either way, it is important to document the change and why the change was made. This artifact will serve as an input to the knowledge system for the current project. It can also serve follow-on projects that desire to fix architectural modifications that were due to a misunderstanding within the implementation team.

Conclusions

Many system architects may not realize that their architectures are driven by characteristics of distributed cognition. Those architects can improve the success rate of an architectural implementation if they actively manage the distributed cognition that occurs in both the problem and solution space. The first step is to recognize that knowledge and understanding exist as distributed entities. Once that is realized, the architect can rely upon familiar systems engineering methods to actively manage the distributed knowledge in the "knowledge system."

This paper has already presented five heuristics derived from recent studies in distributed cognition that help an architect manage knowledge systems. In closing, a sixth heuristic is presented that helps the system architect to begin thinking about how to construct the knowledge systems for their architectures. This heuristic is credited to David Stanislaw, a Federal Aviation Administration (FAA) Designated Engineering Representative (DER):

Heuristic #6: It is better to design how a system will fail as opposed to invent how it will perform or operate.

The first point of this heuristic is that the system architect needs to design their knowledge system by first identifying the failure conditions. If the architect doesn't understand the failure modes, then they won't understand which weaknesses need to be addressed in their knowledge systems.

The second point of the heuristic is that the architect needs to design a knowledge system that continues to work when there is a failure. The world is not a utopia, and systems are almost always a compromise. The architect will not always have the authority to correct a failure. For example, not all stakeholders in the problem space may be willing (or allowed) to actively participate in the knowledge system. In the solution space, bureaucratic pressures could create organizational firewalls that severely constrain the knowledge transfer between segregated teams at different geographical locations. If the architect doesn't have the authority to fix a failure mode, then they should design a knowledge system that continues to work *as best as it can* in the midst of the failure. Good communication and sound architectural understanding rarely happen by accident.

This paper does not present a specific process for managing a knowledge system, nor a specific framework for documenting a knowledge system. Instead, the intent of the paper is to raise awareness of distributed cognition within the community of system architects. Once a system architect understands and owns these principles, then they will be better able to manage and leverage the collective understanding of the entire implementation team. This serves to improve the success rate for implementing architectures.

References

Brooks, F. P., Jr. 1995. *The Mythical Man-Month: Essays on Software Engineering.* Reading, MA: Addison-Wesley.

Cleveland, John. 2006. The Toyota Product Development System's Implementation Challenges. Field Guide to Automotive Technology. Website accessed on 20 March 2010: http://www.autofieldguide.com/columns/0506insight.html

Dagli, Dr., Dr. Ann Miller, and Russell Abbott. 2007. An Approach to a Network Centric Product Development System. In Proceedings of the 17th Annual International Symposium of the International Council on Systems Engineering (San Diego, CA). Seattle: INCOSE.

Gureckis, Todd M., and Robert L. Goldstone. 2006. Thinking in Groups. *Pragmatics & Cognition*. 14:2:293-311. Amsterdam: John Benjamins Publishing Company.

Hutchins, Edwin. 1995. *Knowledge in the Wild.* Cambridge: MIT Press.

Leuf, Bo, Ward Cunningham, 2008, What is Wiki. Website accessed on 7 December 2008: http://www.wiki.org/wiki.cgi?WhatIsWiki.

Saunders, Steven. 2007. Promoting the Real Value of Systems Engineering using an Extended SCARIT Process Model. In Proceedings of the 17th Annual International Symposium of the International Council on Systems Engineering (San Diego, CA). Seattle: INCOSE.

Tausworthe, R. C. 1976. Simple Intuitive Models of Programming. Deep Space Network Progress Report 42-33. Jet Propulsion Laboratory. Pasadena, CA.

Trainor, Timothy E. and Gregory S. Parnell. 2007. Using Stakeholder Analysis to Define the Problem in Systems Engineering. In Proceedings of the 17th Annual International Symposium of the International Council on Systems Engineering (San Diego, CA). Seattle: INCOSE.

Zhong, Yang. 2003. *Local Government and Politics in China: Challenges from Below*. Armonk, NY: M.E. Sharpe.

Biography

Christopher Watkins is a graduate student at the Missouri University of Science and Technology. This paper is a result of his work in pursuit of an MS degree in Systems Engineering. He received his BS degree in Electrical Engineering from Kettering University. Chris was a founding employee of Michigan Aerospace, a remote-sensing research and development company. He is currently a system architect at GE Aviation Systems. He has over 10 years of experience with system design and development in avionics systems. He has led research in the areas of space robotics, orbital semiconductor manufacturing facilities and the development of optical air data systems based on ultraviolet LIDAR remote sensing technologies. Most recently Chris has contributed to the architectural development of Integrated Modular Avionics (IMA) systems that include the Boeing 787 Common Core System and the GE Aviation Genesis computing platforms. He has published 10 papers and is an author in the book: Digital Avionics Handbook, Second Edition (CRC Press).

Cihan Dagli is a Professor of Engineering Management and Systems Engineering, and Affiliated Professor of Electrical and Computer Engineering at the Missouri University of Science and Technology. He received BS and MS degrees in Industrial Engineering from the Middle East Technical University and a Ph.D. in Applied Operations Research in Large Scale Systems Design and Operation from the University of Birmingham, United Kingdom, where from 1976 to 1979 he was a British Council Fellow. His research interests are in the areas of Systems Architecting and Engineering, System of Systems, Smart Engineering System Design, Computational Intelligence: Neural Networks-Fuzzy Logic-Evolutionary Programming.

He is the founder and Boeing Coordinator of the Missouri S&T's System Engineering graduate program. Dr. Dagli is the director of Smart Engineering Systems Laboratory and a Senior Investigator in DoD Systems Engineering Research Center-URAC. He is an INCOSE Fellow 2008 and IIE Fellow 2009. He has been the PI, co-PI, or director of 46 research projects and grants totaling over $29 million from federal, state, and industrial funding agencies

Dr. Dagli is the Area editor for Intelligent Systems of the International Journal of General Systems, published by Taylor and Francis, and Informa Inc. He has published more than 350 papers in refereed journals and proceedings, 21 edited books and cited 900 times. He has consulted with various companies and international organizations including The Boeing Company, AT&T, John Deere, Motorola, U.S. Army, UNIDO, and OECD.